DeePMD-kit

DeepModeling

May 08, 2024

GETTING STARTED

1 Getting Started 3
- 1.1 Easy install . . . 3
  - 1.1.1 Install off-line packages . . . 3
  - 1.1.2 Install with conda . . . 4
  - 1.1.3 Install with docker . . . 5
  - 1.1.4 Install Python interface with pip . . . 5
- 1.2 DeePMD-kit Quick Start Tutorial . . . 6
  - 1.2.1 Task . . . 6
  - 1.2.2 Background . . . 6
  - 1.2.3 Practice . . . 7

2 Backend 27
- 2.1 Supported backends . . . 27
  - 2.1.1 TensorFlow . . . 27
  - 2.1.2 PyTorch . . . 27
  - 2.1.3 DP . . . 28
- 2.2 Switch the backend . . . 28
  - 2.2.1 Training . . . 28
  - 2.2.2 Inference . . . 28
- 2.3 Convert model files between backends . . . 29

3 Installation 31
- 3.1 Easy install . . . 31
  - 3.1.1 Install off-line packages . . . 31
  - 3.1.2 Install with conda . . . 32
  - 3.1.3 Install with docker . . . 32
  - 3.1.4 Install Python interface with pip . . . 33
- 3.2 Install from source code . . . 33
  - 3.2.1 Install the Python interface . . . 34
  - 3.2.2 Install the C++ interface . . . 38
- 3.3 Install from pre-compiled C library . . . 41
  - 3.3.1 Use Pre-compiled C Library to build the LAMMPS plugin, i-PI driver, and GROMACS patch . . . 41
- 3.4 Install LAMMPS . . . 42
  - 3.4.1 Install LAMMPS's DeePMD-kit module (built-in mode) . . . 42
  - 3.4.2 Install LAMMPS (plugin mode) . . . 43
- 3.5 Install i-PI . . . 44
- 3.6 Install GROMACS with DeepMD . . . 44
  - 3.6.1 Patch source code of GROMACS . . . 44
  - 3.6.2 Compile GROMACS with deepmd-kit . . . 45

3.7 Building conda packages . . . 45
3.8 Install Node.js interface . . . 46
3.8.1 Install from npm . . . 46
3.8.2 Build from source . . . 46
3.9 Easy install the latest development version . . . 46
3.9.1 Install with docker . . . 47
3.9.2 Install with pip . . . 47
3.9.3 Download pre-compiled C Library . . . 47
3.9.4 Pre-release conda-forge packages . . . 47

4 Data 49
4.1 System . . . 49
4.2 Formats of a system . . . 51
4.2.1 NumPy format . . . 51
4.2.2 HDF5 format . . . 52
4.2.3 Raw format and data conversion . . . 52
4.3 Prepare data with dpdata . . . 53

5 Model 55
5.1 Overall . . . 55
5.1.1 Theory . . . 55
5.1.2 Instructions . . . 55
5.2 Descriptor "se_e2_a" . . . 57
5.2.1 Theory . . . 58
5.2.2 Instructions . . . 58
5.3 Descriptor "se_e2_r" . . . 60
5.3.1 Theory . . . 61
5.3.2 Instructions . . . 61
5.4 Descriptor "se_e3" . . . 62
5.4.1 Theory . . . 62
5.4.2 Instructions . . . 62
5.5 Descriptor "se_atten" . . . 63
5.5.1 DPA-1: Pretraining of Attention-based Deep Potential Model for Molecular Simulation 63
5.5.2 Theory . . . 63
5.5.3 Introduction to new features of DPA-1 . . . 64
5.5.4 Data format . . . 67
5.5.5 Training example . . . 68
5.6 Descriptor DPA-2 . . . 68
5.7 Descriptor "hybrid" . . . 69
5.7.1 Theory . . . 70
5.7.2 Instructions . . . 70
5.8 Determine `sel` . . . 70
5.9 Fit energy . . . 71
5.9.1 Theory . . . 72
5.9.2 The fitting network . . . 73
5.9.3 Loss . . . 73
5.10 Fit spin energy . . . 75
5.10.1 Spin . . . 76
5.10.2 Spin Loss . . . 77
5.10.3 Data preparation . . . 77
5.11 Fit tensor like Dipole and Polarizability . . . 78
5.11.1 Theory . . . 79
5.11.2 The fitting Network . . . 80
5.11.3 Loss . . . 81

5.11.4 Training Data Preparation . . . 81
5.11.5 Train the Model . . . 82
5.12 Fit electronic density of states (DOS) . . . 83
5.12.1 The fitting Network . . . 83
5.12.2 Loss . . . 84
5.12.3 Training Data Preparation . . . 84
5.12.4 Train the Model . . . 84
5.12.5 Test the Model . . . 85
5.13 Type embedding approach . . . 86
5.13.1 Theory . . . 86
5.13.2 Instructions . . . 87
5.14 Descriptor "se_a_mask" . . . 88
5.15 Deep potential long-range (DPLR) . . . 89
5.15.1 Theory . . . 90
5.15.2 Train a deep Wannier model for Wannier centroids . . . 90
5.15.3 Train the DPLR model . . . 91
5.15.4 Molecular dynamics simulation with DPLR . . . 91
5.16 Deep Potential - Range Correction (DPRc) . . . 95
5.16.1 Theory . . . 96
5.16.2 Training data . . . 96
5.16.3 Training the DPRc model . . . 96
5.16.4 Run MD simulations . . . 99
5.16.5 Pairwise DPRc . . . 99
5.17 Linear model . . . 101
5.18 Interpolation or combination with a pairwise potential . . . 102
5.18.1 Theory . . . 103
5.18.2 Table file . . . 103
5.18.3 Interpolation with a short-range pairwise potential . . . 104
5.18.4 Combination with a pairwise potential . . . 104

6 Training 105
6.1 Train a model . . . 105
6.2 Advanced options . . . 106
6.2.1 Learning rate . . . 106
6.2.2 Training parameters . . . 107
6.2.3 Options and environment variables . . . 109
6.2.4 Adjust `sel` of a frozen model . . . 110
6.3 Training Parameters . . . 111
6.4 Parallel training . . . 161
6.4.1 TensorFlow Implementation . . . 161
6.4.2 PyTorch Implementation . . . 163
6.5 Multi-task training . . . 165
6.5.1 Theory . . . 165
6.5.2 Perform the multi-task training . . . 165
6.5.3 Initialization from pre-trained multi-task model . . . 166
6.5.4 Share layers among energy fitting networks . . . 167
6.6 Multi-task training . . . 168
6.6.1 Theory . . . 168
6.6.2 Perform the multi-task training using PyTorch . . . 168
6.6.3 Finetune from the pre-trained multi-task model . . . 171
6.7 TensorBoard Usage . . . 172
6.7.1 Highlighted features . . . 172
6.7.2 How to use Tensorboard with DeePMD-kit . . . 172
6.7.3 Examples . . . 173

6.7.4 Attention . . . 177
6.8 Known limitations of using GPUs . . . 177
6.9 Finetune the pre-trained model . . . 178
6.9.1 TensorFlow Implementation . . . 178
6.9.2 PyTorch Implementation . . . 179

7 Freeze and Compress 183
7.1 Freeze a model . . . 183
7.2 Compress a model . . . 184
7.2.1 Theory . . . 184
7.2.2 Instructions . . . 185

8 Test 189
8.1 Test a model . . . 189
8.2 Calculate Model Deviation . . . 190
8.2.1 Theory . . . 190
8.2.2 Instructions . . . 190
8.2.3 Relative model deviation . . . 191

9 Inference 193
9.1 Python interface . . . 193
9.1.1 External neighbor list algorithm . . . 194
9.2 C/C++ interface . . . 194
9.2.1 C++ interface . . . 194
9.2.2 C interface . . . 195
9.2.3 Header-only C++ library interface (recommended) . . . 196
9.3 Node.js interface . . . 197

10 Command line interface 199
10.1 Named Arguments . . . 199
10.2 Valid subcommands . . . 199
10.3 Sub-commands . . . 200
10.3.1 transfer . . . 200
10.3.2 train . . . 200
10.3.3 freeze . . . 201
10.3.4 test . . . 202
10.3.5 compress . . . 203
10.3.6 doc-train-input . . . 204
10.3.7 model-devi . . . 205
10.3.8 convert-from . . . 206
10.3.9 neighbor-stat . . . 207
10.3.10 train-nvnmd . . . 207
10.3.11 gui . . . 208
10.3.12 convert-backend . . . 209

11 Integrate with third-party packages 211
11.1 Use deep potential with dpdata . . . 211
11.2 Use deep potential with ASE . . . 211
11.3 Run MD with LAMMPS . . . 212
11.3.1 units . . . 212
11.3.2 Enable DeePMD-kit plugin (plugin mode) . . . 212
11.3.3 pair_style `deepmd` . . . 212
11.3.4 Compute tensorial properties . . . 214
11.3.5 Long-range interaction . . . 215
11.3.6 Use of the centroid/stress/atom to get the full 3x3 “atomic-virial” . . . 215

11.3.7 Computation of heat flux . . . 215
11.4 Run path-integral MD with i-PI . . . 216
11.5 Running MD with GROMACS . . . 217
11.5.1 DP/MM Simulation . . . 217
11.5.2 All-atom DP Simulation . . . 219
11.6 Interfaces out of DeePMD-kit . . . 220
11.6.1 OpenMM plugin for DeePMD-kit . . . 220
11.6.2 Amber interface to DeePMD-kit . . . 220
11.6.3 CP2K interface to DeePMD-kit . . . 220
11.6.4 DP-GEN . . . 220
11.6.5 MLatom . . . 220
11.6.6 ABACUS . . . 221

12 Use NVNMD 223

13 FAQs 225
13.1 How to tune Fitting/embedding-net size ? . . . 225
13.1.1 Al2O3 . . . 225
13.1.2 Cu . . . 226
13.1.3 Water . . . 227
13.1.4 Mg-Al . . . 228
13.2 How to control the parallelism of a job? . . . 229
13.2.1 MPI (optional) . . . 229
13.2.2 Parallelism between independent operators . . . 229
13.2.3 Parallelism within an individual operators . . . 229
13.2.4 Tune the performance . . . 230
13.3 Do we need to set rcut < half boxsize? . . . 230
13.4 How to set sel? . . . 230
13.5 Installation . . . 231
13.5.1 Inadequate versions of gcc/g++ . . . 231
13.5.2 Build files left in DeePMD-kit . . . 231
13.6 The temperature undulates violently during the early stages of MD . . . 231
13.7 MD: cannot run LAMMPS after installing a new version of DeePMD-kit . . . 231
13.8 Model compatibility . . . 232
13.9 Why does a model have low precision? . . . 232
13.9.1 Data . . . 232
13.9.2 Model . . . 233
13.9.3 Training . . . 233

14 Find DeePMD-kit C/C++ library from CMake 235

15 Create a model in TensorFlow 237
15.1 Design a new component . . . 237
15.2 Register new arguments . . . 237
15.3 Package new codes . . . 238

16 Create a model in PyTorch 239
16.1 Design a new component . . . 239
16.1.1 New descriptors . . . 239
16.1.2 New fitting nets . . . 240
16.1.3 New models . . . 241
16.2 Register new arguments . . . 242
16.3 Unit tests . . . 242

17 Atom Type Embedding 243

17.1 Overview . . . 243
17.2 Preliminary . . . 243
17.3 How to use . . . 244
17.4 Code Modification . . . 244
17.4.1 trainer (train/trainer.py) . . . 244
17.4.2 model (model/ener.py) . . . 244
17.4.3 embedding net (descriptor/se*.py) . . . 245
17.4.4 fitting net (fit/ener.py) . . . 245

18 Coding Conventions 247
18.1 Preface . . . 247
18.2 Python . . . 247
18.2.1 Rules . . . 247
18.2.2 Whitespace . . . 248
18.2.3 General advice . . . 248
18.2.4 Writing documentation in the code . . . 248
18.3 C++ . . . 249
18.4 Run scripts to check the code . . . 249

19 CI/CD 251
19.1 CI . . . 251
19.1.1 Test CUDA . . . 251
19.2 CD . . . 251

20 `deepmd` 253
20.1 Subpackages . . . 253
20.1.1 `deepmd.backend` . . . 253
20.1.2 `deepmd.dpmodel` . . . 260
20.1.3 `deepmd.entrypoints` . . . 382
20.1.4 `deepmd.infer` . . . 387
20.1.5 `deepmd.loggers` . . . 411
20.1.6 `deepmd.pt` . . . 414
20.1.7 `deepmd.tf` . . . 639
20.1.8 `deepmd.utils` . . . 1013
20.2 Submodules . . . 1058
20.2.1 `deepmd.__main__` . . . 1058
20.2.2 `deepmd.calculator` . . . 1058
20.2.3 `deepmd.common` . . . 1060
20.2.4 `deepmd.driver` . . . 1063
20.2.5 `deepmd.env` . . . 1063
20.2.6 `deepmd.main` . . . 1064
20.3 Package Contents . . . 1066
20.3.1 Functions . . . 1066

21 OP API 1069
21.1 op_module . . . 1069
21.2 op_grads_module . . . 1122

22 C++ API 1131
22.1 Class Hierarchy . . . 1131
22.2 File Hierarchy . . . 1131
22.3 Full API . . . 1132
22.3.1 Namespaces . . . 1132
22.3.2 Classes and Structs . . . 1134
22.3.3 Enums . . . 1175

22.3.4 Functions . . . 1175
22.3.5 Typedefs . . . 1184

23 C API 1185
23.1 Class Hierarchy . . . 1185
23.2 File Hierarchy . . . 1185
23.3 Full API . . . 1186
23.3.1 Namespaces . . . 1186
23.3.2 Classes and Structs . . . 1187
23.3.3 Functions . . . 1206
23.3.4 Defines . . . 1259
23.3.5 Typedefs . . . 1260

24 Core API 1263
24.1 Class Hierarchy . . . 1263
24.2 File Hierarchy . . . 1264
24.3 Full API . . . 1265
24.3.1 Namespaces . . . 1265
24.3.2 Classes and Structs . . . 1269
24.3.3 Unions . . . 1280
24.3.4 Functions . . . 1280
24.3.5 Variables . . . 1319
24.3.6 Defines . . . 1320
24.3.7 Typedefs . . . 1325

25 License 1327

26 Authors and Credits 1329
26.1 Cite DeePMD-kit and methods . . . 1329
26.2 Package Contributors . . . 1330
26.3 Other Credits . . . 1332

27 Logo 1333

Bibliography 1335

Python Module Index 1339

Index 1343

DeePMD-kit is a package written in Python/C++, designed to minimize the effort required to build deep learning-based models of interatomic potential energy and force field and to perform molecular dynamics (MD). This brings new hopes to addressing the accuracy-versus-efficiency dilemma in molecular simulations. Applications of DeePMD-kit span from finite molecules to extended systems and from metallic systems to chemically bonded systems.

Important: The project DeePMD-kit is licensed under GNU LGPLv3.0. If you use this code in any future publications, please cite the following publications for general purpose:

- Han Wang, Linfeng Zhang, Jiequn Han, and Weinan E. "DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics." Computer Physics Communications 228 (2018): 178-184.
- Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang. "DeePMD-kit v2: A software package for Deep Potential models." J. Chem. Phys., 159, 054801 (2023).

In addition, please follow this page to cite the methods you used.

CHAPTER

ONE

GETTING STARTED

In this text, we will call the deep neural network that is used to represent the interatomic interactions (Deep Potential) the model. The typical procedure of using DeePMD-kit is

1.1 Easy install

There are various easy methods to install DeePMD-kit. Choose one that you prefer. If you want to build by yourself, jump to the next two sections.

After your easy installation, DeePMD-kit (`dp`) and LAMMPS (`lmp`) will be available to execute. You can try `dp -h` and `lmp -h` to see the help. `mpirun` is also available considering you may want to train models or run LAMMPS in parallel.

Note: Note: The off-line packages and conda packages require the GNU C Library 2.17 or above. The GPU version requires compatible NVIDIA driver to be installed in advance. It is possible to force conda to override detection when installation, but these requirements are still necessary during runtime. You can refer to DeepModeling conda FAQ for more information.

Note: Python 3.8 or above is required for Python interface.

- Install off-line packages
- Install with conda
- Install with docker
- Install Python interface with pip

1.1.1 Install off-line packages

Both CPU and GPU version offline packages are available on the Releases page.

Some packages are split into two files due to the size limit of GitHub. One may merge them into one after downloading:

```
cat deepmd-kit-2.2.9-cuda118-Linux-x86_64.sh.0 deepmd-kit-2.2.9-cuda118-Linux-x86_64.sh.1 > deepmd-
↪kit-2.2.9-cuda118-Linux-x86_64.sh
```

One may enable the environment using

```
conda activate /path/to/deepmd-kit
```

1.1.2 Install with conda

DeePMD-kit is available with conda. Install Anaconda, Miniconda, or miniforge first. You can refer to DeepModeling conda FAQ for how to setup a conda environment.

conda-forge channel

DeePMD-kit is available on the conda-forge channel:

```
conda create -n deepmd deepmd-kit lammps horovod -c conda-forge
```

The supported platforms include Linux x86-64, macOS x86-64, and macOS arm64. Read conda-forge FAQ to learn how to install CUDA-enabled packages.

Official channel (deprecated)

> Danger: Deprecated since version 3.0.0: The official channel has been deprecated since 3.0.0, due to the challenging work of building dependencies for multiple backends. Old packages will still be available at https://conda.deepmodeling.com. Maintainers will build packages in the conda-forge organization together with other conda-forge members.

One may create an environment that contains the CPU version of DeePMD-kit and LAMMPS:

```
conda create -n deepmd deepmd-kit=*=*cpu libdeepmd=*=*cpu lammps -c https://conda.deepmodeling.com␣
↪-c defaults
```

Or one may want to create a GPU environment containing CUDA Toolkit:

```
conda create -n deepmd deepmd-kit=*=*gpu libdeepmd=*=*gpu lammps cudatoolkit=11.6 horovod -c␣
↪https://conda.deepmodeling.com -c defaults
```

One could change the CUDA Toolkit version from `10.2` or `11.6`.

One may specify the DeePMD-kit version such as `2.2.9` using

```
conda create -n deepmd deepmd-kit=2.2.9=*cpu libdeepmd=2.2.9=*cpu lammps horovod -c https://conda.
↪deepmodeling.com -c defaults
```

One may enable the environment using

```
conda activate deepmd
```

1.1.3 Install with docker

A docker for installing the DeePMD-kit is available here.

To pull the CPU version:

```
docker pull ghcr.io/deepmodeling/deepmd-kit:2.2.8_cpu
```

To pull the GPU version:

```
docker pull ghcr.io/deepmodeling/deepmd-kit:2.2.8_cuda12.0_gpu
```

1.1.4 Install Python interface with pip

If you have no existing TensorFlow installed, you can use `pip` to install the pre-built package of the Python interface with CUDA 12 supported:

```
pip install deepmd-kit[gpu,cu12,torch]
```

`cu12` is required only when CUDA Toolkit and cuDNN were not installed.

To install the package built against CUDA 11.8, use

```
pip install torch --index-url https://download.pytorch.org/whl/cu118
pip install deepmd-kit-cu11[gpu,cu11]
```

Or install the CPU version without CUDA supported:

```
pip install torch --index-url https://download.pytorch.org/whl/cpu
pip install deepmd-kit[cpu]
```

The LAMMPS module and the i-Pi driver are only provided on Linux and macOS for the TensorFlow backend. To install LAMMPS and/or i-Pi, add `lmp` and/or `ipi` to extras:

```
pip install deepmd-kit[gpu,cu12,torch,lmp,ipi]
```

MPICH is required for parallel running. (The macOS arm64 package doesn't support MPI yet.)

It is suggested to install the package into an isolated environment. The supported platform includes Linux x86-64 and aarch64 with GNU C Library 2.28 or above, macOS x86-64 and arm64, and Windows x86-64. A specific version of TensorFlow and PyTorch which is compatible with DeePMD-kit will be also installed.

> Warning: If your platform is not supported, or you want to build against the installed TensorFlow, or you want to enable ROCM support, please build from source.

1.2 DeePMD-kit Quick Start Tutorial

This is a quick start guide for "Deep Potential" molecular dynamics using DeePMD-kit, through which you can quickly understand the paradigm cycle that DeePMD-kit operates in and apply it to your projects.

1.2.1 Task

> Mastering the paradigm cycle of using DeePMD-kit to establish deep potential molecular dynamics models, and following a complete case to learn how to apply it to molecular dynamics tasks.

By the end of this tutorial, you will be able to:

- Prepare the formataive dataset and running scripts for training with DeePMD-kit;
- Train, freeze, and test DeePMD-kit models;
- Use DeePMD-kit in Lammps for calculations;

Work through this tutorial. It will take you 20 minutes, max!

Table of Contents

- DeePMD-kit Quick Start Tutorial
  - Task
  - Background
  - Practice
    - Data Preparation
    - Prepare input script
    - Train a model
    - Freeze a model
    - Compress a model
    - Test a model
    - Run MD with LAMMPS

1.2.2 Background

In this tutorial, we will take the gaseous methane molecule as an example to provide a detailed introduction to the training and application of the Deep Potential (DP) model.

DeePMD-kit is a software tool that employs neural networks to fit potential energy models based on first-principles data for molecular dynamics simulations. Without manual intervention, it can end-to-end transform the data provided by users into a deep potential model in a matter of hours. This model can seamlessly integrate with common molecular dynamics simulation software (like LAMMPS, OpenMM, and GROMACS).

DeePMD-kit significantly elevates the limits of molecular dynamics through high-performance computing and machine learning, achieving system scales of up to hundreds of millions of atoms while still maintaining the high accuracy of "ab initio" calculations. The simulation time scale is improved by at least 1000 times

compared to traditional methods. Its achievements earned the 2020 ACM Gordon Bell Prize, one of the highest honors in the field of high-performance computing, and it has been used by over a thousand research groups in physics, chemistry, materials science, biology, and other fields globally.

For more detailed usage, you can refer to the DeePMD-kit's documentation as a comprehensive reference.

In this case, the Deep Potential (DP) model was generated using the DeePMD-kit package.

1.2.3 Practice

Data Preparation

We have prepared the initial data for CH_4 required to run DeePMD-kit computations and placed it in the `DeePMD-kit_Tutorial` folder. You can view the corresponding files by clicking on the dataset on the left side:

```
import os

# Define the dataset URL and the paths
dataset_url = "https://bohrium-api.dp.tech/ds-dl/DeePMD-kit-Tutorial-a8z5-v1.zip"
zip_file_name = "DeePMD-kit-Tutorial-a8z5-v1.zip"
dataset_directory = "DeePMD-kit_Tutorial"
local_zip_path = f"/personal/{zip_file_name}"
extract_path = "/personal/"

# Check if the dataset directory exists to avoid re-downloading and re-extracting
if not os.path.isdir(f"{extract_path}{dataset_directory}"):
    # Download and extract if not exists
    if not os.path.isfile(local_zip_path):
        print("Downloading dataset...")
        !wget -q -O {local_zip_path} {dataset_url}

    print("Extracting dataset...")
    !unzip -q -n {local_zip_path} -d {extract_path}
else:
    print("Dataset is already downloaded and extracted.")

# Change the current working directory
os.chdir(f"{extract_path}")
print(f"Current path is: {os.getcwd()}")
```

```
Dataset is already downloaded and extracted.
Current path is: /personal
```

Let's take a look at the downloaded DeePMD-kit_Tutorial folder.

```
! tree DeePMD-kit_Tutorial -L 1
```

```
DeePMD-kit_Tutorial
   00.data
   01.train
   01.train.finished
   02.lmp
   02.lmp.finished

5 directories, 0 files
```

There are 3 subfolders under the DeePMD-kit_Tutorial folder: 00.data, 01.train, and 02.lmp.

- The 00.data folder is used to store training and testing data.
- The 01.train folder contains example scripts for training models using DeePMD-kit.
- The 01.train.finished folder includes the complete results of the training process.
- The 02.lmp folder contains example scripts for molecular dynamics simulations using LAMMPS.

Let's first take a look at the DeePMD-kit_Tutorial/00.data folder.

```
! tree DeePMD-kit_Tutorial/00.data -L 1
```

```
DeePMD-kit_Tutorial/00.data
   abacus_md
   training_data
   validation_data

3 directories, 0 files
```

DeePMD-kit's training data originates from first-principles calculation data, including atomic types, simulation cells, atomic coordinates, atomic forces, system energies, and virials.

In the 00.data folder, there is only the abacus_md folder, which contains data obtained through ab initio Molecular Dynamics (AIMD) simulations using ABACUS. In this tutorial, we have already completed the ab initio molecular dynamics calculations for the methane molecule for you.

Detailed information about ABACUS can be found in its documentation.

DeePMD-kit uses a compressed data format. All training data should first be converted into this format before they can be used in DeePMD-kit. This data format is explained in detail in the DeePMD-kit manual, which can be found on DeePMD-kit's Github.

We provide a convenient tool dpdata, which can convert data generated by VASP, CP2K, Gaussian, Quantum Espresso, ABACUS, and LAMMPS into DeePMD-kit's compressed format.

A snapshot of a molecular system that contains computational data information is called a frame. A data system comprises many frames sharing the same number of atoms and atom types.

For example, a molecular dynamics trajectory can be converted into a data system, where each timestep corresponds to one frame in the system.

Next, we use the dpdata tool to randomly split the data in abacus_md into training and validation data.

```
import dpdata
import numpy as np

# load data of abacus/md format
data = dpdata.LabeledSystem("DeePMD-kit_Tutorial/00.data/abacus_md", fmt="abacus/md")
print("# the data contains %d frames" % len(data))

# random choose 40 index for validation_data
rng = np.random.default_rng()
index_validation = rng.choice(201, size=40, replace=False)

# other indexes are training_data
index_training = list(set(range(201)) - set(index_validation))
data_training = data.sub_system(index_training)
data_validation = data.sub_system(index_validation)
```

(continues on next page)

(continued from previous page)

```
# all training data put into directory:"training_data"
data_training.to_deepmd_npy("DeePMD-kit_Tutorial/00.data/training_data")

# all validation data put into directory:"validation_data"
data_validation.to_deepmd_npy("DeePMD-kit_Tutorial/00.data/validation_data")

print("# the training data contains %d frames" % len(data_training))
print("# the validation data contains %d frames" % len(data_validation))
```

```
# the data contains 201 frames
# the training data contains 161 frames
# the validation data contains 40 frames
```

As you can see, 161 frames are picked as training data, and the other 40 frames are validation dat.

Let's take another look at the 00.data folder, where new files have been generated, which are the training and validation sets required for Deep Potential training with DeePMD-kit.

```
! tree DeePMD-kit_Tutorial/00.data/ -L 1
```

```
DeePMD-kit_Tutorial/00.data/
   abacus_md
   training_data
   validation_data

3 directories, 0 files
```

```
! tree DeePMD-kit_Tutorial/00.data/training_data -L 1
```

```
DeePMD-kit_Tutorial/00.data/training_data
   set.000
   type.raw
   type_map.raw

1 directory, 2 files
```

The functions of these files are as follows:

- set.000: It is a directory that contains compressed format data (NumPy compressed arrays).
- type.raw: It is a file that contains the types of atoms (represented as integers).
- type_map.raw: It is a file that contains the names of the types of atoms.

Let's take a look at these files.

Let's have a look at `type.raw`:

```
! cat DeePMD-kit_Tutorial/00.data/training_data/type.raw
```

```
0
0
0
0
1
```

This tells us there are 5 atoms in this example, 4 atoms represented by type "0", and 1 atom represented by type "1". Sometimes one needs to map the integer types to atom name. The mapping can be given by the file `type_map.raw`

```
! cat DeePMD-kit_Tutorial/00.data/training_data/type_map.raw
```

```
H
C
```

This tells us the type "0" is named by "H", and the type "1" is named by "C".

More detailed documentation on using dpdata for data conversion can be found here

Prepare input script

Once the data preparation is done, we can go on with training. Now go to the training directory. DeePMD-kit requires a `json` format file to specify parameters for training.

```
# Checke dargs version and Install
!pip show dargs || pip install --upgrade dargs
```

```
# Show input.json
from deepmd.utils.argcheck import gen_args
from dargs.notebook import JSON

with open("./DeePMD-kit_Tutorial/01.train/input.json") as f:
    JSON(f.read(), gen_args())
```

```
<IPython.core.display.HTML object>
```

DeePMD-kit requires a `json` format file to specify parameters for training.

In the model section, the parameters of embedding and fitting networks are specified.

```
"model":{
    "type_map":     ["H", "C"],
    "descriptor":{
        "type":             "se_e2_a",
        "rcut":             6.00,
        "rcut_smth":        0.50,
        "sel":              "auto",
        "neuron":           [25, 50, 100],
        "resnet_dt":        false,
        "axis_neuron":      16,
        "seed":             1,
        "_comment":         "that's all"
        },
    "fitting_net":{
        "neuron":           [240, 240, 240],
        "resnet_dt":        true,
        "seed":             1,
        "_comment":         "that's all"
    },
    "_comment":     "that's all"
},
```

The explanation for some of the parameters is as follows:

Parameter	Expiation
type_map	the name of each type of atom
descriptor > type	the type of descriptor
descriptor > rcut	cut-off radius
descriptor > rcut_smth	where the smoothing starts
descriptor > sel	the maximum number of type i atoms in the cut-off radius
descriptor > neuron	size of the embedding neural network
descriptor > axis_neuron	the size of the submatrix of G (embedding matrix)
fitting_net > neuron	size of the fitting neural network

The `se_e2_a` descriptor is used to train the DP model. The item neurons set the size of the descriptors and fitting network to [25, 50, 100] and [240, 240, 240], respectively. The components in local environment to smoothly go to zero from 0.5 to 6 Å.

The following are the parameters that specify the learning rate and loss function.

```
    "learning_rate" :{
        "type":                 "exp",
        "decay_steps":          50,
        "start_lr":             0.001,
        "stop_lr":              3.51e-8,
        "_comment":             "that's all"
    },
    "loss" :{
        "type":                 "ener",
        "start_pref_e":         0.02,
        "limit_pref_e":         1,
        "start_pref_f":         1000,
        "limit_pref_f":         1,
        "start_pref_v":         0,
        "limit_pref_v":         0,
        "_comment":             "that's all"
    },
```

In the loss function, `pref_e` increases from 0.02 to 1, and `pref_f` decreases from 1000 to 1 progressively, which means that the force term dominates at the beginning, while energy and virial terms become important at the end. This strategy is very effective and reduces the total training time. `pref_v` is set to 0 , indicating that no virial data are included in the training process. The starting learning rate, stop learning rate, and decay steps are set to 0.001, 3.51e-8, and 50, respectively. The model is trained for 10000 steps.

The training parameters are given in the following

```
    "training" : {
        "training_data": {
            "systems":              ["../00.data/training_data"],
            "batch_size":           "auto",
            "_comment":             "that's all"
        },
        "validation_data":{
            "systems":              ["../00.data/validation_data/"],
            "batch_size":           "auto",
            "numb_btch":            1,
            "_comment":             "that's all"
        },
```

(continues on next page)

(continued from previous page)

```
        "numb_steps":            10000,
        "seed":                  10,
        "disp_file":             "lcurve.out",
        "disp_freq":             200,
        "save_freq":             10000,
        },
```

More detailed docs about Data conversion can be found here

Train a model

After the training script is prepared, we can start the training with DeePMD-kit by simply running

```
# ########## Time Warning: 120 secs,C32_CPU ; 13 mins ,C2_CPU ##########
! cd DeePMD-kit_Tutorial/01.train/ && dp train input.json
```

```
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/tensorflow/
↪python/compat/v2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_
↪scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
WARNING:root:To get the best performance, it is recommended to adjust the number of threads by␣
↪setting the environment variables OMP_NUM_THREADS, TF_INTRA_OP_PARALLELISM_THREADS, and TF_INTER_
↪OP_PARALLELISM_THREADS. See https://deepmd.rtfd.io/parallelism/ for more information.
DEEPMD INFO    Calculate neighbor statistics... (add --skip-neighbor-stat to skip this step)
DEEPMD INFO    training data with min nbor dist: 1.0460506586976848
DEEPMD INFO    training data with max nbor size: [4 1]
DEEPMD INFO     _____               _____   __  __  _____           _     _  _
DEEPMD INFO    |  __ \             |  __ \ |  \/  ||  __ \         | |   (_)| |
DEEPMD INFO    | |  | |  ___   ___ | |__) || \  / || |  | | ______ | | __ _ | |_
DEEPMD INFO    | |  | | / _ \ / _ \|  ___/ | |\/| || |  | ||______|| |/ /| || __|
DEEPMD INFO    | |__| ||  __/|  __/| |     | |  | || |__| |        |   < | || |_
DEEPMD INFO    |_____/  \___| \___||_|     |_|  |_||_____/         |_|\_\|_| \__|
DEEPMD INFO    Please read and cite:
DEEPMD INFO    Wang, Zhang, Han and E, Comput.Phys.Comm. 228, 178-184 (2018)
DEEPMD INFO    Zeng et al, J. Chem. Phys., 159, 054801 (2023)
DEEPMD INFO    See https://deepmd.rtfd.io/credits/ for details.
DEEPMD INFO    installed to:         /root/miniconda3/envs/deepmd
DEEPMD INFO    source :              v2.2.7
DEEPMD INFO    source brach:         HEAD
DEEPMD INFO    source commit:        839f4fe7
DEEPMD INFO    source commit at:     2023-10-27 21:10:24 +0800
DEEPMD INFO    build float prec:     double
DEEPMD INFO    build variant:        cpu
DEEPMD INFO    build with tf inc:    /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/
↪tensorflow/include;/root/miniconda3/envs/deepmd/lib/python3.10/site-packages/tensorflow/../../../
↪../include
DEEPMD INFO    build with tf lib:
DEEPMD INFO    ---Summary of the training---------------------------------------
DEEPMD INFO    running on:           bohrium-21213-1088639
DEEPMD INFO    computing device:     cpu:0
DEEPMD INFO    Count of visible GPU: 0
DEEPMD INFO    num_intra_threads:    0
DEEPMD INFO    num_inter_threads:    0
DEEPMD INFO    ------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
DEEPMD INFO    ---Summary of DataSystem: training     -----------------------------------------------
↪--
DEEPMD INFO    found 1 system(s):
DEEPMD INFO                                    system  natoms  bch_sz   n_bch   prob  pbc
DEEPMD INFO                      ../00.data/training_data       5       7      23  1.000    T
DEEPMD INFO    ------------------------------------------------------------------------------------
↪--
DEEPMD INFO    ---Summary of DataSystem: validation   -----------------------------------------------
↪--
DEEPMD INFO    found 1 system(s):
DEEPMD INFO                                    system  natoms  bch_sz   n_bch   prob  pbc
DEEPMD INFO                    ../00.data/validation_data       5       7       5  1.000    T
DEEPMD INFO    ------------------------------------------------------------------------------------
↪--
DEEPMD INFO    training without frame parameter
DEEPMD INFO    data stating... (this step may take long time)
DEEPMD INFO    built lr
DEEPMD INFO    built network
DEEPMD INFO    built training
WARNING:root:To get the best performance, it is recommended to adjust the number of threads by␣
↪setting the environment variables OMP_NUM_THREADS, TF_INTRA_OP_PARALLELISM_THREADS, and TF_INTER_
↪OP_PARALLELISM_THREADS. See https://deepmd.rtfd.io/parallelism/ for more information.
DEEPMD INFO    initialize model from scratch
DEEPMD INFO    start training at lr 1.00e-03 (== 1.00e-03), decay_step 50, decay_rate 0.950006,␣
↪final lr will be 3.51e-08
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/deepmd/train/
↪trainer.py:1197: py_func (from tensorflow.python.ops.script_ops) is deprecated and will be␣
↪removed in a future version.
Instructions for updating:
tf.py_func is deprecated in TF V2. Instead, there are two
    options available in V2.
    - tf.py_function takes a python function which manipulates tf eager
    tensors instead of numpy arrays. It's easy to convert a tf eager tensor to
    an ndarray (just call tensor.numpy()) but having access to eager tensors
    means `tf.py_function`s can use accelerators such as GPUs as well as
    being differentiable using a gradient tape.
    - tf.numpy_function maintains the semantics of the deprecated tf.py_func
    (it is not differentiable, and manipulates numpy arrays). It drops the
    stateful argument making all functions stateful.

WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/deepmd/train/
↪trainer.py:1197: py_func (from tensorflow.python.ops.script_ops) is deprecated and will be␣
↪removed in a future version.
Instructions for updating:
tf.py_func is deprecated in TF V2. Instead, there are two
    options available in V2.
    - tf.py_function takes a python function which manipulates tf eager
    tensors instead of numpy arrays. It's easy to convert a tf eager tensor to
    an ndarray (just call tensor.numpy()) but having access to eager tensors
    means `tf.py_function`s can use accelerators such as GPUs as well as
    being differentiable using a gradient tape.
    - tf.numpy_function maintains the semantics of the deprecated tf.py_func
    (it is not differentiable, and manipulates numpy arrays). It drops the
    stateful argument making all functions stateful.

DEEPMD INFO    batch     200 training time 17.53 s, testing time 0.05 s, total wall time 18.41 s
DEEPMD INFO    batch     400 training time 14.96 s, testing time 0.05 s, total wall time 15.11 s
```

(continues on next page)

(continued from previous page)

```
DEEPMD INFO    batch     600 training time 15.47 s, testing time 0.05 s, total wall time 15.65 s
DEEPMD INFO    batch     800 training time 14.25 s, testing time 0.04 s, total wall time 14.41 s
DEEPMD INFO    batch    1000 training time 15.49 s, testing time 0.05 s, total wall time 15.65 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    1200 training time 16.33 s, testing time 0.08 s, total wall time 17.33 s
DEEPMD INFO    batch    1400 training time 14.31 s, testing time 0.05 s, total wall time 14.47 s
DEEPMD INFO    batch    1600 training time 16.54 s, testing time 0.05 s, total wall time 16.72 s
DEEPMD INFO    batch    1800 training time 16.90 s, testing time 0.09 s, total wall time 17.09 s
DEEPMD INFO    batch    2000 training time 17.20 s, testing time 0.06 s, total wall time 17.37 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    2200 training time 14.29 s, testing time 0.04 s, total wall time 14.83 s
DEEPMD INFO    batch    2400 training time 13.11 s, testing time 0.04 s, total wall time 13.29 s
DEEPMD INFO    batch    2600 training time 12.93 s, testing time 0.04 s, total wall time 13.08 s
DEEPMD INFO    batch    2800 training time 14.58 s, testing time 0.04 s, total wall time 14.74 s
DEEPMD INFO    batch    3000 training time 13.21 s, testing time 0.04 s, total wall time 13.35 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    3200 training time 14.40 s, testing time 0.07 s, total wall time 15.14 s
DEEPMD INFO    batch    3400 training time 13.08 s, testing time 0.04 s, total wall time 13.23 s
DEEPMD INFO    batch    3600 training time 12.93 s, testing time 0.06 s, total wall time 13.13 s
DEEPMD INFO    batch    3800 training time 15.23 s, testing time 0.05 s, total wall time 15.43 s
DEEPMD INFO    batch    4000 training time 13.20 s, testing time 0.04 s, total wall time 13.35 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    4200 training time 14.82 s, testing time 0.05 s, total wall time 16.06 s
DEEPMD INFO    batch    4400 training time 14.26 s, testing time 0.05 s, total wall time 14.42 s
DEEPMD INFO    batch    4600 training time 15.50 s, testing time 0.05 s, total wall time 15.66 s
DEEPMD INFO    batch    4800 training time 14.12 s, testing time 0.05 s, total wall time 14.29 s
DEEPMD INFO    batch    5000 training time 15.71 s, testing time 0.05 s, total wall time 15.88 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    5200 training time 14.36 s, testing time 0.07 s, total wall time 15.40 s
DEEPMD INFO    batch    5400 training time 15.77 s, testing time 0.05 s, total wall time 15.93 s
DEEPMD INFO    batch    5600 training time 14.12 s, testing time 0.05 s, total wall time 14.29 s
DEEPMD INFO    batch    5800 training time 15.53 s, testing time 0.04 s, total wall time 15.70 s
DEEPMD INFO    batch    6000 training time 15.39 s, testing time 0.09 s, total wall time 15.58 s
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/tensorflow/
↪python/training/saver.py:1066: remove_checkpoint (from tensorflow.python.training.checkpoint_
↪management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to delete files with this prefix.
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/tensorflow/
↪python/training/saver.py:1066: remove_checkpoint (from tensorflow.python.training.checkpoint_
↪management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to delete files with this prefix.
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    6200 training time 14.74 s, testing time 0.05 s, total wall time 15.64 s
DEEPMD INFO    batch    6400 training time 15.24 s, testing time 0.09 s, total wall time 15.44 s
DEEPMD INFO    batch    6600 training time 14.29 s, testing time 0.05 s, total wall time 14.48 s
DEEPMD INFO    batch    6800 training time 15.46 s, testing time 0.09 s, total wall time 15.66 s
DEEPMD INFO    batch    7000 training time 15.34 s, testing time 0.05 s, total wall time 15.54 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    7200 training time 15.63 s, testing time 0.05 s, total wall time 16.19 s
DEEPMD INFO    batch    7400 training time 14.71 s, testing time 0.06 s, total wall time 14.90 s
DEEPMD INFO    batch    7600 training time 15.96 s, testing time 0.05 s, total wall time 16.12 s
DEEPMD INFO    batch    7800 training time 19.68 s, testing time 0.06 s, total wall time 19.92 s
DEEPMD INFO    batch    8000 training time 15.81 s, testing time 0.07 s, total wall time 16.00 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    8200 training time 13.62 s, testing time 0.04 s, total wall time 14.54 s
```

(continues on next page)

(continued from previous page)

```
DEEPMD INFO    batch    8400 training time 13.23 s, testing time 0.04 s, total wall time 13.38 s
DEEPMD INFO    batch    8600 training time 14.90 s, testing time 0.04 s, total wall time 15.08 s
DEEPMD INFO    batch    8800 training time 13.19 s, testing time 0.04 s, total wall time 13.34 s
DEEPMD INFO    batch    9000 training time 13.78 s, testing time 0.09 s, total wall time 14.00 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    9200 training time 13.76 s, testing time 0.04 s, total wall time 14.41 s
DEEPMD INFO    batch    9400 training time 13.06 s, testing time 0.04 s, total wall time 13.20 s
DEEPMD INFO    batch    9600 training time 14.23 s, testing time 0.04 s, total wall time 14.42 s
DEEPMD INFO    batch    9800 training time 13.72 s, testing time 0.05 s, total wall time 13.88 s
DEEPMD INFO    batch   10000 training time 13.92 s, testing time 0.09 s, total wall time 14.12 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    average training time: 0.0737 s/batch (exclude first 200 batches)
DEEPMD INFO    finished training
DEEPMD INFO    wall time: 756.650 s
```

On the screen, you will see the information of the data system(s)

```
DEEPMD INFO    ---------------------------------------------------------------------
DEEPMD INFO    ---Summary of DataSystem: training     -----------------------------------
DEEPMD INFO    found 1 system(s):
DEEPMD INFO                                system  natoms  bch_sz   n_bch   prob  pbc
DEEPMD INFO             ../00.data/training_data       5       7      23  1.000    T
DEEPMD INFO    ---------------------------------------------------------------------------
DEEPMD INFO    ---Summary of DataSystem: validation   -----------------------------------
DEEPMD INFO    found 1 system(s):
DEEPMD INFO                                system  natoms  bch_sz   n_bch   prob  pbc
DEEPMD INFO           ../00.data/validation_data       5       7       5  1.000    T
DEEPMD INFO    ---------------------------------------------------------------------------
```

and the starting and final learning rate of this training

```
DEEPMD INFO    start training at lr 1.00e-03 (== 1.00e-03), decay_step 50, decay_rate 0.950006,␣
↪final lr will be 3.51e-08
```

If everything works fine, you will see, on the screen, information printed every 1000 steps, like

```
DEEPMD INFO    batch     200 training time 6.04 s, testing time 0.02 s
DEEPMD INFO    batch     400 training time 4.80 s, testing time 0.02 s
DEEPMD INFO    batch     600 training time 4.80 s, testing time 0.02 s
DEEPMD INFO    batch     800 training time 4.78 s, testing time 0.02 s
DEEPMD INFO    batch    1000 training time 4.77 s, testing time 0.02 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    1200 training time 4.47 s, testing time 0.02 s
DEEPMD INFO    batch    1400 training time 4.49 s, testing time 0.02 s
DEEPMD INFO    batch    1600 training time 4.45 s, testing time 0.02 s
DEEPMD INFO    batch    1800 training time 4.44 s, testing time 0.02 s
DEEPMD INFO    batch    2000 training time 4.46 s, testing time 0.02 s
DEEPMD INFO    saved checkpoint model.ckpt
```

They present the training and testing time counts. At the end of the 1000th batch, the model is saved in Tensorflow's checkpoint file `model.ckpt`. At the same time, the training and testing errors are presented in file `lcurve.out`.

The file contains 8 columns, form left to right, are the training step, the validation loss, training loss, root mean square (RMS) validation error of energy, RMS training error of energy, RMS validation error of force, RMS training error of force and the learning rate. The RMS error (RMSE) of the energy is normalized by number of atoms in the system.

```
head -n 2 lcurve.out
#  step      rmse_val    rmse_trn    rmse_e_val  rmse_e_trn    rmse_f_val  rmse_f_trn         lr
      0      2.02e+01    1.51e+01      1.37e-01    1.41e-01      6.40e-01    4.79e-01    1.0e-03
```

and

```
$ tail -n 2 lcurve.out
   9800      2.45e-02    4.02e-02      3.20e-04    3.88e-04      2.40e-02    3.94e-02    4.3e-08
  10000      4.60e-02    3.76e-02      8.65e-04    5.35e-04      4.52e-02    3.69e-02    3.5e-08
```

Volumes 4, 5 and 6, 7 present energy and force training and testing errors, respectively.

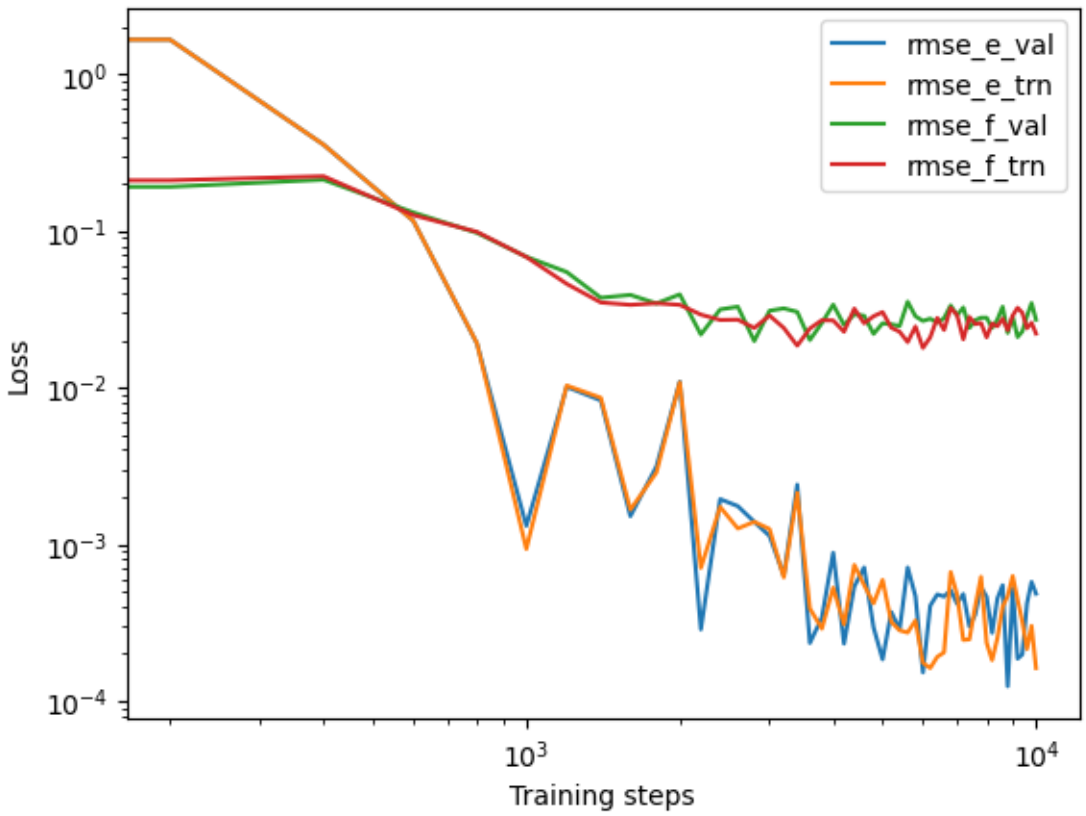
```
! cd DeePMD-kit_Tutorial/01.train.finished/ && head -n 2 lcurve.out && tail -n 2 lcurve.out
```

```
#  step      rmse_val    rmse_trn    rmse_e_val  rmse_e_trn    rmse_f_val  rmse_f_trn         lr
      0      1.79e+01    2.26e+01      1.35e-01    1.33e-01      5.67e-01    7.15e-01    1.0e-03
   9800      3.53e-02    2.64e-02      5.75e-04    3.01e-04      3.46e-02    2.59e-02    4.3e-08
  10000      2.76e-02    2.25e-02      4.83e-04    1.62e-04      2.71e-02    2.21e-02    3.5e-08
```

The loss function can be visualized to monitor the training process.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

with open("./DeePMD-kit_Tutorial/01.train.finished/lcurve.out") as f:
    headers = f.readline().split()[1:]
lcurve = pd.DataFrame(
    np.loadtxt("./DeePMD-kit_Tutorial/01.train.finished/lcurve.out"), columns=headers
)
legends = ["rmse_e_val", "rmse_e_trn", "rmse_f_val", "rmse_f_trn"]
for legend in legends:
    plt.loglog(lcurve["step"], lcurve[legend], label=legend)
plt.legend()
plt.xlabel("Training steps")
plt.ylabel("Loss")
plt.show()
```



Freeze a model

At the end of the training, the model parameters saved in TensorFlow's checkpoint file should be frozen as a model file that is usually ended with extension .pb. Simply execute

```
## Navigate to the DeePMD-kit_Tutorial/01.train/ Directory to Freeze the Model
! cd DeePMD-kit_Tutorial/01.train.finished/ && dp freeze -o graph.pb
```

```
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/tensorflow/
↪python/compat/v2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_
↪scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
WARNING:root:To get the best performance, it is recommended to adjust the number of threads by␣
↪setting the environment variables OMP_NUM_THREADS, TF_INTRA_OP_PARALLELISM_THREADS, and TF_INTER_
↪OP_PARALLELISM_THREADS. See https://deepmd.rtfd.io/parallelism/ for more information.
DEEPMD WARNING The following nodes are not in the graph: {'fitting_attr/aparam_nall', 'spin_attr/
↪ntypes_spin'}. Skip freezeing these nodes. You may be freezing a checkpoint generated by an old␣
↪version.
DEEPMD INFO    The following nodes will be frozen: ['descrpt_attr/rcut', 'model_attr/model_version
↪', 'o_atom_virial', 'model_attr/tmap', 'model_attr/model_type', 'o_force', 'o_energy', 'train_
↪attr/min_nbor_dist', 'model_type', 't_mesh', 'fitting_attr/daparam', 'train_attr/training_script
↪', 'fitting_attr/dfparam', 'o_atom_energy', 'descrpt_attr/ntypes', 'o_virial']
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/deepmd/
↪entrypoints/freeze.py:370: convert_variables_to_constants (from tensorflow.python.framework.
↪graph_util_impl) is deprecated and will be removed in a future version.
Instructions for updating:
```

(continues on next page)

(continued from previous page)

```
Use `tf.compat.v1.graph_util.convert_variables_to_constants`
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/deepmd/
↪entrypoints/freeze.py:370: convert_variables_to_constants (from tensorflow.python.framework.
↪graph_util_impl) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.compat.v1.graph_util.convert_variables_to_constants`
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/tensorflow/
↪python/framework/convert_to_constants.py:925: extract_sub_graph (from tensorflow.python.
↪framework.graph_util_impl) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.compat.v1.graph_util.extract_sub_graph`
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/tensorflow/
↪python/framework/convert_to_constants.py:925: extract_sub_graph (from tensorflow.python.
↪framework.graph_util_impl) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.compat.v1.graph_util.extract_sub_graph`
DEEPMD INFO    1222 ops in the final graph.
```

and it will output a model file named `graph.pb` in the current directory.

Compress a model

To enhance computational efficiency with DP models, compression significantly accelerates DP-based calculations and reduces memory usage. We can compress the model by running:

```
## Navigate to the DeePMD-kit_Tutorial/01.train/ Directory to Compress the Model
! cd DeePMD-kit_Tutorial/01.train.finished/ && dp compress -i graph.pb -o compress.pb
```

```
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/tensorflow/
↪python/compat/v2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_
↪scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
WARNING:root:To get the best performance, it is recommended to adjust the number of threads by␣
↪setting the environment variables OMP_NUM_THREADS, TF_INTRA_OP_PARALLELISM_THREADS, and TF_INTER_
↪OP_PARALLELISM_THREADS. See https://deepmd.rtfd.io/parallelism/ for more information.
DEEPMD INFO


DEEPMD INFO    stage 1: compress the model
DEEPMD INFO     _____               _____   __  __  _____           _     _  _
DEEPMD INFO    |  __ \             |  __ \ |  \/  ||  __ \         | |   (_)| |
DEEPMD INFO    | |  | |  ___   ___ | |__) || \  / || |  | | ______ | | __ _ | |_
DEEPMD INFO    | |  | | / _ \ / _ \|  ___/ | |\/| || |  | ||______|| |/ /| || __|
DEEPMD INFO    | |__| ||  __/|  __/| |     | |  | || |__| |        |   < | || |_
DEEPMD INFO    |_____/  \___| \___||_|     |_|  |_||_____/         |_|\_\|_| \__|
DEEPMD INFO    Please read and cite:
DEEPMD INFO    Wang, Zhang, Han and E, Comput.Phys.Comm. 228, 178-184 (2018)
DEEPMD INFO    Zeng et al, J. Chem. Phys., 159, 054801 (2023)
DEEPMD INFO    See https://deepmd.rtfd.io/credits/ for details.
DEEPMD INFO    installed to:         /root/miniconda3/envs/deepmd
DEEPMD INFO    source :              v2.2.7
DEEPMD INFO    source brach:         HEAD
DEEPMD INFO    source commit:        839f4fe7
DEEPMD INFO    source commit at:     2023-10-27 21:10:24 +0800
```

(continues on next page)

(continued from previous page)

```
DEEPMD INFO    build float prec:     double
DEEPMD INFO    build variant:        cpu
DEEPMD INFO    build with tf inc:    /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/
↪tensorflow/include;/root/miniconda3/envs/deepmd/lib/python3.10/site-packages/tensorflow/../../../
↪../include
DEEPMD INFO    build with tf lib:
DEEPMD INFO    ---Summary of the training---------------------------------------
DEEPMD INFO    running on:           bohrium-21213-1088639
DEEPMD INFO    computing device:     cpu:0
DEEPMD INFO    Count of visible GPU: 0
DEEPMD INFO    num_intra_threads:    0
DEEPMD INFO    num_inter_threads:    0
DEEPMD INFO    -----------------------------------------------------------------
DEEPMD INFO    training without frame parameter
DEEPMD INFO    training data with lower boundary: [-0.92929175 -0.99957951]
DEEPMD INFO    training data with upper boundary: [1.97058099 1.10195361]
DEEPMD INFO    built lr
DEEPMD INFO    built network
DEEPMD INFO    built training
WARNING:root:To get the best performance, it is recommended to adjust the number of threads by␣
↪setting the environment variables OMP_NUM_THREADS, TF_INTRA_OP_PARALLELISM_THREADS, and TF_INTER_
↪OP_PARALLELISM_THREADS. See https://deepmd.rtfd.io/parallelism/ for more information.
DEEPMD INFO    initialize model from scratch
DEEPMD INFO    finished compressing
DEEPMD INFO


DEEPMD INFO    stage 2: freeze the model
DEEPMD WARNING The following nodes are not in the graph: {'spin_attr/ntypes_spin', 'fitting_attr/
↪aparam_nall'}. Skip freezeing these nodes. You may be freezing a checkpoint generated by an old␣
↪version.
DEEPMD INFO    The following nodes will be frozen: ['train_attr/min_nbor_dist', 'o_energy',
↪'descrpt_attr/rcut', 'o_force', 'model_type', 'fitting_attr/daparam', 'model_attr/tmap', 'o_atom_
↪energy', 'descrpt_attr/ntypes', 'o_virial', 't_mesh', 'model_attr/model_type', 'fitting_attr/
↪dfparam', 'o_atom_virial', 'train_attr/training_script', 'model_attr/model_version']
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/deepmd/
↪entrypoints/freeze.py:370: convert_variables_to_constants (from tensorflow.python.framework.
↪graph_util_impl) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.compat.v1.graph_util.convert_variables_to_constants`
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/deepmd/
↪entrypoints/freeze.py:370: convert_variables_to_constants (from tensorflow.python.framework.
↪graph_util_impl) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.compat.v1.graph_util.convert_variables_to_constants`
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/tensorflow/
↪python/framework/convert_to_constants.py:925: extract_sub_graph (from tensorflow.python.
↪framework.graph_util_impl) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.compat.v1.graph_util.extract_sub_graph`
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/tensorflow/
↪python/framework/convert_to_constants.py:925: extract_sub_graph (from tensorflow.python.
↪framework.graph_util_impl) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.compat.v1.graph_util.extract_sub_graph`
DEEPMD INFO    858 ops in the final graph.
```

Test a model

We can check the quality of the trained model by running

```
! cd DeePMD-kit_Tutorial/01.train.finished/ && dp test -m graph.pb -s ../00.data/validation_data
```

```
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/tensorflow/
↪python/compat/v2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_
↪scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
WARNING:root:To get the best performance, it is recommended to adjust the number of threads by␣
↪setting the environment variables OMP_NUM_THREADS, TF_INTRA_OP_PARALLELISM_THREADS, and TF_INTER_
↪OP_PARALLELISM_THREADS. See https://deepmd.rtfd.io/parallelism/ for more information.
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/deepmd/utils/
↪batch_size.py:62: is_gpu_available (from tensorflow.python.framework.test_util) is deprecated␣
↪and will be removed in a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/deepmd/utils/
↪batch_size.py:62: is_gpu_available (from tensorflow.python.framework.test_util) is deprecated␣
↪and will be removed in a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
DEEPMD WARNING You can use the environment variable DP_INFER_BATCH_SIZE tocontrol the inference␣
↪batch size (nframes * natoms). The default value is 1024.
DEEPMD INFO    # ---------------output of dp test---------------
DEEPMD INFO    # testing system : ../00.data/validation_data
DEEPMD INFO    # number of test data : 40
DEEPMD INFO    Energy MAE         : 1.473845e-03 eV
DEEPMD INFO    Energy RMSE        : 2.007936e-03 eV
DEEPMD INFO    Energy MAE/Natoms  : 2.947689e-04 eV
DEEPMD INFO    Energy RMSE/Natoms : 4.015871e-04 eV
DEEPMD INFO    Force  MAE         : 2.146239e-02 eV/A
DEEPMD INFO    Force  RMSE        : 2.748797e-02 eV/A
DEEPMD INFO    Virial MAE         : 2.879183e-02 eV
DEEPMD INFO    Virial RMSE        : 3.817983e-02 eV
DEEPMD INFO    Virial MAE/Natoms  : 5.758366e-03 eV
DEEPMD INFO    Virial RMSE/Natoms : 7.635965e-03 eV
DEEPMD INFO    # ----------------------------------------------
```

The correlation between predicted data and original data can also be calculated.

```
import dpdata

training_systems = dpdata.LabeledSystem(
    "./DeePMD-kit_Tutorial/00.data/training_data", fmt="deepmd/npy"
)
predict = training_systems.predict("./DeePMD-kit_Tutorial/01.train.finished/graph.pb")
```

```
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/tensorflow/
↪python/compat/v2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_
↪scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

```
WARNING:root:To get the best performance, it is recommended to adjust the number of threads by␣
↪setting the environment variables OMP_NUM_THREADS, TF_INTRA_OP_PARALLELISM_THREADS, and TF_INTER_
↪OP_PARALLELISM_THREADS. See https://deepmd.rtfd.io/parallelism/ for more information.
```

```
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/deepmd/utils/
↪batch_size.py:62: is_gpu_available (from tensorflow.python.framework.test_util) is deprecated␣
↪and will be removed in a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
```

```
2024-03-24 23:05:17.177887: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow␣
↪binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU␣
↪instructions in performance-critical operations:  SSE4.1 SSE4.2 AVX AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-03-24 23:05:17.179243: I tensorflow/core/common_runtime/process_util.cc:146] Creating new␣
↪thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads for best␣
↪performance.
2024-03-24 23:05:17.197330: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:354] MLIR␣
↪V1 optimization pass is not enabled
WARNING:tensorflow:From /root/miniconda3/envs/deepmd/lib/python3.10/site-packages/deepmd/utils/
↪batch_size.py:62: is_gpu_available (from tensorflow.python.framework.test_util) is deprecated␣
↪and will be removed in a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
WARNING:deepmd.utils.batch_size:You can use the environment variable DP_INFER_BATCH_SIZE tocontrol␣
↪the inference batch size (nframes * natoms). The default value is 1024.
```

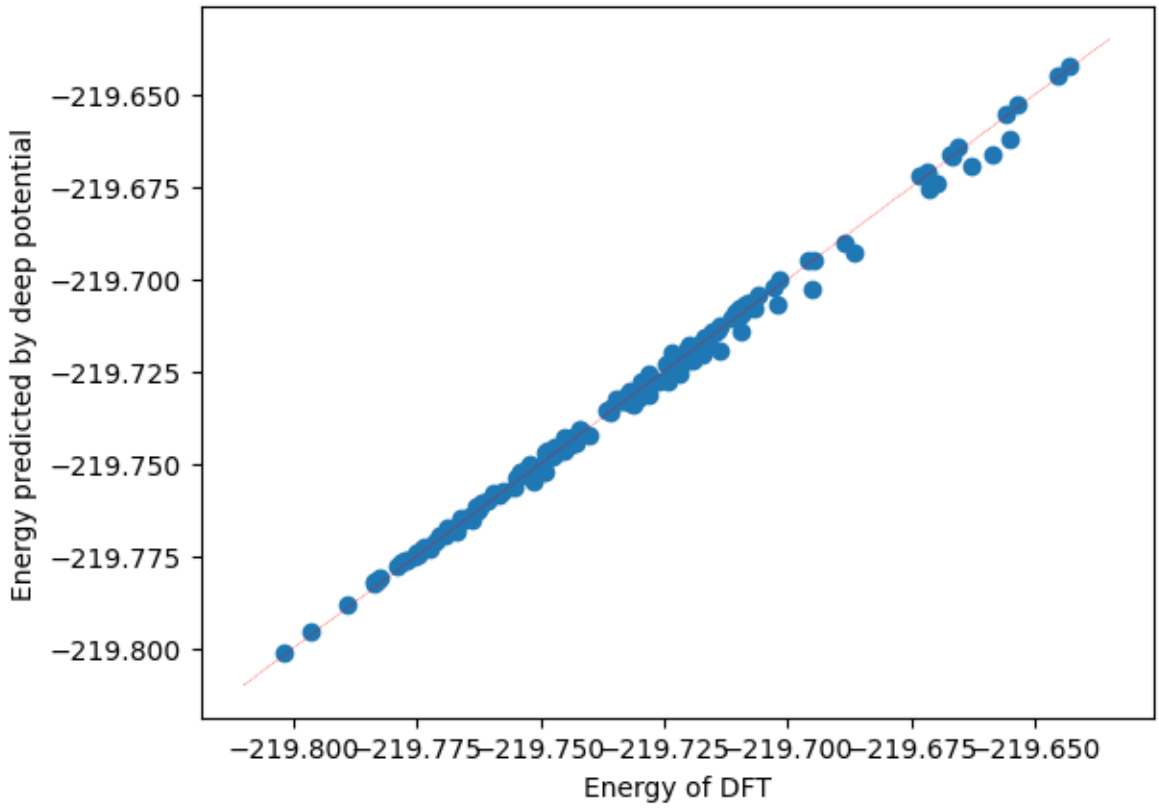
```
import matplotlib.pyplot as plt
import numpy as np

plt.scatter(training_systems["energies"], predict["energies"])

x_range = np.linspace(plt.xlim()[0], plt.xlim()[1])

plt.plot(x_range, x_range, "r--", linewidth=0.25)
plt.xlabel("Energy of DFT")
plt.ylabel("Energy predicted by deep potential")
plt.plot()
```

```
[]
```



Run MD with LAMMPS

The model can drive molecular dynamics in LAMMPS.

```
! ls
! cd ./DeePMD-kit_Tutorial/02.lmp && cp ../01.train.finished/graph.pb ./ && tree -L 1
```

```
DeePMD-kit_Tutorial
.
   ch4.dump
   conf.lmp
   graph.pb
   in.lammps
   log.lammps

0 directories, 5 files
```

Here `conf.lmp` gives the initial configuration of a gas phase methane MD simulation, and the file `in.lammps` is the LAMMPS input script. One may check in.lammps and finds that it is a rather standard LAMMPS input file for a MD simulation, with only two exception lines:

```
pair_style  deepmd graph.pb
pair_coeff  * *
```

where the pair style deepmd is invoked and the model file `graph.pb` is provided, which means the atomic interaction will be computed by the DP model that is stored in the file `graph.pb`.

In an environment with a compatibable version of LAMMPS, the deep potential molecular dynamics can be performed via

```
lmp -i input.lammps
```

```
! cd ./DeePMD-kit_Tutorial/02.lmp && lmp -i in.lammps
```

```
LAMMPS (2 Aug 2023 - Update 1)
OMP_NUM_THREADS environment is not set. Defaulting to 1 thread. (src/comm.cpp:98)
  using 1 OpenMP thread(s) per MPI task
Loaded 1 plugins from /root/miniconda3/envs/deepmd/lib/deepmd_lmp
Reading data file ...
  triclinic box = (0 0 0) to (10.114259 10.263124 10.216793) with tilt (0.036749877 0.13833062 -0.
↪056322169)
  1 by 1 by 1 MPI processor grid
  reading atoms ...
  5 atoms
  read_data CPU = 0.002 seconds
DeePMD-kit WARNING: Environmental variable OMP_NUM_THREADS is not set. Tune OMP_NUM_THREADS for␣
↪the best performance. See https://deepmd.rtfd.io/parallelism/ for more information.
Summary of lammps deepmd module ...
  >>> Info of deepmd-kit:
  installed to:       /root/miniconda3/envs/deepmd
  source:             v2.2.7
  source branch:       HEAD
  source commit:      839f4fe7
  source commit at:   2023-10-27 21:10:24 +0800
  surpport model ver.:1.1
  build variant:      cpu
  build with tf inc:  /root/miniconda3/envs/deepmd/include;/root/miniconda3/envs/deepmd/include
  build with tf lib:  /root/miniconda3/envs/deepmd/lib/libtensorflow_cc.so
  set tf intra_op_parallelism_threads: 0
  set tf inter_op_parallelism_threads: 0
  >>> Info of lammps module:
  use deepmd-kit at:  /root/miniconda3/envs/deepmdDeePMD-kit WARNING: Environmental variable OMP_
↪NUM_THREADS is not set. Tune OMP_NUM_THREADS for the best performance. See https://deepmd.rtfd.
↪io/parallelism/ for more information.
2024-03-24 23:05:49.768736: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow␣
↪binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU␣
↪instructions in performance-critical operations:  SSE4.1 SSE4.2 AVX AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-03-24 23:05:49.770401: I tensorflow/core/common_runtime/process_util.cc:146] Creating new␣
↪thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads for best␣
↪performance.
2024-03-24 23:05:49.817983: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:354] MLIR␣
↪V1 optimization pass is not enabled
INVALID_ARGUMENT: Tensor spin_attr/ntypes_spin:0, specified in either feed_devices or fetch_
↪devices was not found in the Graph
  >>> Info of model(s):
  using   1 model(s): graph.pb
  rcut in model:      6
  ntypes in model:    2

CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE

Your simulation uses code contributions which should be cited:
- USER-DEEPMD package:
```

(continues on next page)

(continued from previous page)

```
The log file lists these citations in BibTeX format.

CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE

Generated 0 of 1 mixed pair_coeff terms from geometric mixing rule
Neighbor list info ...
  update: every = 10 steps, delay = 0 steps, check = no
  max neighbors/atom: 2000, page size: 100000
  master list distance cutoff = 7
  ghost atom cutoff = 7
  binsize = 3.5, bins = 3 3 3
  1 neighbor lists, perpetual/occasional/extra = 1 0 0
  (1) pair deepmd, perpetual
      attributes: full, newton on
      pair build: full/bin/atomonly
      stencil: full/bin/3d
      bin: standard
Setting up Verlet run ...
  Unit style    : metal
  Current step  : 0
  Time step     : 0.001
Per MPI rank memory allocation (min/avg/max) = 2.559 | 2.559 | 2.559 Mbytes
   Step         PotEng         KinEng         TotEng          Temp          Press          Volume
         0  -219.77409      0.025852029   -219.74824      50           -799.80566      1060.5429
       100  -219.77101      0.02250472    -219.7485       43.526023    -563.15562      1060.5429
       200  -219.77525      0.025722761   -219.74953      49.749984    -55.768826      1060.5429
       300  -219.78111      0.030123111   -219.75098      58.260632     415.50143      1060.5429
       400  -219.78545      0.03264184    -219.7528       63.132067     724.77655      1060.5429
       500  -219.7897       0.034591934   -219.75511      66.903712     664.01323      1060.5429
       600  -219.78944      0.031599794   -219.75784      61.116661     307.82983      1060.5429
       700  -219.78389      0.023121639   -219.76076      44.719197    -166.66606      1060.5429
       800  -219.77712      0.013122374   -219.764        25.379775    -493.10259      1060.5429
       900  -219.7791       0.011293959   -219.76781      21.843468    -609.86395      1060.5429
      1000  -219.78712      0.01531002    -219.77181      29.610866    -422.5828       1060.5429
      1100  -219.7939       0.018709632   -219.77519      36.186003    -61.443156      1060.5429
      1200  -219.79395      0.016606919   -219.77734      32.11918      331.62678      1060.5429
      1300  -219.79132      0.012642575   -219.77868      24.451803     505.6361       1060.5429
      1400  -219.79314      0.013255468   -219.77989      25.637191     381.73541      1060.5429
      1500  -219.79509      0.014397006   -219.78069      27.845022     48.696022      1060.5429
      1600  -219.79313      0.012485864   -219.78064      24.148711    -302.67659      1060.5429
      1700  -219.78841      0.0085717658  -219.77983      16.578516    -476.08062      1060.5429
      1800  -219.78663      0.0081557171  -219.77847      15.773843    -407.83792      1060.5429
      1900  -219.78715      0.010996426   -219.77615      21.268013    -98.699573      1060.5429
      2000  -219.78836      0.016278673   -219.77209      31.484324     293.02315      1060.5429
      2100  -219.78819      0.022161035   -219.76603      42.861306     587.40225      1060.5429
      2200  -219.79165      0.031838471   -219.75981      61.578284     543.58893      1060.5429
      2300  -219.79343      0.038239208   -219.75519      73.957846     104.54643      1060.5429
      2400  -219.78301      0.031060153   -219.75195      60.072951    -293.72903      1060.5429
      2500  -219.77209      0.022352657   -219.74974      43.231919    -606.61353      1060.5429
      2600  -219.76604      0.017305685   -219.74873      33.47065     -623.66583      1060.5429
      2700  -219.77552      0.026563069   -219.74895      51.375211    -332.34033      1060.5429
      2800  -219.78594      0.0362724     -219.74967      70.153875     120.73427      1060.5429
      2900  -219.78868      0.038558744   -219.75012      74.575856     542.93567      1060.5429
      3000  -219.78351      0.03281317    -219.75069      63.463433     746.24646      1060.5429
      3100  -219.78106      0.028937414   -219.75212      55.967395     583.87016      1060.5429
      3200  -219.77929      0.025275432   -219.75402      48.884814     128.24387      1060.5429
      3300  -219.77781      0.022017978   -219.75579      42.584622    -395.55332      1060.5429
```

(continues on next page)

(continued from previous page)

```
      3400   -219.77696      0.019305132   -219.75765      37.33775       -679.74745      1060.5429
      3500   -219.78369      0.023714356   -219.75997      45.86556       -656.9891       1060.5429
      3600   -219.79244      0.030071312   -219.76237      58.160448      -354.34542      1060.5429
      3700   -219.79168      0.027557568   -219.76412      53.298657       199.00964      1060.5429
      3800   -219.78639      0.021137515   -219.76525      40.881734       596.54224      1060.5429
      3900   -219.77923      0.012972221   -219.76626      25.089367       713.41996      1060.5429
      4000   -219.78185      0.014202505   -219.76765      27.46884        430.83529      1060.5429
      4100   -219.78477      0.016041208   -219.76872      31.025047      -28.605377      1060.5429
      4200   -219.78545      0.016332231   -219.76912      31.587909      -457.5328       1060.5429
      4300   -219.78602      0.016882726   -219.76914      32.652612      -608.55966      1060.5429
      4400   -219.78949      0.020680419   -219.76881      39.99767       -456.72943      1060.5429
      4500   -219.79121      0.023411938   -219.7678       45.280658      -79.406734      1060.5429
      4600   -219.7882       0.022574198   -219.76562      43.660398       414.11955      1060.5429
      4700   -219.78521      0.022736692   -219.76248      43.974676       663.73939      1060.5429
      4800   -219.7834       0.025050214   -219.75835      48.449222       598.39611      1060.5429
      4900   -219.78291      0.030199797   -219.75271      58.408949       203.75805      1060.5429
      5000   -219.77611      0.030245158   -219.74586      58.496682      -300.80549      1060.5429
Loop time of 38.8363 on 1 procs for 5000 steps with 5 atoms

Performance: 11.124 ns/day, 2.158 hours/ns, 128.746 timesteps/s, 643.728 atom-step/s
104.3% CPU use with 1 MPI tasks x 1 OpenMP threads

MPI task timing breakdown:
Section |  min time  |  avg time  |  max time  |%varavg| %total
---------------------------------------------------------------
Pair    | 38.703     | 38.703     | 38.703     |   0.0 | 99.66
Neigh   | 0.0079815  | 0.0079815  | 0.0079815  |   0.0 |  0.02
Comm    | 0.0334     | 0.0334     | 0.0334     |   0.0 |  0.09
Output  | 0.0065195  | 0.0065195  | 0.0065195  |   0.0 |  0.02
Modify  | 0.070599   | 0.070599   | 0.070599   |   0.0 |  0.18
Other   |            | 0.01491    |            |       |  0.04

Nlocal:              5 ave           5 max           5 min
Histogram: 1 0 0 0 0 0 0 0 0 0
Nghost:            130 ave         130 max         130 min
Histogram: 1 0 0 0 0 0 0 0 0 0
Neighs:              0 ave           0 max           0 min
Histogram: 1 0 0 0 0 0 0 0 0 0
FullNghs:           20 ave          20 max          20 min
Histogram: 1 0 0 0 0 0 0 0 0 0

Total # of neighbors = 20
Ave neighs/atom = 4
Neighbor list builds = 500
Dangerous builds not checked
Total wall time: 0:00:39
```

CHAPTER

TWO

BACKEND

2.1 Supported backends

DeePMD-kit supports multiple backends: TensorFlow and PyTorch. To use DeePMD-kit, you must install at least one backend. Each backend does not support all features. In the documentation, TensorFlow and PyTorch icons are used to mark whether a backend supports a feature.

2.1.1 TensorFlow

- Model filename extension: `.pb`
- Checkpoint filename extension: `.meta`, `.index`, `.data-00000-of-00001`

TensorFlow 2.2 or above is required. DeePMD-kit does not use the TensorFlow v2 API but uses the TensorFlow v1 API (`tf.compat.v1`) in the graph mode.

2.1.2 PyTorch

- Model filename extension: `.pth`
- Checkpoint filename extension: `.pt`

PyTorch 2.0 or above is required. While `.pth` and `.pt` are the same in the PyTorch package, they have different meanings in the DeePMD-kit to distinguish the model and the checkpoint.

2.1.3 DP

Note: This backend is only for development and should not take into production.

- Model filename extension: `.dp`

DP is a reference backend for development, which uses pure NumPy to implement models without using any heavy deep-learning frameworks. Due to the limitation of NumPy, it doesn't support gradient calculation and thus cannot be used for training. As a reference backend, it is not aimed at the best performance, but only the correct results. The DP backend uses HDF5 to store model serialization data, which is backend-independent. Only Python inference interface can load this format.

2.2 Switch the backend

2.2.1 Training

When training and freezing a model, you can use `dp --tf` or `dp --pt` in the command line to switch the backend.

2.2.2 Inference

When doing inference, DeePMD-kit detects the backend from the model filename. For example, when the model filename ends with `.pb` (the ProtoBuf file), DeePMD-kit will consider it using the TensorFlow backend.

2.3 Convert model files between backends

If a model is supported by two backends, one can use *dp convert-backend* to convert the model file between these two backends.

Warning: Currently, only the `se_e2_a` model fully supports the backend conversion between TensorFlow and PyTorch.

CHAPTER

THREE

INSTALLATION

3.1 Easy install

There are various easy methods to install DeePMD-kit. Choose one that you prefer. If you want to build by yourself, jump to the next two sections.

After your easy installation, DeePMD-kit (`dp`) and LAMMPS (`lmp`) will be available to execute. You can try `dp -h` and `lmp -h` to see the help. `mpirun` is also available considering you may want to train models or run LAMMPS in parallel.

Note: Note: The off-line packages and conda packages require the GNU C Library 2.17 or above. The GPU version requires compatible NVIDIA driver to be installed in advance. It is possible to force conda to override detection when installation, but these requirements are still necessary during runtime. You can refer to DeepModeling conda FAQ for more information.

Note: Python 3.8 or above is required for Python interface.

- Install off-line packages
- Install with conda
- Install with docker
- Install Python interface with pip

3.1.1 Install off-line packages

Both CPU and GPU version offline packages are available on the Releases page.

Some packages are split into two files due to the size limit of GitHub. One may merge them into one after downloading:

```
cat deepmd-kit-2.2.9-cuda118-Linux-x86_64.sh.0 deepmd-kit-2.2.9-cuda118-Linux-x86_64.sh.1 > deepmd-
↪kit-2.2.9-cuda118-Linux-x86_64.sh
```

One may enable the environment using

```
conda activate /path/to/deepmd-kit
```

3.1.2 Install with conda

DeePMD-kit is available with conda. Install Anaconda, Miniconda, or miniforge first. You can refer to DeepModeling conda FAQ for how to setup a conda environment.

conda-forge channel

DeePMD-kit is available on the conda-forge channel:

```
conda create -n deepmd deepmd-kit lammps horovod -c conda-forge
```

The supported platforms include Linux x86-64, macOS x86-64, and macOS arm64. Read conda-forge FAQ to learn how to install CUDA-enabled packages.

Official channel (deprecated)

> Danger: Deprecated since version 3.0.0: The official channel has been deprecated since 3.0.0, due to the challenging work of building dependencies for multiple backends. Old packages will still be available at https://conda.deepmodeling.com. Maintainers will build packages in the conda-forge organization together with other conda-forge members.

One may create an environment that contains the CPU version of DeePMD-kit and LAMMPS:

```
conda create -n deepmd deepmd-kit=*=*cpu libdeepmd=*=*cpu lammps -c https://conda.deepmodeling.com␣
↪-c defaults
```

Or one may want to create a GPU environment containing CUDA Toolkit:

```
conda create -n deepmd deepmd-kit=*=*gpu libdeepmd=*=*gpu lammps cudatoolkit=11.6 horovod -c␣
↪https://conda.deepmodeling.com -c defaults
```

One could change the CUDA Toolkit version from `10.2` or `11.6`.

One may specify the DeePMD-kit version such as `2.2.9` using

```
conda create -n deepmd deepmd-kit=2.2.9=*cpu libdeepmd=2.2.9=*cpu lammps horovod -c https://conda.
↪deepmodeling.com -c defaults
```

One may enable the environment using

```
conda activate deepmd
```

3.1.3 Install with docker

A docker for installing the DeePMD-kit is available here.

To pull the CPU version:

```
docker pull ghcr.io/deepmodeling/deepmd-kit:2.2.8_cpu
```

To pull the GPU version:

```
docker pull ghcr.io/deepmodeling/deepmd-kit:2.2.8_cuda12.0_gpu
```

3.1.4 Install Python interface with pip

If you have no existing TensorFlow installed, you can use `pip` to install the pre-built package of the Python interface with CUDA 12 supported:

```
pip install deepmd-kit[gpu,cu12,torch]
```

`cu12` is required only when CUDA Toolkit and cuDNN were not installed.

To install the package built against CUDA 11.8, use

```
pip install torch --index-url https://download.pytorch.org/whl/cu118
pip install deepmd-kit-cu11[gpu,cu11]
```

Or install the CPU version without CUDA supported:

```
pip install torch --index-url https://download.pytorch.org/whl/cpu
pip install deepmd-kit[cpu]
```

The LAMMPS module and the i-Pi driver are only provided on Linux and macOS for the TensorFlow backend. To install LAMMPS and/or i-Pi, add `lmp` and/or `ipi` to extras:

```
pip install deepmd-kit[gpu,cu12,torch,lmp,ipi]
```

MPICH is required for parallel running. (The macOS arm64 package doesn't support MPI yet.)

It is suggested to install the package into an isolated environment. The supported platform includes Linux x86-64 and aarch64 with GNU C Library 2.28 or above, macOS x86-64 and arm64, and Windows x86-64. A specific version of TensorFlow and PyTorch which is compatible with DeePMD-kit will be also installed.

> Warning: If your platform is not supported, or you want to build against the installed TensorFlow, or you want to enable ROCM support, please build from source.

3.2 Install from source code

Please follow our GitHub webpage to download the latest released version and development version.

Or get the DeePMD-kit source code by `git clone`

```
cd /some/workspace
git clone https://github.com/deepmodeling/deepmd-kit.git deepmd-kit
```

For convenience, you may want to record the location of the source to a variable, saying `deepmd_source_dir` by

```
cd deepmd-kit
deepmd_source_dir=`pwd`
```

3.2.1 Install the Python interface

Install Backend's Python interface

First, check the Python version on your machine. Python 3.8 or above is required.

```
python --version
```

We follow the virtual environment approach to install the backend's Python interface. Now we assume that the Python interface will be installed in the virtual environment directory `$deepmd_venv`:

```
virtualenv -p python3 $deepmd_venv
source $deepmd_venv/bin/activate
pip install --upgrade pip
```

TensorFlow

The full instruction to install TensorFlow can be found on the official TensorFlow website. TensorFlow 2.2 or later is supported.

```
pip install --upgrade tensorflow
```

If one does not need the GPU support of DeePMD-kit and is concerned about package size, the CPU-only version of TensorFlow should be installed by

```
pip install --upgrade tensorflow-cpu
```

One can also use conda to install TensorFlow from conda-forge.

To verify the installation, run

```
python -c "import tensorflow as tf;print(tf.reduce_sum(tf.random.normal([1000, 1000])))"
```

One can also build the TensorFlow Python interface from source for customized hardware optimization, such as CUDA, ROCM, or OneDNN support.

PyTorch

To install PyTorch, run

```
pip install torch
```

Follow PyTorch documentation to install PyTorch built against different CUDA versions or without CUDA.

One can also use conda to install PyTorch from conda-forge.

It is important that every time a new shell is started and one wants to use `DeePMD-kit`, the virtual environment should be activated by

```
source $deepmd_venv/bin/activate
```

if one wants to skip out of the virtual environment, he/she can do

```
deactivate
```

If one has multiple python interpreters named something like python3.x, it can be specified by, for example

```
virtualenv -p python3.8 $deepmd_venv
```

One should remember to activate the virtual environment every time he/she uses DeePMD-kit.

Install the DeePMD-kit's python interface

Check the compiler version on your machine

```
gcc --version
```

The compiler GCC 4.8 or later is supported in the DeePMD-kit.

TensorFlow

Note that TensorFlow may have specific requirements for the compiler version to support the C++ standard version and `_GLIBCXX_USE_CXX11_ABI` used by TensorFlow. It is recommended to use the same compiler version as TensorFlow, which can be printed by `python -c "import tensorflow;print(tensorflow.version.COMPILER_VERSION)"`.

PyTorch

You can set the environment variable `export DP_ENABLE_PYTORCH=1` to enable customized C++ OPs in the PyTorch backend. Note that PyTorch may have specific requirements for the compiler version to support the C++ standard version and `_GLIBCXX_USE_CXX11_ABI` used by PyTorch.

The customized C++ OPs are not enabled by default because TensorFlow and PyTorch packages from the PyPI use different `_GLIBCXX_USE_CXX11_ABI` flags. We recommend conda-forge packages in this case.

Execute

```
cd $deepmd_source_dir
pip install .
```

One may set the following environment variables before executing `pip`:

Environment variables	Allowed value	Default value	Usage
DP_VARI	`cpu`, `cuda` `rocm`	`cpu`	Build CPU variant or GPU variant with CUDA or ROCM support.
CUDA-Toolkit_R	Path	Detected automatically	The path to the CUDA toolkit directory. CUDA 9.0 or later is supported. NVCC is required.
ROCM_R	Path	Detected automatically	The path to the ROCM toolkit directory.
DP_ENAB	0, 1	1	Enable the TensorFlow backend.
DP_ENAB	0, 1	0	Enable customized C++ OPs for the PyTorch backend. PyTorch can still run without customized C++ OPs, but features will be limited.
TENSORFLOW_R	Path	Detected automatically	The path to TensorFlow Python library. By default the installer only finds TensorFlow under user site-package directory (`site.getusersitepackages()`) or system site-package directory (`sysconfig.get_path("purelib")`) due to limitation of PEP-517. If not found, the latest TensorFlow (or the environment variable `TENSORFLOW_VERSION` if given) from PyPI will be built against.
DP_ENAB	0, 1	0	Enable compilation optimization for the native machine's CPU type. Do not enable it if generated code will run on different CPUs.
CMAKE_A	str	-	Additional CMake arguments
<LANG>I (`<LANG>=C` `CUDA` or `HIP`)	str	-	Default compilation flags to be used when compiling `<LANG>` files. See CMake documentation.

To test the installation, one should first jump out of the source directory

```
cd /some/other/workspace
```

then execute

```
dp -h
```

It will print the help information like

```
usage: dp [-h] {train,freeze,test} ...
```

(continues on next page)

(continued from previous page)

```
DeePMD-kit: A deep learning package for many-body potential energy
representation and molecular dynamics

optional arguments:
  -h, --help            show this help message and exit

Valid subcommands:
  {train,freeze,test}
    train               train a model
    freeze              freeze the model
    test                test the model
```

Install horovod and mpi4py

Horovod and mpi4py are used for parallel training. For better performance on GPU, please follow the tuning steps in Horovod on GPU.

```
# With GPU, prefer NCCL as a communicator.
HOROVOD_WITHOUT_GLOO=1 HOROVOD_WITH_TENSORFLOW=1 HOROVOD_GPU_OPERATIONS=NCCL HOROVOD_NCCL_HOME=/
↪path/to/nccl pip install horovod mpi4py
```

If your work in a CPU environment, please prepare runtime as below:

```
# By default, MPI is used as communicator.
HOROVOD_WITHOUT_GLOO=1 HOROVOD_WITH_TENSORFLOW=1 pip install horovod mpi4py
```

To ensure Horovod has been built with proper framework support enabled, one can invoke the `horovodrun --check-build` command, e.g.,

```
$ horovodrun --check-build

Horovod v0.22.1:

Available Frameworks:
    [X] TensorFlow
    [X] PyTorch
    [ ] MXNet

Available Controllers:
    [X] MPI
    [X] Gloo

Available Tensor Operations:
    [X] NCCL
    [ ] DDL
    [ ] CCL
    [X] MPI
    [X] Gloo
```

Since version 2.0.1, Horovod and mpi4py with MPICH support are shipped with the installer.

If you don't install Horovod, DeePMD-kit will fall back to serial mode.

3.2.2 Install the C++ interface

If one does not need to use DeePMD-kit with Lammps or I-Pi, then the python interface installed in the previous section does everything and he/she can safely skip this section.

Install Backends' C++ interface (optional)

TensorFlow

Since TensorFlow 2.12, TensorFlow C++ library (`libtensorflow_cc`) is packaged inside the Python library. Thus, you can skip building TensorFlow C++ library manually. If that does not work for you, you can still build it manually.

The C++ interface of DeePMD-kit was tested with compiler GCC >= 4.8. It is noticed that the I-Pi support is only compiled with GCC >= 4.8. Note that TensorFlow may have specific requirements for the compiler version.

First, the C++ interface of Tensorflow should be installed. It is noted that the version of Tensorflow should be consistent with the python interface. You may follow the instruction or run the script `$deepmd_source_dir/source/install/build_tf.py` to install the corresponding C++ interface.

PyTorch

If you have installed PyTorch using pip, you can use libtorch inside the PyTorch Python package. You can also download libtorch prebuilt library from the PyTorch website.

Install DeePMD-kit's C++ interface

Now go to the source code directory of DeePMD-kit and make a building place.

```
cd $deepmd_source_dir/source
mkdir build
cd build
```

The installation requires CMake 3.16 or later for the CPU version, CMake 3.23 or later for the CUDA support, and CMake 3.21 or later for the ROCM support. One can install CMake via `pip` if it is not installed or the installed version does not satisfy the requirement:

```
pip install -U cmake
```

You must enable at least one backend. If you enable two or more backends, these backend libraries must be built in a compatible way, e.g. using the same `_GLIBCXX_USE_CXX11_ABI` flag. We recommend using conda pacakges from conda-forge, which are usually compatible to each other.

TensorFlow

I assume you have activated the TensorFlow Python environment and want to install DeePMD-kit into path `$deepmd_root`, then execute CMake

```
cmake -DENABLE_TENSORFLOW=TRUE -DUSE_TF_PYTHON_LIBS=TRUE -DCMAKE_INSTALL_PREFIX=$deepmd_root ..
```

If you specify `-DUSE_TF_PYTHON_LIBS=FALSE`, you need to give the location where TensorFlow's C++ interface is installed to `-DTENSORFLOW_ROOT=${tensorflow_root}`.

PyTorch

I assume you have installed the PyTorch (either Python or C++ interface) to `$torch_root`, then execute CMake

```
cmake -DENABLE_PYTORCH=TRUE -DCMAKE_PREFIX_PATH=$torch_root -DCMAKE_INSTALL_PREFIX=$deepmd_root ..
```

You can specify `-DUSE_PT_PYTHON_LIBS=TRUE` to use libtorch from the Python installation, but you need to be careful that PyTorch PyPI packages are still built using `_GLIBCXX_USE_CXX11_ABI=0`, which may be not compatible with other libraries.

```
cmake -DENABLE_PYTORCH=TRUE -DUSE_PT_PYTHON_LIBS=TRUE -DCMAKE_INSTALL_PREFIX=$deepmd_root ..
```

One may add the following arguments to `cmake`:

CMake Aurgements	Allowed value	Default value	Usage
-DENABLE_TENSORFLO	TRUE or FALSE	FALSE	Whether building the TensorFlow backend.
-DENABLE_PYTORCH=	TRUE or FALSE	FALSE	Whether building the PyTorch backend.
-DTENSORFLOW_ROOT	Path	-	The Path to TensorFlow's C++ interface.
-DCMAKE_INSTALL_PR	Path	-	The Path where DeePMD-kit will be installed.
-DUSE_CUDA_TOOLKIT	TRUE or FALSE	FALSE	If TRUE, Build GPU support with CUDA toolkit.
-DCUDAToolkit_ROOT=	Path	Detected automatically	The path to the CUDA toolkit directory. CUDA 9.0 or later is supported. NVCC is required.
-DUSE_ROCM_TOOLKIT	TRUE or FALSE	FALSE	If TRUE, Build GPU support with ROCM toolkit.
-DCMAKE_HIP_COMPIL	Path	Detected automatically	The path to the ROCM toolkit directory.
-DLAMMPS_SOURCE_RO	Path	-	Only neccessary for LAMMPS plugin mode. The path to the LAMMPS source code. LAMMPS 8Apr2021 or later is supported. If not assigned, the plugin mode will not be enabled.
-DUSE_TF_PYTHON_LI	TRUE or FALSE	FALSE	If TRUE, Build C++ interface with TensorFlow's Python libraries (TensorFlow's Python Interface is required). And there's no need for building TensorFlow's C++ interface.
-DUSE_PT_PYTHON_LI	TRUE or FALSE	FALSE	If TRUE, Build C++ interface with PyTorch's Python libraries (PyTorch's Python Interface is required). And there's no need for downloading PyTorch's C++ libraries.
-DENABLE_NATIVE_OP	TRUE or FALSE	FALSE	Enable compilation optimization for the native machine's CPU type. Do not enable it if generated code will run on different CPUs.
-DCMAKE_<LANG>_FL (<LANG>=CXX, CUDA or HIP)	str	-	Default compilation flags to be used when compiling <LANG> files. See CMake documentation.

If the CMake has been executed successfully, then run the following make commands to build the package:

```
make -j4
make install
```

Option `-j4` means using 4 processes in parallel. You may want to use a different number according to your hardware.

If everything works fine, you will have the executable and libraries installed in `$deepmd_root/bin` and `$deepmd_root/lib`

```
$ ls $deepmd_root/bin
$ ls $deepmd_root/lib
```

3.3 Install from pre-compiled C library

Note: Supported backends: TensorFlow

DeePMD-kit provides pre-compiled C library package (`libdeepmd_c.tar.gz`) in each release. It can be used to build the LAMMPS plugin and GROMACS patch, as well as many third-party software packages, without building TensorFlow and DeePMD-kit on one's own. It can be downloaded via the shell command:

```
wget https://github.com/deepmodeling/deepmd-kit/releases/latest/download/libdeepmd_c.tar.gz
tar xzf libdeepmd_c.tar.gz
```

The library is built in Linux (GLIBC 2.17) with CUDA 12.2 (`libdeepmd_c.tar.gz`) or 11.8 (`libdeepmd_c_cu11.tar.gz`). It's noted that this package does not contain CUDA Toolkit and cuDNN, so one needs to download them from the NVIDIA website.

3.3.1 Use Pre-compiled C Library to build the LAMMPS plugin, i-PI driver, and GROMACS patch

When one installs DeePMD-kit's C++ interface, one can use the CMake argument `DEEPMD_C_ROOT` to the path `libdeepmd_c`.

```
cd $deepmd_source_dir/source
mkdir build
cd build
cmake -DDEEPMD_C_ROOT=/path/to/libdeepmd_c -DCMAKE_INSTALL_PREFIX=$deepmd_root ..
make -j8
make install
```

Then the i-PI driver `dp_ipi` will be built and installed. One can also follow the manual Install LAMMPS and/or Install GROMACS.

3.4 Install LAMMPS

There are two ways to install LAMMPS: the built-in mode and the plugin mode. The built-in mode builds LAMMPS along with the DeePMD-kit and DeePMD-kit will be loaded automatically when running LAMMPS. The plugin mode builds LAMMPS and a plugin separately, so one needs to use `plugin load` command to load the DeePMD-kit's LAMMPS plugin library.

3.4.1 Install LAMMPS's DeePMD-kit module (built-in mode)

Before following this section, DeePMD-kit C++ interface should have be installed.

DeePMD-kit provides a module for running MD simulations with LAMMPS. Now make the DeePMD-kit module for LAMMPS.

```
cd $deepmd_source_dir/source/build
make lammps
```

DeePMD-kit will generate a module called `USER-DEEPMD` in the `build` directory, which supports either double or single float precision interface. Now download the LAMMPS code, and uncompress it.

```
cd /some/workspace
wget https://github.com/lammps/lammps/archive/stable_2Aug2023_update3.tar.gz
tar xf stable_2Aug2023_update3.tar.gz
```

The source code of LAMMPS is stored in the directory `lammps-stable_2Aug2023_update3`.

Then, you can build LAMMPS with either make or CMake.

With make

Now go into the LAMMPS code and copy the DeePMD-kit module like this

```
cd lammps-stable_2Aug2023_update3/src/
cp -r $deepmd_source_dir/source/build/USER-DEEPMD .
make yes-kspace
make yes-extra-fix
make yes-user-deepmd
```

You can enable any other package you want. Now build LAMMPS

```
make mpi -j4
```

If everything works fine, you will end up with an executable `lmp_mpi`.

```
./lmp_mpi -h
```

The DeePMD-kit module can be removed from the LAMMPS source code by

```
make no-user-deepmd
```

With CMake

Now go into the LAMMPS directory and create a directory called `build`:

```
mkdir -p lammps-stable_2Aug2023_update3/build/
cd lammps-stable_2Aug2023_update3/build/
```

Patch the LAMMPS `CMakeLists.txt` file:

```
echo "include(${deepmd_source_dir}/source/lmp/builtin.cmake)" >> ../cmake/CMakeLists.txt
```

It's expected to see one extra line in the end of `CMakeLists.txt`.

Now build LAMMPS. You can install any other package you want.

```
cmake -D LAMMPS_INSTALL_RPATH=ON -D BUILD_SHARED_LIBS=yes -D CMAKE_INSTALL_PREFIX=${deepmd_root} -
↪DCMAKE_PREFIX_PATH=${deepmd_root} ../cmake
make -j4
make install
```

If everything works fine, you will end up with an executable `${deepmd_root}/bin/lmp`.

```
${deepmd_root}/bin/lmp -h
```

3.4.2 Install LAMMPS (plugin mode)

Starting from `8Apr2021`, LAMMPS also provides a plugin mode, allowing one to build LAMMPS and a plugin separately.

Now download the LAMMPS code (`8Apr2021` or later), and uncompress it:

```
cd /some/workspace
wget https://github.com/lammps/lammps/archive/stable_2Aug2023_update3.tar.gz
tar xf stable_2Aug2023_update3.tar.gz
```

The source code of LAMMPS is stored in the directory `lammps-stable_2Aug2023_update3`. The directory of the source code should be specified as the CMAKE argument `LAMMPS_SOURCE_ROOT` during installation of the DeePMD-kit C++ interface. Now go into the LAMMPS directory and create a directory called `build`

```
mkdir -p lammps-stable_2Aug2023_update3/build/
cd lammps-stable_2Aug2023_update3/build/
```

Now build LAMMPS. Note that `PLUGIN` must be enabled, and `BUILD_SHARED_LIBS` must be set to `yes`. You can install any other package you want.

```
cmake -D PKG_PLUGIN=ON -D LAMMPS_INSTALL_RPATH=ON -D BUILD_SHARED_LIBS=yes -D CMAKE_INSTALL_PREFIX=
↪${deepmd_root} -D CMAKE_INSTALL_LIBDIR=lib -D CMAKE_INSTALL_FULL_LIBDIR=${deepmd_root}/lib ../
↪cmake
make -j4
make install
```

If everything works fine, you will end up with an executable `${deepmd_root}/bin/lmp`.

```
${deepmd_root}/bin/lmp -h
```

Note: If `${tensorflow_root}`, `${deepmd_root}`, or the path to TensorFlow Python package if applicable is different from the prefix of LAMMPS, you need to append the library path to `RUNPATH` of `liblammps.so`. For example, use patchelf >= 0.13

```
patchelf --add-rpath "${tensorflow_root}/lib" liblammps.so
```

3.5 Install i-PI

The i-PI works in a client-server model. The i-PI provides the server for integrating the replica positions of atoms, while the DeePMD-kit provides a client named `dp_ipi` that computes the interactions (including energy, forces and virials). The server and client communicate via the Unix domain socket or the Internet socket. Full documentation for i-PI can be found here. The source code and a complete installation guide for i-PI can be found here. To use i-PI with already existing drivers, install and update using Pip:

```
pip install -U i-PI
```

Test with Pytest:

```
pip install pytest
pytest --pyargs ipi.tests
```

3.6 Install GROMACS with DeepMD

Before following this section, DeePMD-kit C++ interface should have be installed.

3.6.1 Patch source code of GROMACS

Download the source code of a supported GROMACS version (2020.2) from https://manual.gromacs.org/2020.2/download.html. Run the following command:

```
export PATH=$PATH:$deepmd_kit_root/bin
dp_gmx_patch -d $gromacs_root -v $version -p
```

where `deepmd_kit_root` is the directory where the latest version of DeePMD-kit is installed, and `gromacs_root` refers to the source code directory of GROMACS. And `version` represents the version of GROMACS, where only 2020.2 is supported now. If attempting to patch another version of GROMACS you will still need to set `version` to `2020.2` as this is the only supported version, we cannot guarantee that patching other versions of GROMACS will work.

3.6.2 Compile GROMACS with deepmd-kit

The C++ interface of `Deepmd-kit 2.x` and `TensorFlow 2.x` are required. And be aware that only DeePMD-kit with high precision is supported now since we cannot ensure single precision is enough for a GROMACS simulation. Here is a sample compile script:

```
#!/bin/bash
export CC=/usr/bin/gcc
export CXX=/usr/bin/g++
export CMAKE_PREFIX_PATH="/path/to/fftw-3.3.9" # fftw libraries
mkdir build
cd build

cmake3 .. -DCMAKE_CXX_STANDARD=14 \ # not required, but c++14 seems to be more compatible with␣
↪higher version of tensorflow
          -DGMX_MPI=ON \
          -DGMX_GPU=CUDA \ # Gromacs on ROCm has not been fully developed yet
          -DCUDAToolkit_ROOT=/path/to/cuda \
          -DCMAKE_INSTALL_PREFIX=/path/to/gromacs-2020.2-deepmd
make -j
make install
```

3.7 Building conda packages

> Danger: Deprecated since version 3.0.0: The official channel has been deprecated since 3.0.0. Refer to conda-forge documentation for how to contribute and build packages locally.

One may want to keep both convenience and personalization of the DeePMD-kit. To achieve this goal, one can consider building conda packages. We provide building scripts in deepmd-kit-recipes organization. These building tools are driven by conda-build and conda-smithy.

For example, if one wants to turn on `MPIIO` package in LAMMPS, go to `lammps-feedstock` repository and modify `recipe/build.sh`. `-D PKG_MPIIO=OFF` should be changed to `-D PKG_MPIIO=ON`. Then go to the main directory and execute

```
./build-locally.py
```

This requires that Docker has been installed. After the building, the packages will be generated in `build_artifacts/linux-64` and `build_artifacts/noarch`, and then one can install then executing

```
conda create -n deepmd lammps -c file:///path/to/build_artifacts -c https://conda.deepmodeling.com␣
↪-c nvidia
```

One may also upload packages to one's Anaconda channel, so they can be installed on other machines:

```
anaconda upload /path/to/build_artifacts/linux-64/*.tar.bz2 /path/to/build_artifacts/noarch/*.tar.
↪bz2
```

3.8 Install Node.js interface

DeePMD-kit has an inference interface for Node.js, the most common programming language in the world, via a wrapper of the header-only C++ interface created by SWIG.

3.8.1 Install from npm

```
npm i deepmd-kit
# Or if you want to install globally
npm i -g deepmd-kit
```

3.8.2 Build from source

Before building DeePMD-kit, install Node.js, SWIG (v4.1.0 for Node.js v12-v18 support), and node-gyp globally.

When using CMake to build DeePMD-kit from source, set argument `BUILD_NODEJS_IF=ON` and `NODEJS_INCLUDE_DIRS=/path/to/nodejs/include` (the path to the include directory of Node.js):

```
cmake -D BUILD_NODEJS_IF=ON \
      -D NODEJS_INCLUDE_DIRS=/path/to/nodejs/include \
      .. # and other arguments
make
make install
```

After installing DeePMD-kit, two files, `bind.gyp` and `deepmdJAVASCRIPT_wrap.cxx` will be generated in `$deepmd_source_dir/source/nodejs`.

Go to this directory, and install the Node.js package globally:

```
cd $deepmd_source_dir/source/nodejs
npm i
npm link
```

The `deepmd-kit` package should be globally available in Node.js environments:

```
const deepmd = require("deepmd-kit");
```

3.9 Easy install the latest development version

DeePMD-kit is actively developed in the `devel` branch. The documentation of the `latest` version matches the `devel` branch.

The following is the way to install the pre-compiled packages without building from source. All of them are built with GitHub Actions.

3.9.1 Install with docker

The `devel` tag is used to mark the latest development version with CUDA 12.2 support:

```
docker pull ghcr.io/deepmodeling/deepmd-kit:devel
```

For CUDA 11.8 support, use the `devel_cu11` tag.

3.9.2 Install with pip

Below is an one-line shell command to download the artifact containing wheels and install it with `pip`:

```
pip install -U --pre deepmd-kit[gpu,cu12,lmp,torch] --extra-index-url https://deepmodeling.github.
↪io/deepmd-kit/simple
```

`cu12` and `lmp` are optional, which is the same as the stable version.

3.9.3 Download pre-compiled C Library

Note: Supported backends: TensorFlow

The pre-comiled C library can be downloaded from here, or via a shell command:

```
wget https://nightly.link/deepmodeling/deepmd-kit/workflows/package_c/devel/libdeepmd_c-0-
↪libdeepmd_c.tar.gz.zip && unzip libdeepmd_c-0-libdeepmd_c.tar.gz.zip
```

3.9.4 Pre-release conda-forge packages

Pre-release conda-forge packages are in `conda-forge/label/deepmd-kit_dev` or `conda-forge/label/deepmd-kit_rc` channels, other than the `conda-forge` channel. See conda-forge documentation for more information.

CHAPTER

FOUR

DATA

In this section, we will introduce how to convert the DFT-labeled data into the data format used by DeePMD-kit.

The DeePMD-kit organizes data in `systems`. Each `system` is composed of a number of `frames`. One may roughly view a `frame` as a snapshot of an MD trajectory, but it does not necessarily come from an MD simulation. A `frame` records the coordinates and types of atoms, cell vectors if the periodic boundary condition is assumed, energy, atomic forces and virials. It is noted that the `frames` in one `system` share the same number of atoms with the same type.

4.1 System

DeePMD-kit takes a system as the data structure. A snapshot of a system is called a frame. A system may contain multiple frames with the same atom types and numbers, i.e. the same formula (like `H2O`). To contains data with different formulas, one usually needs to divide data into multiple systems, which may sometimes result in sparse-frame systems. See a new system format to further combine different systems with the same atom numbers, when training with descriptor `se_atten`.

A system should contain system properties, input frame properties, and labeled frame properties. The system property contains the following property:

ID	Property	Raw file	Required	Shape	Description
type	Atom type indexes	type.	Required	Nato	Integers that start with 0. If both the training parameter type_map is set and `type_map.raw` is provided, the system atom type should be mapped to `type_map.raw` in `type.raw` and will be mapped to the model atom type when training; otherwise, the system atom type will be always mapped to the model atom type (whether type_map is set or not)
type_	Atom type name	type_	Optional	Ntyp	Atom names that map to atom type, which is unnecessary to be contained in the periodic table. Only works when the training parameter type_map is set
nopb	Non-periodic system	nopb	Optional	1	If True, this system is non-periodic; otherwise it's periodic

The input frame properties contain the following property, the first axis of which is the number of frames:

ID	Property	Raw file	Unit	Required/Optic	Shape	Description
coord	Atomic coordinates	coord.raw	Å	Required	Nframes * Natoms * 3	
box	Boxes	box.raw	Å	Required if periodic	Nframes * 3 * 3	in the order `XX XY XZ YX YY YZ ZX ZY ZZ`
fparam	Extra frame parameters	fparam.	Any	Optional	Nframes * Any	
aparam	Extra atomic parameters	aparam	Any	Optional	Nframes * aparam * Any	
numb_c	Each frame is copied by the `numb_copy` (int) times	prob.ra	1	Optional	Nframes	Integer; Default is 1 for all frames

The labeled frame properties are listed as follows, all of which will be used for training if and only if the loss function contains such property:

ID	Property	Raw file	Unit	Shape	Description
energy	Frame energies	energy.raw	eV	Nframes	
force	Atomic forces	force.raw	eV/Å	Nframes * Natoms * 3	
virial	Frame virial	virial.raw	eV	Nframes * 9	in the order `XX XY XZ YX YY YZ ZX ZY ZZ`
atom_ener	Atomic energies	atom_ener.r	eV	Nframes * Natoms	
atom_pref	Weights of atomic forces	atom_pref.r	1	Nframes * Natoms	
dipole	Frame dipole	dipole.raw	Any	Nframes * 3	
atomic_di	Atomic dipole	atomic_dipo	Any	Nframes * Natoms * 3	
polariz-ability	Frame polarizability	polar-izabil-ity.raw	Any	Nframes * 9	in the order `XX XY XZ YX YY YZ ZX ZY ZZ`
atomic_po	Atomic polarizability	atomic_pola	Any	Nframes * Natoms * 9	in the order `XX XY XZ YX YY YZ ZX ZY ZZ`
drdq	Partial derivative of atomic coordinates with respect to generalized coordinates	drdq.raw	1	Nframes * Natoms * 3 * Ngen_coords	

In general, we always use the following convention of units:

Property	Unit
Time	ps
Length	Å
Energy	eV
Force	eV/Å
Virial	eV
Pressure	Bar

4.2 Formats of a system

Two binary formats, NumPy and HDF5, are supported for training. The raw format is not directly supported, but a tool is provided to convert data from the raw format to the NumPy format.

4.2.1 NumPy format

In a system with the Numpy format, the system properties are stored as text files ending with `.raw`, such as `type.raw` and `type_map.raw`, under the system directory. If one needs to train a non-periodic system, an empty `nopbc` file should be put under the system directory. Both input and labeled frame properties are saved as the NumPy binary data (NPY) files ending with `.npy` in each of the `set.*` directories. Take an example, a system may contain the following files:

```
type.raw
type_map.raw
nopbc
set.000/coord.npy
set.000/energy.npy
set.000/force.npy
set.001/coord.npy
set.001/energy.npy
set.001/force.npy
```

We assume that the atom types do not change in all frames. It is provided by `type.raw`, which has one line with the types of atoms written one by one. The atom types should be integers. For example the `type.raw` of a system that has 2 atoms with 0 and 1:

```
$ cat type.raw
0 1
```

Sometimes one needs to map the integer types to atom names. The mapping can be given by the file `type_map.raw`. For example

```
$ cat type_map.raw
O H
```

The type `0` is named by `"O"` and the type `1` is named by `"H"`.

For training models with descriptor `se_atten`, a new system format is supported to put together the frame-sparse systems with the same atom number.

4.2.2 HDF5 format

A system with the HDF5 format has the same structure as the Numpy format, but in an HDF5 file, a system is organized as an HDF5 group. The file name of a Numpy file is the key in an HDF5 file, and the data is the value of the key. One needs to use `#` in a DP path to divide the path to the HDF5 file and the HDF5 path:

```
/path/to/data.hdf5#/H2O
```

Here, `/path/to/data.hdf5` is the file path and `/H2O` is the HDF5 path. All HDF5 paths should start with `/`. There should be some data in the `H2O` group, such as `/H2O/type.raw` and `/H2O/set.000/force.npy`.

An HDF5 file with a large number of systems has better performance than multiple NumPy files in a large cluster.

4.2.3 Raw format and data conversion

A raw file is a plain text file with each information item written in one file and one frame written on one line. It's not directly supported, but we provide a tool to convert them.

In the raw format, the property of one frame is provided per line, ending with `.raw`. Take an example, the default files that provide box, coordinate, force, energy and virial are `box.raw`, `coord.raw`, `force.raw`, `energy.raw` and `virial.raw`, respectively. Here is an example of `force.raw`:

```
$ cat force.raw
-0.724  2.039 -0.951  0.841 -0.464  0.363
 6.737  1.554 -5.587 -2.803  0.062  2.222
-1.968 -0.163  1.020 -0.225 -0.789  0.343
```

This `force.raw` contains 3 frames with each frame having the forces of 2 atoms, thus it has 3 lines and 6 columns. Each line provides all the 3 force components of 2 atoms in 1 frame. The first three numbers are the 3 force components of the first atom, while the second three numbers are the 3 force components of the second atom. Other files are organized similarly. The number of lines of all raw files should be identical.

One can use the script `$deepmd_source_dir/data/raw/raw_to_set.sh` to convert the prepared raw files to the NumPy format. For example, if we have a raw file that contains 6000 frames,

```
$ ls
box.raw  coord.raw  energy.raw  force.raw  type.raw  virial.raw
$ $deepmd_source_dir/data/raw/raw_to_set.sh 2000
nframe is 6000
nline per set is 2000
will make 3 sets
making set 0 ...
making set 1 ...
making set 2 ...
$ ls
box.raw  coord.raw  energy.raw  force.raw  set.000  set.001  set.002  type.raw  virial.raw
```

It generates three sets `set.000`, `set.001` and `set.002`, with each set containing 2000 frames in the Numpy format.

4.3 Prepare data with dpdata

One can use a convenient tool `dpdata` to convert data directly from the output of first principle packages to the DeePMD-kit format.

To install one can execute

```
pip install dpdata
```

An example of converting data VASP data in `OUTCAR` format to DeePMD-kit data can be found at

```
$deepmd_source_dir/examples/data_conv
```

Switch to that directory, then one can convert data by using the following python script

```
import dpdata

dsys = dpdata.LabeledSystem("OUTCAR")
dsys.to("deepmd/npy", "deepmd_data", set_size=dsys.get_nframes())
```

`get_nframes()` method gets the number of frames in the `OUTCAR`, and the argument `set_size` enforces that the set size is equal to the number of frames in the system, viz. only one `set` is created in the `system`.

The data in DeePMD-kit format is stored in the folder `deepmd_data`.

A list of all supported data format and more nice features of `dpdata` can be found on the official website.

CHAPTER

FIVE

MODEL

5.1 Overall

5.1.1 Theory

A Deep Potential (DP) model, denoted by $\mathcal{M}$, can be generally represented as

$$\boldsymbol{y}_i = \mathcal{M}(\boldsymbol{x}_i, \{\boldsymbol{x}_j\}_{j\in n(i)}; \boldsymbol{\theta}) = \mathcal{F}\big(\mathcal{D}(\boldsymbol{x}_i, \{\boldsymbol{x}_j\}_{j\in n(i)}; \boldsymbol{\theta}_d); \boldsymbol{\theta}_f\big),$$

where $\boldsymbol{y}_i$ is the fitting properties, $\mathcal{F}$ is the fitting network, $\mathcal{D}$ is the descriptor. $\boldsymbol{x} = (\boldsymbol{r}_i, \alpha_i)$, with $\boldsymbol{r}_i$ being the Cartesian coordinates and α_i being the chemical species, denotes the degrees of freedom of the atom i.

The indices of the neighboring atoms (i.e. atoms within a certain cutoff radius) of atom i are given by the notation $n(i)$. Note that the Cartesian coordinates can be either under the periodic boundary condition (PBC) or in vacuum (under the open boundary condition). The network parameters are denoted by $\boldsymbol{\theta} = \{\boldsymbol{\theta}_d, \boldsymbol{\theta}_f\}$, where $\boldsymbol{\theta}_d$ and $\boldsymbol{\theta}_f$ yield the network parameters of the descriptor (if any) and those of the fitting network, respectively. From the above equation, one may compute the global property of the system by

$$\boldsymbol{y} = \sum_{i=1}^{N} \boldsymbol{y}_i,$$

where N is the number of atoms in a frame. For example, if y_i represents the potential energy contribution of atom i, then y gives the total potential energy of the frame.[1]

5.1.2 Instructions

A model has two parts, a descriptor that maps atomic configuration to a set of symmetry invariant features, and a fitting net that takes descriptor as input and predicts the atomic contribution to the target physical property. It's defined in the model section of the `input.json`, for example,

```
"model": {
    "type_map":         ["O", "H"],
    "descriptor" :{
        "...": "..."
    },
```

(continues on next page)

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

(continued from previous page)

```
        "fitting_net" : {
            "...": "..."
        }
    }
```

The two subsections, descriptor and fitting_net, define the descriptor and the fitting net, respectively.

The type_map is optional, which provides the element names (but not necessarily same as the actual name of the element) of the corresponding atom types. A water model, as in this example, has two kinds of atoms. The atom types are internally recorded as integers, e.g., `0` for oxygen and `1` for hydrogen here. A mapping from the atom type to their names is provided by type_map.

DeePMD-kit implements the following descriptors:

1. *se_e2_a*: DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.
2. *se_e2_r*: DeepPot-SE constructed from radial information of atomic configurations. The embedding takes the distance between atoms as input.
3. *se_e3*: DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes angles between two neighboring atoms as input.
4. *se_a_mask*: DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The input frames in one system can have a varied number of atoms. Input particles are padded with virtual particles of the same length.
5. `loc_frame`: Defines a local frame at each atom and compute the descriptor as local coordinates under this frame.
6. *hybrid*: Concate a list of descriptors to form a new descriptor.

The fitting of the following physical properties is supported

1. *ener*: Fit the energy of the system. The force (derivative with atom positions) and the virial (derivative with the box tensor) can also be trained.
2. *dipole*: The dipole moment.
3. *polar*: The polarizability.

5.2 Descriptor `"se_e2_a"`

Note: Supported backends: TensorFlow, PyTorch, DP

The notation of `se_e2_a` is short for the Deep Potential Smooth Edition (DeepPot-SE) constructed from all information (both angular and radial) of atomic configurations. The `e2` stands for the embedding with two-atoms information. This descriptor was described in detail in the DeepPot-SE paper.

Note that it is sometimes called a "two-atom embedding descriptor" which means the input of the embedding net is atomic distances. The descriptor does encode multi-body information (both angular and radial information of neighboring atoms).

5.2.1 Theory

The two-body embedding smooth edition of the DP descriptor $\mathcal{D}^i \in \mathbb{R}^{M \times M_<}$, is usually named DeepPot-SE descriptor. It is noted that the descriptor is a multi-body representation of the local environment of the atom i. We call it two-body embedding because the embedding network takes only the distance between atoms i and j (see below), but it is not implied that the descriptor takes only the pairwise information between i and its neighbors. The descriptor, using full information, is given by

$$\mathcal{D}^i = \frac{1}{N_c^2}(\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}^i_<,$$

where N_c is the expected maximum number of neighboring atoms, which is the same constant for all atoms over all frames. A matrix with a dimension of N_c will be padded if the number of neighboring atoms is less than N_c. $\mathcal{R}^i \in \mathbb{R}^{N_c \times 4}$ is the coordinate matrix, and each row of $\mathcal{R}^i$ can be constructed as

$$(\mathcal{R}^i)_j = \{ \ s(r_{ij}) \quad \frac{s(r_{ij})x_{ij}}{r_{ij}} \quad \frac{s(r_{ij})y_{ij}}{r_{ij}} \quad \frac{s(r_{ij})z_{ij}}{r_{ij}} \ \},$$

where $\boldsymbol{r}_{ij} = \boldsymbol{r}_j - \boldsymbol{r}_i = (x_{ij}, y_{ij}, z_{ij})$ is the relative coordinate and $r_{ij} = \|\boldsymbol{r}_{ij}\|$ is its norm. The switching function $s(r)$ is defined as

$$s(r) = \begin{cases} \frac{1}{r}, & r r_s, \\ \frac{1}{r}\left[x^3(-6x^2 + 15x - 10) + 1\right], & r_s \le r r_c, \\ 0, & r \ge r_c, \end{cases}$$

where $x = \frac{r - r_s}{r_c - r_s}$ switches from 1 at r_s to 0 at the cutoff radius r_c. The switching function $s(r)$ is smooth in the sense that the second-order derivative is continuous.

Each row of the embedding matrix $\mathcal{G}^i \in \mathbb{R}^{N_c \times M}$ consists of M nodes from the output layer of an NN function $\mathcal{N}_g$ of $s(r_{ij})$:

$$(\mathcal{G}^i)_j = \mathcal{N}_{e,2}(s(r_{ij})),$$

where the subscript $e, 2$ is used to distinguish the NN from other NNs used in the DP model. In the above equation, the network parameters are not explicitly written. $\mathcal{G}^i_< \in \mathbb{R}^{N_c \times M_<}$ only takes first $M_<$ columns of $\mathcal{G}^i$ to reduce the size of $\mathcal{D}^i$. r_s, r_c, M and $M_<$ are hyperparameters provided by the user. The DeepPot-SE is continuous up to the second-order derivative in its domain.[1]

5.2.2 Instructions

In this example, we will train a DeepPot-SE model for a water system. A complete training input script of this example can be found in the directory.

```
$deepmd_source_dir/examples/water/se_e2_a/input.json
```

With the training input script, data are also provided in the example directory. One may train the model with the DeePMD-kit from the directory.

The construction of the descriptor is given by section descriptor. An example of the descriptor is provided as follows

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

```
    "descriptor" :{
        "type":                "se_e2_a",
        "rcut_smth":         0.50,
        "rcut":                6.00,
        "sel":                [46, 92],
        "neuron":                [25, 50, 100],
        "type_one_side":       true,
        "axis_neuron":        16,
        "resnet_dt":         false,
        "seed":                1
    }
```

- The type of the descriptor is set to `"se_e2_a"`.
- rcut is the cut-off radius for neighbor searching, and the rcut_smth gives where the smoothing starts.
- sel gives the maximum possible number of neighbors in the cut-off radius. It is a list, the length of which is the same as the number of atom types in the system, and `sel[i]` denotes the maximum possible number of neighbors with type `i`.
- The neuron specifies the size of the embedding net. From left to right the members denote the sizes of each hidden layer from the input end to the output end, respectively. If the outer layer is twice the size of the inner layer, then the inner layer is copied and concatenated, then a ResNet architecture is built between them.
- If the option type_one_side is set to `true`, the embedding network parameters vary by types of neighbor atoms only, so there will be N_{types} sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be N_{types}^2 sets of embedding network parameters.
- The axis_neuron specifies the size of the submatrix of the embedding matrix, the axis matrix as explained in the DeepPot-SE paper
- If the option resnet_dt is set to `true`, then a timestep is used in the ResNet.
- seed gives the random seed that is used to generate random numbers when initializing the model parameters.

5.3 Descriptor `"se_e2_r"`

Note: Supported backends: TensorFlow, PyTorch, DP

The notation of `se_e2_r` is short for the Deep Potential Smooth Edition (DeepPot-SE) constructed from the radial information of atomic configurations. The `e2` stands for the embedding with two-atom information.

5.3.1 Theory

The descriptor, using either radial-only information, is given by

$$\mathcal{D}^i = \frac{1}{N_c} \sum_j (\mathcal{G}^i)_{jk},$$

where N_c is the expected maximum number of neighboring atoms, which is the same constant for all atoms over all frames. A matrix with a dimension of N_c will be padded if the number of neighboring atoms is less than N_c.

Each row of the embedding matrix $\mathcal{G}^i \in \mathbb{R}^{N_c \times M}$ consists of M nodes from the output layer of an NN function $\mathcal{N}_g$ of $s(r_{ij})$:

$$(\mathcal{G}^i)_j = \mathcal{N}_{e,2}(s(r_{ij})),$$

where $\boldsymbol{r}_{ij} = \boldsymbol{r}_j - \boldsymbol{r}_i = (x_{ij}, y_{ij}, z_{ij})$ is the relative coordinate and $r_{ij} = \|\boldsymbol{r}_{ij}\|$ is its norm. The switching function $s(r)$ is defined as

$$s(r) = \begin{cases} \frac{1}{r}, & r r_s, \\ \frac{1}{r}\left[x^3(-6x^2 + 15x - 10) + 1\right], & r_s \leq r r_c, \\ 0, & r \geq r_c, \end{cases}$$

where $x = \frac{r - r_s}{r_c - r_s}$ switches from 1 at r_s to 0 at the cutoff radius r_c. The switching function $s(r)$ is smooth in the sense that the second-order derivative is continuous.

In the above equations, the network parameters are not explicitly written. r_s, r_c and M are hyperparameters provided by the user. The DeepPot-SE is continuous up to the second-order derivative in its domain.[1]

5.3.2 Instructions

A complete training input script of this example can be found in the directory

```
$deepmd_source_dir/examples/water/se_e2_r/input.json
```

The training input script is very similar to that of *se_e2_a*. The only difference lies in the descriptor section

```
    "descriptor": {
        "type":                "se_e2_r",
        "sel":                 [46, 92],
        "rcut_smth":           0.50,
        "rcut":                6.00,
        "neuron":                [5, 10, 20],
    "type_one_side": true,
        "resnet_dt":           false,
        "seed":                1,
        "_comment": " that's all"
    },
```

The type of the descriptor is set by the key type.

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

5.4 Descriptor "se_e3"

Note: Supported backends: TensorFlow

The notation of `se_e3` is short for the Deep Potential Smooth Edition (DeepPot-SE) constructed from all information (both angular and radial) of atomic configurations. The embedding takes bond angles between a central atom and its two neighboring atoms as input (denoted by `e3`).

5.4.1 Theory

The three-body embedding DeepPot-SE descriptor incorporates bond-angle information, making the model more accurate. The descriptor $\mathcal{D}^i$ can be represented as

$$\mathcal{D}^i = \frac{1}{N_c^2}(\mathcal{R}^i(\mathcal{R}^i)^T) : \mathcal{G}^i,$$

where N_c is the expected maximum number of neighboring atoms, which is the same constant for all atoms over all frames. $\mathcal{R}^i$ is constructed as

$$(\mathcal{R}^i)_j = \{ \ s(r_{ij}) \quad \frac{s(r_{ij})x_{ij}}{r_{ij}} \quad \frac{s(r_{ij})y_{ij}}{r_{ij}} \quad \frac{s(r_{ij})z_{ij}}{r_{ij}} \ \},$$

Currently, only the full information case of $\mathcal{R}^i$ is supported by the three-body embedding. Each element of $\mathcal{G}^i \in \mathbb{R}^{N_c \times N_c \times M}$ comes from M nodes from the output layer of an NN $\mathcal{N}_{e,3}$ function:

$$(\mathcal{G}^i)_{jk} = \mathcal{N}_{e,3}((\theta_i)_{jk}),$$

where $(\theta_i)_{jk} = (\mathcal{R}^i)_{j,2,3,4} \cdot (\mathcal{R}^i)_{k,2,3,4}$ considers the angle form of two neighbours (j and k). The notation : in the equation indicates the contraction between matrix $\mathcal{R}^i(\mathcal{R}^i)^T$ and the first two dimensions of tensor $\mathcal{G}^i$.[1]

5.4.2 Instructions

A complete training input script of this example can be found in the directory

```
$deepmd_source_dir/examples/water/se_e3/input.json
```

The training input script is very similar to that of *se_e2_a*. The only difference lies in the `descriptor <model/descriptor>` section

```
        "descriptor": {
            "type":                "se_e3",
            "sel":                [40, 80],
            "rcut_smth":        0.50,
            "rcut":                6.00,
            "neuron":                [2, 4, 8],
            "resnet_dt":        false,
```

(continues on next page)

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

(continued from previous page)

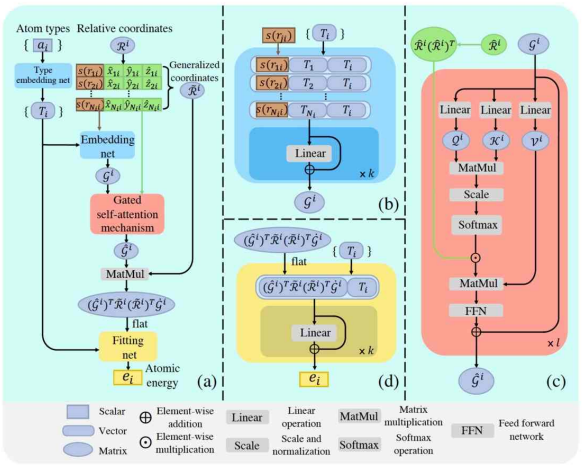
```
        "seed":                 1,
        "_comment":                 " that's all"
    },
```

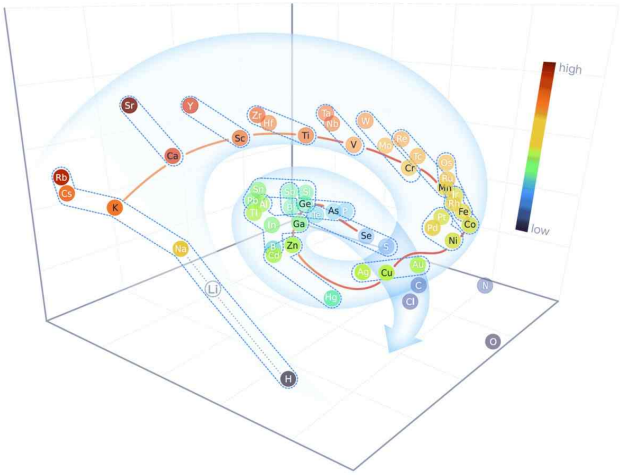
The type of the descriptor is set by the key type.

5.5 Descriptor "se_atten"

Note: Supported backends: TensorFlow, PyTorch

5.5.1 DPA-1: Pretraining of Attention-based Deep Potential Model for Molecular Simulation





Here we propose DPA-1, a Deep Potential model with a novel attention mechanism, which is highly effective for representing the conformation and chemical spaces of atomic systems and learning the PES.

See this paper for more information. DPA-1 is implemented as a new descriptor `"se_atten"` for model training, which can be used after simply editing the input.json.

5.5.2 Theory

Attention-based descriptor $\mathcal{D}^i \in \mathbb{R}^{M \times M_<}$, which is proposed in pretrainable DPA-1 model, is given by

$$\mathcal{D}^i = \frac{1}{N_c^2}(\hat{\mathcal{G}}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \hat{\mathcal{G}}^i_<,$$

where $\hat{\mathcal{G}}^i$ represents the embedding matrix $\mathcal{G}^i$ after additional self-attention mechanism and $\mathcal{R}^i$ is defined by the full case in the *se_e2_a*. Note that we obtain $\mathcal{G}^i$ using the type embedding method by default in this descriptor.

To perform the self-attention mechanism, the queries $\mathcal{Q}^{i,l} \in \mathbb{R}^{N_c \times d_k}$, keys $\mathcal{K}^{i,l} \in \mathbb{R}^{N_c \times d_k}$, and values $\mathcal{V}^{i,l} \in \mathbb{R}^{N_c \times d_v}$ are first obtained:

$$\left(\mathcal{Q}^{i,l}\right)_j = Q_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$
$$\left(\mathcal{K}^{i,l}\right)_j = K_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$
$$\left(\mathcal{V}^{i,l}\right)_j = V_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$

where Q_l, K_l, V_l represent three trainable linear transformations that output the queries and keys of dimension d_k and values of dimension d_v, and l is the index of the attention layer. The input embedding matrix to the attention layers, denoted by $\mathcal{G}^{i,0}$, is chosen as the two-body embedding matrix.

Then the scaled dot-product attention method is adopted:

$$A(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{V}^{i,l}, \mathcal{R}^{i,l}) = \varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right)\mathcal{V}^{i,l},$$

where $\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right) \in \mathbb{R}^{N_c \times N_c}$ is attention weights. In the original attention method, one typically has $\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}\right) = \mathrm{softmax}\left(\frac{\mathcal{Q}^{i,l}(\mathcal{K}^{i,l})^T}{\sqrt{d_k}}\right)$, with $\sqrt{d_k}$ being the normalization temperature. This is slightly modified to incorporate the angular information:

$$\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right) = \mathrm{softmax}\left(\frac{\mathcal{Q}^{i,l}(\mathcal{K}^{i,l})^T}{\sqrt{d_k}}\right) \odot \hat{\mathcal{R}}^i(\hat{\mathcal{R}}^i)^T,$$

where $\hat{\mathcal{R}}^i \in \mathbb{R}^{N_c \times 3}$ denotes normalized relative coordinates , $\hat{\mathcal{R}}^i_j = \frac{\boldsymbol{r}_{ij}}{\|\boldsymbol{r}_{ij}\|}$ and $\odot$ means element-wise multiplication.

Then layer normalization is added in a residual way to finally obtain the self-attention local embedding matrix $\hat{\mathcal{G}}^i = \mathcal{G}^{i,L_a}$ after L_a attention layers:[1]

$$\mathcal{G}^{i,l} = \mathcal{G}^{i,l-1} + \mathrm{LayerNorm}(A(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{V}^{i,l}, \mathcal{R}^{i,l})).$$

5.5.3 Introduction to new features of DPA-1

Next, we will list the detailed settings in input.json and the data format, especially for large systems with dozens of elements. An example of DPA-1 input can be found here.

Descriptor `"se_atten"`

The notation of `se_atten` is short for the smooth edition of Deep Potential with an attention mechanism. This descriptor was described in detail in the DPA-1 paper and the images above.

In this example, we will train a DPA-1 model for a water system. A complete training input script of this example can be found in the directory:

```
$deepmd_source_dir/examples/water/se_atten/input.json
```

With the training input script, data are also provided in the example directory. One may train the model with the DeePMD-kit from the directory.

An example of the DPA-1 descriptor is provided as follows

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

TensorFlow

```
"descriptor" :{
  "type":                "se_atten",
  "rcut_smth":        0.50,
  "rcut":                6.00,
  "sel":                 120,
  "neuron":                 [25, 50, 100],
  "axis_neuron":         16,
  "resnet_dt":        false,
  "attn":         128,
  "attn_layer":         2,
  "attn_mask":        false,
  "attn_dotr":        true,
  "seed":         1
}
```

- The type of the descriptor is set to `"se_atten"`, which will use DPA-1 structures.
- rcut is the cut-off radius for neighbor searching, and the rcut_smth gives where the smoothing starts.
- sel gives the maximum possible number of neighbors in the cut-off radius. It is an int. Note that this number highly affects the efficiency of training, which we usually use less than 200. (We use 120 for training 56 elements in OC2M dataset)
- The neuron specifies the size of the embedding net. From left to right the members denote the sizes of each hidden layer from the input end to the output end, respectively. If the outer layer is twice the size of the inner layer, then the inner layer is copied and concatenated, then a ResNet architecture is built between them.
- The axis_neuron specifies the size of the submatrix of the embedding matrix, the axis matrix as explained in the DeepPot-SE paper
- If the option resnet_dt is set to `true`, then a timestep is used in the ResNet.
- seed gives the random seed that is used to generate random numbers when initializing the model parameters.
- attn sets the length of a hidden vector during scale-dot attention computation.
- attn_layer sets the number of layers in attention mechanism.
- attn_mask determines whether to mask the diagonal in the attention weights and False is recommended.
- attn_dotr determines whether to dot the relative coordinates on the attention weights as a gated scheme, True is recommended.

PyTorch

```
"descriptor" :{
  "type":                "dpa1",
  "rcut_smth":        0.50,
  "rcut":                6.00,
  "sel":                 120,
  "neuron":                 [25, 50, 100],
  "tebd_dim": 8,
```

(continues on next page)

(continued from previous page)

```
        "axis_neuron":        16,
        "attn":        128,
        "attn_layer":        2,
        "attn_mask": false,
        "attn_dotr": true,
      }
```

- The type of the descriptor is set to `"dpa1"`, which will use DPA-1 structures.
- rcut is the cut-off radius for neighbor searching, and the rcut_smth gives where the smoothing starts.
- sel gives the maximum possible number of neighbors in the cut-off radius. It is an int. Note that this number highly affects the efficiency of training, which we usually use less than 200. (We use 120 for training 56 elements in OC2M dataset)
- The neuron specifies the size of the embedding net. From left to right the members denote the sizes of each hidden layer from the input end to the output end, respectively. If the outer layer is twice the size of the inner layer, then the inner layer is copied and concatenated, then a ResNet architecture is built between them.
- The tebd_dim specifies the dimension of the type embedding.
- The axis_neuron specifies the size of the submatrix of the embedding matrix, the axis matrix as explained in the DeepPot-SE paper
- attn sets the length of a hidden vector during scale-dot attention computation.
- attn_layer sets the number of layers in attention mechanism.
- attn_mask determines whether to mask the diagonal in the attention weights and False is recommended.
- attn_dotr determines whether to dot the relative coordinates on the attention weights as a gated scheme, True is recommended.

Descriptor `"se_atten_v2"`

We highly recommend using the version 2.0 of the attention-based descriptor `"se_atten_v2"`, which is inherited from `"se_atten"` but with the following parameter modifications:

```
    "tebd_input_mode": "strip",
    "smooth_type_embedding": true,
    "set_davg_zero": false
```

Practical evidence demonstrates that `"se_atten_v2"` offers better and more stable performance compared to `"se_atten"`.

Notice: Model compression for the `se_atten_v2` descriptor is exclusively designed for models with the training parameter attn_layer set to 0.

Fitting `"ener"`

DPA-1 only supports `"ener"` fitting type, and you can refer here for detailed information.

Type embedding

DPA-1 only supports models with type embeddings. And the default setting is as follows:

```
"type_embedding":{
            "neuron":           [8],
            "resnet_dt":        false,
            "seed":             1
        }
```

You can add these settings in input.json if you want to change the default ones, see here for detailed information.

Type map

For training large systems, especially those with dozens of elements, the type determines the element index of training data:

```
"type_map": [
   "Mg",
   "Al",
   "Cu"
  ]
```

which should include all the elements in the dataset you want to train on.

5.5.4 Data format

DPA-1 supports the standard data format, which is detailed in data-conv.md and system.md. Note that in this format, only those frames with the same fingerprint (i.e. the number of atoms of different elements) can be put together as a unified system. This may lead to sparse frame numbers in those rare systems.

An ideal way is to put systems with the same total number of atoms together, which is the way we trained DPA-1 on OC2M. This system format, which is called `mixed_type`, is proper to put frame-sparse systems together and is slightly different from the standard one. Take an example, a `mixed_type` may contain the following files:

```
type.raw
type_map.raw
set.*/box.npy
set.*/coord.npy
set.*/energy.npy
set.*/force.npy
set.*/real_atom_types.npy
```

This system contains `Nframes` frames with the same atom number `Natoms`, the total number of element types contained in all frames is `Ntypes`. Most files are the same as those in standard formats, here we only list the distinct ones:

ID	Property	File	Required/O	Shape	Description
/	Atom type indexes (place holder)	type.raw	Required	Natoms	All zeros to fake the type input
type_	Atom type names	type_map	Required	Ntypes	Atom names that map to atom type contained in all the frames, which is unnecessart to be contained in the periodic table
type	Atom type indexes of each frame	real_atom	Required	Nframes * Natoms	Integers that describe atom types in each frame, corresponding to indexes in type_map. `-1` means virtual atoms.

With these edited files, one can put together frames with the same `Natoms`, instead of the same formula (like `H2O`). Note that this `mixed_type` format only supports `se_atten` descriptor.

To put frames with different `Natoms` into the same system, one can pad systems by adding virtual atoms whose type is `-1`. Virtual atoms do not contribute to any fitting property, so the atomic property of virtual atoms (e.g. forces) should be given zero.

The API to generate or transfer to `mixed_type` format is available on dpdata for a more convenient experience.

5.5.5 Training example

Here we upload the AlMgCu example shown in the paper, you can download it here: Baidu disk; Google disk.

5.6 Descriptor DPA-2

Note: Supported backends: PyTorch

5.7 Descriptor "hybrid"

Note: Supported backends: TensorFlow, PyTorch, DP

This descriptor hybridizes multiple descriptors to form a new descriptor. For example, we have a list of descriptors denoted by $\mathcal{D}_1$, $\mathcal{D}_2$, ..., $\mathcal{D}_N$, the hybrid descriptor this the concatenation of the list, i.e. $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_N)$.

5.7.1 Theory

A hybrid descriptor $\mathcal{D}^i_{\text{hyb}}$ concatenates multiple kinds of descriptors into one descriptor:

$$\mathcal{D}^i_{\text{hyb}} = \{ \quad \mathcal{D}^i_1 \quad \mathcal{D}^i_2 \quad \cdots \quad \mathcal{D}^i_n \quad \}.$$

The list of descriptors can be different types or the same descriptors with different parameters. This way, one can set the different cutoff radii for different descriptors.[1]

5.7.2 Instructions

To use the descriptor in DeePMD-kit, one firstly set the type to hybrid, then provide the definitions of the descriptors by the items in the `list`,

```
        "descriptor" :{
            "type": "hybrid",
            "list" : [
                {
                    "type" : "se_e2_a",
                    ...
                },
                {
                    "type" : "se_e2_r",
                    ...
                }
            ]
        },
```

A complete training input script of this example can be found in the directory

```
$deepmd_source_dir/examples/water/hybrid/input.json
```

5.8 Determine `sel`

All descriptors require to set `sel`, which means the expected maximum number of type-i neighbors of an atom. DeePMD-kit will allocate memory according to `sel`.

`sel` should not be too large or too small. If `sel` is too large, the computing will become much slower and cost more memory. If `sel` is not enough, the energy will be not conserved, making the accuracy of the model worse.

To determine a proper `sel`, one can calculate the neighbor stat of the training data before training:

```
dp neighbor-stat -s data -r 6.0 -t O H
```

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

where `data` is the directory of data, `6.0` is the cutoff radius, and `O` and `H` is the type map. The program will give the `max_nbor_size`. For example, `max_nbor_size` of the water example is `[38, 72]`, meaning an atom may have 38 O neighbors and 72 H neighbors in the training data.

The `sel` should be set to a higher value than that of the training data, considering there may be some extreme geometries during MD simulations. As a result, we set `sel` to `[46, 92]` in the water example.

5.9 Fit energy

Note: Supported backends: TensorFlow, PyTorch, DP

In this section, we will take `$deepmd_source_dir/examples/water/se_e2_a/input.json` as an example of the input file.

5.9.1 Theory

In the DP model, we let the fitting network $\mathcal{F}_0$ maps the descriptor $\mathcal{D}^i$ to a scalar, where the subscript 0 means that the output is a zero-order tensor (i.e. scalar). The model can then be used to predict the total potential energy of the system by

$$E = \sum_i E_i = \sum_i \mathcal{F}_0(\mathcal{D}^i),$$

where the output of the fitting network is treated as the atomic potential energy contribution, i.e. E_i. The output scalar can also be treated as other scalar properties defined on an atom, for example, the partial charge of atom i.

In some cases, atomic-specific or frame-specific parameters, such as electron temperature, may be treated as extra input to the fitting network. We denote the atomic and frame-specific parameters by $\boldsymbol{P}^i \in \mathbb{R}^{N_p}$ (with N_p being the dimension) and $\boldsymbol{Q} \in \mathbb{R}^{N_q}$ (with N_q being the dimension), respectively.

$$E_i = \mathcal{F}_0(\{\mathcal{D}^i, \boldsymbol{P}^i, \boldsymbol{Q}\}).$$

The atomic force $\boldsymbol{F}_i$ and the virial tensor $\boldsymbol{\Xi} = (\Xi_{\alpha\beta})$ (if PBC is applied) can be derived from the potential energy E:

$$F_{i,\alpha} = -\frac{\partial E}{\partial r_{i,\alpha}},$$

$$\Xi_{\alpha\beta} = -\sum_\gamma \frac{\partial E}{\partial h_{\gamma\alpha}} h_{\gamma\beta},$$

where $r_{i,\alpha}$ and $F_{i,\alpha}$ denotes the α-th component of the coordinate and force of atom i. $h_{\alpha\beta}$ is the β-th component of the α-th basis vector of the simulation region.

The properties η of the energy loss function could be energy E, force $\boldsymbol{F}$, virial $\boldsymbol{\Xi}$, relative energy ΔE, or any combination among them, and the loss functions of them are

$$L_E(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{N}(E(\boldsymbol{x};\boldsymbol{\theta}) - E^*)^2,$$

$$L_F(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{3N}\sum_{k=1}^{N}\sum_{\alpha=1}^{3}(F_{k,\alpha}(\boldsymbol{x};\boldsymbol{\theta}) - F^*_{k,\alpha})^2,$$

$$L_\Xi(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{9N}\sum_{\alpha,\beta=1}^{3}(\Xi_{\alpha\beta}(\boldsymbol{x};\boldsymbol{\theta}) - \Xi^*_{\alpha\beta})^2,$$

$$L_{\Delta E}(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{N}(\Delta E(\boldsymbol{x};\boldsymbol{\theta}) - \Delta E^*)^2,$$

where $F_{k,\alpha}$ is the α-th component of the force on atom k, and the superscript $*$ indicates the label of the property that should be provided in advance. Using N ensures that each loss of fitting property is averaged over atomic contributions before they contribute to the total loss by weight.

If part of atoms is more important than others, for example, certain atoms play an essential role when calculating free energy profiles or kinetic isotope effects, the MSE of atomic forces with prefactors q_k can also be used as the loss function:

$$L^p_F(\mathrm{x};\boldsymbol{\theta}) = \frac{1}{3N}\sum_{k=1}^{N}\sum_{\alpha} q_k(F_{k,\alpha}(\mathrm{x};\boldsymbol{\theta}) - F^*_{k,\alpha})^2.$$

The atomic forces with larger prefactors will be fitted more accurately than those in other atoms.

If some forces are quite large, for example, forces can be greater than 60 eV/Å in high-temperature reactive simulations, one may also prefer the force loss is relative to the magnitude:

$$L_F^r(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{3N}\sum_{k=1}^{N}\sum_{\alpha}\left(\frac{F_{k,\alpha}(\boldsymbol{x};\boldsymbol{\theta}) - F_{k,\alpha}^*}{|\boldsymbol{F}_k^*|+\nu}\right)^2.$$

where ν is a small constant used to protect an atom where the magnitude of $\boldsymbol{F}_k^*$ is small from having a large L_F^r. Benefiting from the relative force loss, small forces can be fitted more accurately.[1]

5.9.2 The fitting network

The construction of the fitting net is given by section fitting_net

```
"fitting_net" : {
    "neuron":               [240, 240, 240],
    "resnet_dt":        true,
    "seed":                 1
},
```

- neuron specifies the size of the fitting net. If two neighboring layers are of the same size, then a ResNet architecture is built between them.
- If the option resnet_dt is set to `true`, then a timestep is used in the ResNet.
- seed gives the random seed that is used to generate random numbers when initializing the model parameters.

5.9.3 Loss

The loss function L for training energy is given by

$$L = p_e L_e + p_f L_f + p_v L_v$$

where L_e, L_f, and L_v denote the loss in energy, forces and virials, respectively. p_e, p_f, and p_v give the prefactors of the energy, force and virial losses. The prefectors may not be a constant, rather it changes linearly with the learning rate. Taking the force prefactor for example, at training step t, it is given by

$$p_f(t) = p_f^0\frac{\alpha(t)}{\alpha(0)} + p_f^\infty(1-\frac{\alpha(t)}{\alpha(0)})$$

where $\alpha(t)$ denotes the learning rate at step t. p_f^0 and p_f^∞ specifies the p_f at the start of the training and the limit of $t \to \infty$ (set by start_pref_f and limit_pref_f, respectively), i.e.

$$pref_f(t) = start_pref_f * (lr(t)/start_lr) + limit_pref_f * (1 - lr(t)/start_lr)$$

The loss section in the `input.json` is

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

```
"loss" : {
    "start_pref_e":         0.02,
    "limit_pref_e":         1,
    "start_pref_f":         1000,
    "limit_pref_f":         1,
    "start_pref_v":         0,
    "limit_pref_v":         0
}
```

The options start_pref_e, limit_pref_e, start_pref_f, limit_pref_f, start_pref_v and limit_pref_v determine the start and limit prefactors of energy, force and virial, respectively.

If one does not want to train with virial, then he/she may set the virial prefactors start_pref_v and limit_pref_v to 0.

5.10 Fit spin energy

Note: Supported backends: TensorFlow, PyTorch, DP

To train a model that takes additional spin information as input, you only need to modify the following sections to define the spin-specific settings, keeping other sections the same as the normal energy model's input script.

Warning: Note that when adding spin into the model, there will be some implicit modifications automatically done by the program:

- In the TensorFlow backend, the `se_e2_a` descriptor will treat those atom types with spin as new (virtual) types, and duplicate their corresponding selected numbers of neighbors (sel) from their real atom types.

- In the PyTorch backend, if spin settings are added, all the types (with or without spin) will have their virtual types. The `se_e2_a` descriptor will thus double the sel list, while in other descriptors with mixed types (such as `dpa1` or `dpa2`), the sel number will not be changed for clarity. If you are using descriptors with mixed types, to achieve better performance, you should manually extend your sel number (maybe double) depending on the balance between performance and efficiency.

5.10.1 Spin

The spin settings are given by the spin section, which sets the magnetism for each type of atoms as described in the following sections.

Note: Note that the construction of spin settings is different between TensorFlow and PyTorch/DP.

Spin settings in TensorFlow

The implementation in TensorFlow only supports `se_e2_a` descriptor. See examples in `$deepmd_source_dir/examples/spin/se_e2_a/input_tf.json`, the spin section is defined as the following:

```
"spin" : {
    "use_spin":         [true, false],
    "virtual_len":      [0.4],
    "spin_norm":        [1.2737],
},
```

- use_spin is a list of boolean values indicating whether to use atomic spin for each atom type. True for spin and False for not. The index of this option matches option `type_map <model/type_map>`.
- virtual_len specifies the distance between virtual atom and the belonging real atom.
- spin_norm gives the magnitude of the magnetic moment for each magnatic atom.

Spin settings in PyTorch/DP

In PyTorch/DP, the spin implementation is more flexible and so far supports the following descriptors:

- `se_e2_a`
- `dpa1(se_atten)`
- `dpa2`

See `se_e2_a` examples in `$deepmd_source_dir/examples/spin/se_e2_a/input_torch.json`, the spin section is defined as the following with a much more clear interface:

```
"spin": {
  "use_spin": [true, false],
  "virtual_scale": [0.3140]
},
```

- use_spin is a list of boolean values indicating whether to use atomic spin for each atom type, or a list of type indexes that use atomic spin. The index of this option matches option `type_map <model/type_map>`.

- virtual_len defines the scaling factor to determine the virtual distance between a virtual atom representing spin and its corresponding real atom for each atom type with spin. This factor is defined as the virtual distance divided by the magnitude of atomic spin for each atom type with spin. The virtual coordinate is defined as the real coordinate plus spin * virtual_scale. List of float values with shape of `ntypes` or `ntypes_spin` or one single float value for all types, only used when use_spin is True for each atom type.

5.10.2 Spin Loss

The spin loss function L for training energy is given by

$$L = p_e L_e + p_{fr} L_{fr} + p_{fm} L_{fm} + p_v L_v$$

where L_e, L_{fr}, L_{fm} and L_v denote the loss in energy, atomic force, magnatic force and virial, respectively. p_e, p_{fr}, p_{fm} and p_v give the prefactors of the energy, atomic force, magnatic force and virial losses.

The prefectors may not be a constant, rather it changes linearly with the learning rate. Taking the atomic force prefactor for example, at training step t, it is given by

$$p_{fr}(t) = p_{fr}^0 \frac{\alpha(t)}{\alpha(0)} + p_{fr}^\infty (1 - \frac{\alpha(t)}{\alpha(0)})$$

where $\alpha(t)$ denotes the learning rate at step t. p_{fr}^0 and p_{fr}^∞ specifies the p_f at the start of the training and at the limit of $t \to \infty$ (set by start_pref_fr and limit_pref_f, respectively), i.e.

$$pref_f r(t) = start_p ref_f r * (lr(t)/start_l r) + limit_p ref_f r * (1 - lr(t)/start_l r)$$

The loss section in the `input.json` is

```
"loss" :{
    "type":                    "ener_spin",
    "start_pref_e":            0.02,
    "limit_pref_e":            1,
    "start_pref_fr":           1000,
"limit_pref_fr":           1.0,
    "start_pref_fm":           10000,
    "limit_pref_fm":           10.0,
    "start_pref_v":            0,
    "limit_pref_v":            0,
},
```

The options start_pref_e, limit_pref_e, start_pref_fr, limit_pref_fm, start_pref_v and limit_pref_v determine the start and limit prefactors of energy, atomic force, magnatic force and virial, respectively.

If one does not want to train with virial, then he/she may set the virial prefactors start_pref_v and limit_pref_v to 0.

5.10.3 Data preparation

(Need a documentation for data format for TensorFlow and PyTorch/DP.)

5.11 Fit `tensor` like `Dipole` and `Polarizability`

Note: Supported backends: TensorFlow

Unlike `energy`, which is a scalar, one may want to fit some high dimensional physical quantity, like `dipole` (vector) and `polarizability` (matrix, shorted as `polar`). Deep Potential has provided different APIs to do this. In this example, we will show you how to train a model to fit a water system. A complete training input script of the examples can be found in

TensorFlow

```
$deepmd_source_dir/examples/water_tensor/dipole/dipole_input.json
$deepmd_source_dir/examples/water_tensor/polar/polar_input.json
```

PyTorch

```
$deepmd_source_dir/examples/water_tensor/dipole/dipole_input_torch.json
$deepmd_source_dir/examples/water_tensor/polar/polar_input_torch.json
```

The training and validation data are also provided our examples. But note that the data provided along with the examples are of limited amount, and should not be used to train a production model.

Similar to the `input.json` used in `ener` mode, training JSON is also divided into model, learning_rate, loss and training. Most keywords remain the same as `ener` mode, and their meaning can be found here. To fit a tensor, one needs to modify model/fitting_net and loss.

5.11.1 Theory

To represent the first-order tensorial properties (i.e. vector properties), we let the fitting network, denoted by $\mathcal{F}_1$, output an M-dimensional vector; then we have the representation,

$$(T_i^{(1)})_\alpha = \frac{1}{N_c} \sum_{j=1}^{N_c} \sum_{m=1}^{M} (\mathcal{G}^i)_{jm} (\mathcal{R}^i)_{j,\alpha+1} (\mathcal{F}_1(\mathcal{D}^i))_m, \ \alpha = 1, 2, 3.$$

We let the fitting network $\mathcal{F}_2$ output an M-dimensional vector, and the second-order tensorial properties (matrix properties) are formulated as

$$(T_i^{(2)})_{\alpha\beta} = \frac{1}{N_c^2} \sum_{j=1}^{N_c} \sum_{k=1}^{N_c} \sum_{m=1}^{M} (\mathcal{G}^i)_{jm} (\mathcal{R}^i)_{j,\alpha+1} (\mathcal{R}^i)_{k,\beta+1} (\mathcal{G}^i)_{km} (\mathcal{F}_2(\mathcal{D}^i))_m, \ \alpha, \beta = 1, 2, 3,$$

where $\mathcal{G}^i$ and $\mathcal{R}^i$ can be found in *se_e2_a*. Thus, the tensor fitting network requires the descriptor to have the same or similar form as the DeepPot-SE descriptor. $\mathcal{F}_1$ and $\mathcal{F}_2$ are the neural network functions. The total tensor $\boldsymbol{T}$ (total dipole $\boldsymbol{T}^{(1)}$ or total polarizability $\boldsymbol{T}^{(2)}$) is the sum of the atomic tensor:

$$\boldsymbol{T} = \sum_i \boldsymbol{T}_i.$$

The tensorial models can be used to calculate IR spectrum and Raman spectrum.[1]

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

5.11.2 The fitting Network

The fitting_net section tells DP which fitting net to use.

TensorFlow

The JSON of `dipole` type should be provided like

```
    "fitting_net" : {
            "type": "dipole",
            "sel_type": [0],
            "neuron": [100,100,100],
            "resnet_dt": true,
            "seed": 1,
    },
```

The JSON of `polar` type should be provided like

```
    "fitting_net" : {
               "type": "polar",
            "sel_type": [0],
            "neuron": [100,100,100],
            "resnet_dt": true,
            "seed": 1,
    },
```

- `type` specifies which type of fitting net should be used. It should be either `dipole` or `polar`. Note that `global_polar` mode in version 1.x is already deprecated and is merged into `polar`. To specify whether a system is global or atomic, please see here.
- `sel_type` is a list specifying which type of atoms have the quantity you want to fit. For example, in the water system, `sel_type` is `[0]` since `0` represents atom `O`. If left unset, all types of atoms will be fitted.
- The rest arguments have the same meaning as they do in `ener` mode.

PyTorch

The JSON of `dipole` type should be provided like

```
    "atom_exclude_types": [
  1
],
    "fitting_net" : {
            "type": "dipole",
            "neuron": [100,100,100],
            "resnet_dt": true,
            "seed": 1,
    },
```

The JSON of `polar` type should be provided like

```
    "atom_exclude_types": [
  1
],
```

(continues on next page)

(continued from previous page)

```
        "fitting_net" : {
                   "type": "polar",
                "neuron": [100,100,100],
                "resnet_dt": true,
                "seed": 1,
        },
```

- `type` specifies which type of fitting net should be used. It should be either `dipole` or `polar`. Note that `global_polar` mode in version 1.x is already deprecated and is merged into `polar`. To specify whether a system is global or atomic, please see here.
- `atom_exclude_types` is a list specifying the which type of atoms have the quantity you want to set to zero. For example, in the water system, `atom_exclude_types` is `[1]` since `1` represents atom `H`.
- The rest arguments have the same meaning as they do in `ener` mode.

5.11.3 Loss

DP supports a combinational training of the global system (only a global `tensor` label, i.e. dipole or polar, is provided in a frame) and atomic system (labels for each atom included in `sel_type`/ not included in `atom_exclude_types` are provided). In a global system, each frame has just one `tensor` label. For example, when fitting `polar`, each frame will just provide a `1 x 9` vector which gives the elements of the polarizability tensor of that frame in order XX, XY, XZ, YX, YY, YZ, XZ, ZY, ZZ. By contrast, in an atomic system, each atom in `sel_type` has a `tensor` label. For example, when fitting a dipole, each frame will provide a `#sel_atom x 3` matrices, where `#sel_atom` is the number of atoms whose type are in `sel_type`.

The loss section tells DP the weight of these two kinds of loss, i.e.

```
loss = pref * global_loss + pref_atomic * atomic_loss
```

The loss section should be provided like

```
        "loss" : {
                "type":                 "tensor",
                "pref":                 1.0,
                "pref_atomic":          1.0
        },
```

- type should be written as `tensor` as a distinction from `ener` mode.
- pref and pref_atomic respectively specify the weight of global loss and atomic loss. It can not be left unset. If set to 0, the corresponding label will NOT be included in the training process.

5.11.4 Training Data Preparation

In tensor mode, the identification of the label's type (global or atomic) is derived from the file name. The global label should be named `dipole.npy/raw` or `polarizability.npy/raw`, while the atomic label should be named `atomic_dipole.npy/raw` or `atomic_polarizability.npy/raw`. If wrongly named, DP will report an error

```
ValueError: cannot reshape array of size xxx into shape (xx,xx). This error may occur when your␣
↪label mismatch it's name, i.e. you might store global tensor in `atomic_tensor.npy` or atomic␣
↪tensor in `tensor.npy`.
```

In this case, please check the file name of the label.

5.11.5 Train the Model

The training command is the same as `ener` mode, i.e.

TensorFlow

```
dp train input.json
```

PyTorch

```
dp --pt train input.json
```

The detailed loss can be found in `lcurve.out`:

```
#  step      rmse_val    rmse_trn  rmse_lc_val rmse_lc_trn rmse_gl_val rmse_gl_trn  lr
      0      8.34e+00    8.26e+00    8.34e+00    8.26e+00    0.00e+00    0.00e+00    1.0e-02
    100      3.51e-02    8.55e-02    0.00e+00    8.55e-02    4.38e-03    0.00e+00    5.0e-03
    200      4.77e-02    5.61e-02    0.00e+00    5.61e-02    5.96e-03    0.00e+00    2.5e-03
    300      5.68e-02    1.47e-02    0.00e+00    0.00e+00    7.10e-03    1.84e-03    1.3e-03
    400      3.73e-02    3.48e-02    1.99e-02    0.00e+00    2.18e-03    4.35e-03    6.3e-04
    500      2.77e-02    5.82e-02    1.08e-02    5.82e-02    2.11e-03    0.00e+00    3.2e-04
    600      2.81e-02    5.43e-02    2.01e-02    0.00e+00    1.01e-03    6.79e-03    1.6e-04
    700      2.97e-02    3.28e-02    2.03e-02    0.00e+00    1.17e-03    4.10e-03    7.9e-05
    800      2.25e-02    6.19e-02    9.05e-03    0.00e+00    1.68e-03    7.74e-03    4.0e-05
    900      3.18e-02    5.54e-02    9.93e-03    5.54e-02    2.74e-03    0.00e+00    2.0e-05
   1000      2.63e-02    5.02e-02    1.02e-02    5.02e-02    2.01e-03    0.00e+00    1.0e-05
   1100      3.27e-02    5.89e-02    2.13e-02    5.89e-02    1.43e-03    0.00e+00    5.0e-06
   1200      2.85e-02    2.42e-02    2.85e-02    0.00e+00    0.00e+00    3.02e-03    2.5e-06
   1300      3.47e-02    5.71e-02    1.07e-02    5.71e-02    3.00e-03    0.00e+00    1.3e-06
   1400      3.13e-02    5.76e-02    3.13e-02    5.76e-02    0.00e+00    0.00e+00    6.3e-07
   1500      3.34e-02    1.11e-02    2.09e-02    0.00e+00    1.57e-03    1.39e-03    3.2e-07
   1600      3.11e-02    5.64e-02    3.11e-02    5.64e-02    0.00e+00    0.00e+00    1.6e-07
   1700      2.97e-02    5.05e-02    2.97e-02    5.05e-02    0.00e+00    0.00e+00    7.9e-08
   1800      2.64e-02    7.70e-02    1.09e-02    0.00e+00    1.94e-03    9.62e-03    4.0e-08
   1900      3.28e-02    2.56e-02    3.28e-02    0.00e+00    0.00e+00    3.20e-03    2.0e-08
   2000      2.59e-02    5.71e-02    1.03e-02    5.71e-02    1.94e-03    0.00e+00    1.0e-08
```

One may notice that in each step, some of the local loss and global loss will be `0.0`. This is because our training data and validation data consist of the global system and atomic system, i.e.

```
        --training_data
                >atomic_system
                >global_system
        --validation_data
                >atomic_system
                >global_system
```

During training, at each step when the `lcurve.out` is printed, the system used for evaluating the training (validation) error may be either with only global or only atomic labels, thus the corresponding atomic or global errors are missing and are printed as zeros.

5.12 Fit electronic density of states (DOS)

Note: Supported backends: TensorFlow

Here we present an API to DeepDOS model, which can be used to fit electronic density of state (DOS) (which is a vector).

See the PRB paper for details.

In this example, we will show you how to train a model to fit a silicon system. A complete training input script of the examples can be found in

```
$deepmd_source_dir/examples/dos/input.json
```

The training and validation data are also provided our examples. But note that the data provided along with the examples are of limited amount, and should not be used to train a production model.

Similar to the `input.json` used in `ener` mode, training JSON is also divided into model, learning_rate, loss and training. Most keywords remain the same as `ener` mode, and their meaning can be found here. To fit the `dos`, one needs to modify model/fitting_net and loss.

5.12.1 The fitting Network

The fitting_net section tells DP which fitting net to use.

The JSON of `dos` type should be provided like

```
"fitting_net" : {
        "type": "dos",
        "numb_dos": 250,
        "sel_type": [0],
        "neuron": [120,120,120],
        "resnet_dt": true,
        "fparam": 0,
        "seed": 1,
},
```

- `type` specifies which type of fitting net should be used. It should be `dos`.
- `numb_dos` specifies the length of output vector (density of states), which the same as the `NEDOS` set in VASP software, this argument defines the output length of the neural network. We note that the length of `dos` provided in training set should be the same.
- The rest arguments have the same meaning as they do in `ener` mode.

5.12.2 Loss

DeepDOS supports trainings of the global system (a global `dos` label is provided in a frame) or atomic system (atomic labels `atom_dos` is provided for each atom in a frame). In a global system, each frame has just one `dos` label. For example, when fitting `dos`, each frame will just provide a `1 x numb_dos` vector which gives the total electronic density of states. By contrast, in an atomic system, each atom in has a `atom_dos` label. For example, when fitting the site-projected electronic density of states, each frame will provide a `natom x numb_dos` matrices,

The loss section tells DP the weight of these two kinds of loss, i.e.

```
loss = pref * global_loss + pref_atomic * atomic_loss
```

The loss section should be provided like

```
        "loss" : {
                "type": "dos",
                "start_pref_dos": 0.0,
                "limit_pref_dos": 0.0,
                "start_pref_cdf": 0.0,
                "limit_pref_cdf": 0.0,
                "start_pref_ados": 1.0,
                "limit_pref_ados": 1.0,
                "start_pref_acdf": 0.0,
                "limit_pref_acdf": 0.0
        },
```

- type should be written as `dos` as a distinction from `ener` mode.
- `pref_dos` and `pref_ados`, respectively specify the weight of global and atomic loss. If set to 0, the corresponding label will not be included in the training process.
- We also provides a combination training of vector and its cumulative distribution function `cdf`, which can be defined as

$$D(\epsilon) = \int_{e_{min}}^{\epsilon} g(\epsilon')d\epsilon'$$

5.12.3 Training Data Preparation

The global label should be named `dos.npy/raw`, while the atomic label should be named `atomic_dos.npy/raw`. If wrongly named, DP will report an error.

To prepare the data, we recommend shifting the DOS data by the Fermi level.

5.12.4 Train the Model

The training command is the same as `ener` mode, i.e.

```
dp train input.json
```

The detailed loss can be found in `lcurve.out`:

```
#  step      rmse_trn   rmse_ados_trn   rmse_ados_lr
      0      1.11e+00      1.11e+00    1.0e-03
    100      5.00e-02      5.00e-02    1.0e-03
```

(continues on next page)

(continued from previous page)

```
   200      4.70e-02      4.70e-02    1.0e-03
   300      6.45e-02      6.45e-02    1.0e-03
   400      3.39e-02      3.39e-02    1.0e-03
   500      4.60e-02      4.60e-02    1.0e-03
   600      3.98e-02      3.98e-02    1.0e-03
   700      9.50e-02      9.50e-02    1.0e-03
   800      5.49e-02      5.49e-02    1.0e-03
   900      5.57e-02      5.57e-02    1.0e-03
  1000      3.73e-02      3.73e-02    1.0e-03
  1100      4.33e-02      4.33e-02    1.0e-03
  1200      3.27e-02      3.27e-02    1.0e-03
  1300      3.68e-02      3.68e-02    1.0e-03
  1400      3.09e-02      3.09e-02    1.0e-03
  1500      3.42e-02      3.42e-02    1.0e-03
  1600      5.62e-02      5.62e-02    1.0e-03
  1700      6.12e-02      6.12e-02    1.0e-03
  1800      4.10e-02      4.10e-02    1.0e-03
  1900      5.30e-02      5.30e-02    1.0e-03
  2000      3.85e-02      3.85e-02    1.0e-03
```

5.12.5 Test the Model

In this earlier version, we can use `dp test` to infer the electronic density of state for given frames.

```
$DP freeze -o frozen_model.pb

$DP test -m frozen_model.pb -s ../data/111/$k -d ${output_prefix} -a -n 100
```

if `dp test -d ${output_prefix} -a` is specified, the predicted DOS and atomic DOS for each frame is output in the working directory

```
${output_prefix}.ados.out.0   ${output_prefix}.ados.out.1  ${output_prefix}.ados.out.2  ${output_
↪prefix}.ados.out.3
${output_prefix}.dos.out.0   ${output_prefix}.dos.out.1  ${output_prefix}.dos.out.2  ${output_
↪prefix}.dos.out.3
```

for `*.dos.out.*`, it contains matrix with shape of `(2, numb_dos)`, for `*.ados.out.*`, it contains matrix with shape of `(2, natom x numb_dos)`,

```
# frame - 0: data_dos pred_dos
0.0000000000000000e+00 1.963193264917645342e-03
0.0000000000000000e+00 1.178440836781313727e-03
0.0000000000000000e+00 1.441258071790407769e-04
0.0000000000000000e+00 1.787297933314058174e-03
0.0000000000000000e+00 1.901603280243024940e-03
0.0000000000000000e+00 2.279848925571981155e-03
0.0000000000000000e+00 2.149355854688561607e-03
0.0000000000000000e+00 1.829848459515726056e-03
0.0000000000000000e+00 1.905156512419792225e-03
```

5.13 Type embedding approach

Note: Supported backends: TensorFlow

We generate specific a type embedding vector for each atom type so that we can share one descriptor embedding net and one fitting net in total, which decline training complexity largely.

The training input script is similar to that of `se_e2_a`, but different by adding the type_embedding section.

5.13.1 Theory

Usually, when the type embedding approach is not enabled, for a system with multiple chemical species ($|\{\alpha_i\}| > 1$), parameters of the embedding network $\mathcal{N}_{e,\{2,3\}}$ are as follows chemical-species-wise:

$$(\mathcal{G}^i)_j = \mathcal{N}_{e,2}^{\alpha_i,\alpha_j}(s(r_{ij})) \quad \text{or} \quad (\mathcal{G}^i)_j = \mathcal{N}_{e,2}^{\alpha_j}(s(r_{ij})),$$

$$(\mathcal{G}^i)_{jk} = \mathcal{N}_{e,3}^{\alpha_j,\alpha_k}((\theta_i)_{jk}).$$

Thus, there will be N_t^2 or N_t embedding networks where N_t is the number of chemical species. To improve the performance of matrix operations, $n(i)$ is divided into blocks of different chemical species. Each matrix with a dimension of N_c is divided into corresponding blocks, and each block is padded to $N_c^{\alpha_j}$ separately. The limitation of this approach is that when there are large numbers of chemical species, the number of embedding networks will increase, requiring large memory and decreasing computing efficiency.

Similar to the embedding networks, if the type embedding approach is not used, the fitting network parameters are chemical-species-wise, and there are N_t sets of fitting network parameters. For performance, atoms are sorted by their chemical species α_i in advance. Take an example, the atomic energy E_i is represented as follows:

$$E_i = \mathcal{F}_0^{\alpha_i}(\mathcal{D}^i).$$

To reduce the number of NN parameters and improve computing efficiency when there are large numbers of chemical species, the type embedding $\mathcal{A}$ is introduced, represented as a NN function $\mathcal{N}_t$ of the atomic type α:

$$\mathcal{A}^i = \mathcal{N}_t\big(\text{one hot}(\alpha_i)\big),$$

where α_i is converted to a one-hot vector representing the chemical species before feeding to the NN. The type embeddings of central and neighboring atoms $\mathcal{A}^i$ and $\mathcal{A}^j$ are added as an extra input of the embedding network $\mathcal{N}_{e,\{2,3\}}$:

$$(\mathcal{G}^i)_j = \mathcal{N}_{e,2}(\{s(r_{ij}), \mathcal{A}^i, \mathcal{A}^j\}) \quad \text{or} \quad (\mathcal{G}^i)_j = \mathcal{N}_{e,2}(\{s(r_{ij}), \mathcal{A}^j\}),$$

$$(\mathcal{G}^i)_{jk} = \mathcal{N}_{e,3}(\{(\theta_i)_{jk}, \mathcal{A}^j, \mathcal{A}^k\}).$$

In fitting networks, the type embedding is inserted into the input of the fitting networks:

$$E_i = \mathcal{F}_0(\{\mathcal{D}^i, \mathcal{A}^i\}).$$

In this way, all chemical species share the same network parameters through the type embedding.[1]

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

5.13.2 Instructions

The model defines how the model is constructed, adding a section of type embedding net:

```
"model": {
    "type_map":        ["O", "H"],
    "type_embedding":{
                    ...
    },
    "descriptor" :{
        ...
    },
    "fitting_net" : {
        ...
    }
}
```

The model will automatically apply the type embedding approach and generate type embedding vectors. If the type embedding vector is detected, the descriptor and fitting net would take it as a part of the input.

The construction of type embedding net is given by type_embedding. An example of type_embedding is provided as follows

```
    "type_embedding":{
        "neuron":               [2, 4, 8],
        "resnet_dt":        false,
        "seed":                1
    }
```

- The neuron specifies the size of the type embedding net. From left to right the members denote the sizes of each hidden layer from the input end to the output end, respectively. It takes a one-hot vector as input and output dimension equals to the last dimension of the neuron list. If the outer layer is twice the size of the inner layer, then the inner layer is copied and concatenated, then a ResNet architecture is built between them.
- If the option resnet_dt is set to `true`, then a timestep is used in the ResNet.
- seed gives the random seed that is used to generate random numbers when initializing the model parameters.

A complete training input script of this example can be found in the directory.

```
$deepmd_source_dir/examples/water/se_e2_a_tebd/input.json
```

See here for further explanation of `type embedding`.

Note: You can't apply the compression method while using the atom type embedding.

5.14 Descriptor "se_a_mask"

Note: Supported backends: TensorFlow

Descriptor `se_a_mask` is a concise implementation of the descriptor `se_e2_a`, but functions slightly differently. `se_a_mask` is specially designed for DP/MM simulations where the number of atoms in DP regions is dynamically changed in simulations.

Therefore, the descriptor `se_a_mask` is not supported for training with PBC systems for simplicity. Besides, to make the output shape of the descriptor matrix consistent, the input coordinates are padded with virtual particle coordinates to the maximum number of atoms (specified with `sel` in the descriptor setting) in the system. The real/virtual sign of the atoms is specified with the `aparam.npy` ([nframes * natoms]) file in the input systems set directory. The `aparam.npy` can also be seen as the mask of the atoms in the system, which is also the origin of the name `se_a_mask`.

In this example, we will train a DP Mask model for zinc protein interactions. The input systems are the collection of zinc and its coordinates residues. A sample input system that contains 2 frames is included in the directory.

```
$deepmd_source_dir/examples/zinc_protein/data_dp_mask
```

A complete training input script of this example can be found in the directory.

```
$deepmd_source_dir/examples/zinc_protein/zinc_se_a_mask.json
```

The construction of the descriptor is given by section descriptor. An example of the descriptor is provided as follows

```
    "descriptor" :{
        "type":         "se_a_mask",
        "sel":                [36, 16, 24, 64, 6, 1],
        "neuron":               [25, 50, 100],
            "axis_neuron": 16,
        "type_one_side":        false,
        "resnet_dt":        false,
        "seed":                 1
    }
```

- The type of the descriptor is set to `"se_a_mask"`.
- sel gives the maximum number of atoms in input coordinates. It is a list, the length of which is the same as the number of atom types in the system, and `sel[i]` denotes the maximum number of atoms with type `i`.
- The neuron specifies the size of the embedding net. From left to right the members denote the sizes of each hidden layer from the input end to the output end, respectively. If the outer layer is twice the size of the inner layer, then the inner layer is copied and concatenated, then a ResNet architecture is built between them.
- The axis_neuron specifies the size of the submatrix of the embedding matrix, the axis matrix as explained in the DeepPot-SE paper
- If the option type_one_side is set to `true`, the embedding network parameters vary by types of neighbor atoms only, so there will be N_{types} sets of embedding network parameters. Otherwise, the embed-

ding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be N_{types}^2 sets of embedding network parameters.

- If the option resnet_dt is set to `true`, then a timestep is used in the ResNet.
- seed gives the random seed that is used to generate random numbers when initializing the model parameters.

To make the `aparam.npy` used for descriptor `se_a_mask`, two variables in `fitting_net` section are needed.

```
        "fitting_net" :{
            "neuron": [240, 240, 240],
              "resnet_dt": true,
              "seed": 1,
              "numb_aparam": 1,
              "use_aparam_as_mask": true
        }
```

- `neuron`, `resnet_dt` and `seed` are the same as the fitting_net section for fitting energy.
- numb_aparam gives the dimesion of the `aparam.npy` file. In this example, it is set to 1 and stores the real/virtual sign of the atoms. For real/virtual atoms, the corresponding sign in `aparam.npy` is set to 1/0.
- use_aparam_as_mask is set to `true` to use the `aparam.npy` as the mask of the atoms in the descriptor `se_a_mask`.

Finally, to make a reasonable fitting task with `se_a_mask` descriptor for DP/MM simulations, the loss function with `se_a_mask` is designed to include the atomic forces difference in specific atoms of the input particles only. More details about the selection of the specific atoms can be found in paper [DP/MM](left to be filled). Thus, `atom_pref.npy` ([nframes * natoms]) is required as the indicator of the specific atoms in the input particles. And the `loss` section in the training input script should be set as follows.

```
"loss": {
    "type": "ener",
    "start_pref_e": 0.0,
    "limit_pref_e": 0.0,
    "start_pref_f": 0.0,
    "limit_pref_f": 0.0,
    "start_pref_pf": 1.0,
    "limit_pref_pf": 1.0,
    "_comment": " that's all"
  }
```

5.15 Deep potential long-range (DPLR)

Note: Supported backends: TensorFlow

Notice: The interfaces of DPLR are not stable and subject to change

The method of DPLR is described in this paper. One is recommended to read the paper before using the DPLR.

In the following, we take the DPLR model for example to introduce the training and LAMMPS simulation with the DPLR model. The DPLR model is trained in two steps.

5.15.1 Theory

The Deep Potential Long Range (DPLR) model adds the electrostatic energy to the total energy:

$$E = E_{\mathrm{DP}} + E_{\mathrm{ele}},$$

where E_{DP} is the short-range contribution constructed as the standard energy model that is fitted against $(E^* - E_{\mathrm{ele}})$. E_{ele} is the electrostatic energy introduced by a group of Gaussian distributions that is an approximation of the electronic structure of the system, and is calculated in Fourier space by

$$E_{\mathrm{ele}} = \frac{1}{2\pi V} \sum_{m \neq 0, \|m\| \leq L} \frac{\exp(-\pi^2 m^2 / \beta^2)}{m^2} S^2(m),$$

where β is a freely tunable parameter that controls the spread of the Gaussians. L is the cutoff in Fourier space and $S(m)$, the structure factor, is given by

$$S(m) = \sum_i q_i e^{-2\pi \imath m \boldsymbol{r}_i} + \sum_n q_n e^{-2\pi \imath m \boldsymbol{W}_n},$$

where $\imath = \sqrt{-1}$ denotes the imaginary unit, $\boldsymbol{r}_i$ indicates ion coordinates, q_i is the charge of the ion i, and W_n is the n-th Wannier centroid (WC) which can be obtained from a separated dipole model. It can be proved that the error in the electrostatic energy introduced by the Gaussian approximations is dominated by a summation of dipole-quadrupole interactions that decay as r^{-4}, where r is the distance between the dipole and quadrupole.[1]

5.15.2 Train a deep Wannier model for Wannier centroids

We use the deep Wannier model (DW) to represent the relative position of the Wannier centroid (WC) with the atom with which it is associated. One may consult the introduction of the dipole model for a detailed introduction. An example input `wc.json` and a small dataset `data` for tutorial purposes can be found in

```
$deepmd_source_dir/examples/water/dplr/train/
```

It is noted that the tutorial dataset is not enough for training a productive model. Two settings make the training input script different from an energy training input:

```
        "fitting_net": {
            "type":                 "dipole",
            "dipole_type":          [0],
            "neuron":                  [128, 128, 128],
            "seed":                 1
        },
```

The type of fitting is set to dipole. The dipole is associated with type 0 atoms (oxygens), by the setting `"dipole_type": [0]`. What we trained is the displacement of the WC from the corresponding oxygen atom. It shares the same training input as the atomic dipole because both are 3-dimensional vectors defined on atoms. The loss section is provided as follows

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

```
    "loss": {
        "type":                "tensor",
        "pref":                0.0,
        "pref_atomic":         1.0
    },
```

so that the atomic dipole is trained as labels. Note that the NumPy compressed file `atomic_dipole.npy` should be provided in each dataset.

The training and freezing can be started from the example directory by

```
dp train dw.json && dp freeze -o dw.pb
```

5.15.3 Train the DPLR model

The training of the DPLR model is very similar to the standard short-range DP models. An example input script can be found in the example directory. The following section is introduced to compute the long-range energy contribution of the DPLR model, and modify the short-range DP model by this part.

```
        "modifier": {
            "type":             "dipole_charge",
            "model_name":       "dw.pb",
            "model_charge_map": [-8],
            "sys_charge_map":   [6, 1],
            "ewald_h":          1.00,
            "ewald_beta":       0.40
        },
```

The model_name specifies which DW model is used to predict the position of WCs. model_charge_map gives the amount of charge assigned to WCs. sys_charge_map provides the nuclear charge of oxygen (type 0) and hydrogen (type 1) atoms. ewald_beta (unit $Å^{-1}$) gives the spread parameter controls the spread of Gaussian charges, and ewald_h (unit Å) assigns the grid size of Fourier transformation. The DPLR model can be trained and frozen by (from the example directory)

```
dp train ener.json && dp freeze -o ener.pb
```

5.15.4 Molecular dynamics simulation with DPLR

In MD simulations, the long-range part of the DPLR is calculated by the LAMMPS `kspace` support. Then the long-range interaction is back-propagated to atoms by DeePMD-kit. This setup is commonly used in classical molecular dynamics simulations as the "virtual site". Unfortunately, LAMMPS does not natively support virtual sites, so we have to hack the LAMMPS code, which makes the input configuration and script a little wired.

An example of an input configuration file and script can be found in

```
$deepmd_source_dir/examples/water/dplr/lmp/
```

We use `atom_style full` for DPLR simulations. the coordinates of the WCs are explicitly written in the configuration file. Moreover, a virtual bond is established between the oxygens and the WCs to indicate they are associated together. The configuration file containing 128 H2O molecules is thus written as

```
512 atoms
3 atom types
128 bonds
1 bond types

0 16.421037674 xlo xhi
0 16.421037674 ylo yhi
0 16.421037674 zlo zhi
0 0 0 xy xz yz

Masses

1 16
2 2
3 16

Atoms

       1        1 1  6 8.4960699081e+00 7.5073699951e+00 9.6371297836e+00
       2        2 1  6 4.0597701073e+00 6.8156299591e+00 1.2051420212e+01
...

     385        1 3 -8 8.4960699081e+00 7.5073699951e+00 9.6371297836e+00
     386        2 3 -8 4.0597701073e+00 6.8156299591e+00 1.2051420212e+01
...

Bonds

1 1 1 385
2 1 2 386
...
```

The oxygens and hydrogens are assigned with atom types 1 and 2 (corresponding to training atom types 0 and 1), respectively. The WCs are assigned with atom type 3. We want to simulate heavy water so the mass of hydrogens is set to 2.

An example input script is provided in

```
$deepmd_source_dir/examples/water/dplr/lmp/in.lammps
```

Here are some explanations

```
# groups of real and virtual atoms
group           real_atom type 1 2
group           virtual_atom type 3

# bond between real and its corresponding virtual site should be given
# to setup a map between real and virtual atoms. However, no real
# bonded interaction is applied, thus bond_sytle "zero" is used.
pair_style      deepmd ener.pb
pair_coeff      * *
bond_style      zero
bond_coeff      *
special_bonds   lj/coul 1 1 1 angle no
```

Type 1 and 2 (O and H) are `real_atoms`, while type 3 (WCs) are `virtual_atoms`. The model file `ener.pb` stores both the DW and DPLR models, so the position of WCs and the energy can be inferred from it. A

virtual bond type is specified by `bond_style zero`. The `special_bonds` command switches off the exclusion of intramolecular interactions.

```
# kspace_style "pppm/dplr" should be used. in addition the
# gewald(1/distance) should be set the same as that used in
# training. Currently only ik differentiation is supported.
kspace_style        pppm/dplr 1e-5
kspace_modify        gewald ${BETA} diff ik mesh ${KMESH} ${KMESH} ${KMESH}
```

The long-range part is calculated by the `kspace` support of LAMMPS. The `kspace_style pppm/dplr` is required. The spread parameter set by variable `BETA` should be set the same as that used in training. The `KMESH` should be set dense enough so the long-range calculation is converged.

fix dplr command

Syntax

```
fix ID group-ID style_name keyword value ...
```

- ID, group-ID are documented in :doc:`fix <fix>` command
- style_name = dplr
- three or more keyword/value pairs may be appended

```
keyword = *model* or *type_associate* or *bond_type* or *efield*
  *model* value = name
    name = name of DPLR model file (e.g. frozen_model.pb) (not DW model)
  *type_associate* values = NR1 NW1 NR2 NW2 ...
    NRi = type of real atom in i-th (real atom, Wannier centroid) pair
    NWi = type of Wannier in i-th (real atom, Wannier centroid) pair
  *bond_type* values = NB1 NB2 ...
    NBi = bond type of i-th (real atom, Wannier centroid) pair
  *efield* (optional) values = Ex Ey Ez
    Ex/Ey/Ez = electric field along x/y/z direction
```

Examples

```
# "fix dplr" set the position of the virtual atom, and spread the
# electrostatic interaction asserting on the virtual atom to the real
# atoms. "type_associate" associates the real atom type its
# corresponding virtual atom type. "bond_type" gives the type of the
# bond between the real and virtual atoms.
fix                 0 all dplr model ener.pb type_associate 1 3 bond_type 1
fix_modify         0 virial yes
```

The fix command `dplr` calculates the position of WCs by the DW model and back-propagates the long-range interaction on virtual atoms to real toms. The atom names specified in pair_style *deepmd* will be used to determine elements. If it is not set, the training parameter type_map will be mapped to LAMMPS atom types.

To use a time-dependent electric field, LAMMPS's `variable` feature can be utilized:

```
variable EFIELD_Z equal 2*sin(2*PI*time/0.006)
fix 0 all dplr model ener.pb type_associate 1 3 bond_type 1 efield 0 0 v_EFIELD_Z
fix_modify 0 energy yes virial yes
```

The `efield` feature of `fix dplr` behaves similarly to LAMMPS's fix efield. Note that the atomic energy or potential in `fix efield` is not yet supported in `fix dplr`. For a detailed description on how a time-dependent variable can be defined, refer to LAMMPS's document of variable.

```
# compute the temperature of real atoms, excluding virtual atom contribution
compute               real_temp real_atom temp
compute               real_press all pressure real_temp
fix                1 real_atom nvt temp ${TEMP} ${TEMP} ${TAU_T}
fix_modify         1 temp real_temp
```

The temperature of the system should be computed from the real atoms. The kinetic contribution in the pressure tensor is also computed from the real atoms. The thermostat is applied to only real atoms. The computed temperature and pressure of real atoms can be accessed by, e.g.

```
fix             thermo_print all print ${THERMO_FREQ} "$(step) $(pe) $(ke) $(etotal) $(enthalpy)
↪$(c_real_temp) $(c_real_press) $(vol) $(c_real_press[1]) $(c_real_press[2]) $(c_real_press[3])"␣
↪append thermo.out screen no title "# step pe ke etotal enthalpy temp press vol pxx pyy pzz"
```

The LAMMPS simulation can be started from the example directory by

```
lmp -i in.lammps
```

If LAMMPS complains that no model file `ener.pb` exists, it can be copied from the training example directory.

The MD simulation lasts for only 20 steps. If one runs a longer simulation, it will blow up, because the model is trained with a very limited dataset for very short training steps, thus is of poor quality.

Another restriction that should be noted is that the energies printed at the zero steps are not correct. This is because at the zero steps the position of the WC has not been updated with the DW model. The energies printed in later steps are correct.

5.16 Deep Potential - Range Correction (DPRc)

Note: Supported backends: TensorFlow, PyTorch, DP

Deep Potential - Range Correction (DPRc) is designed to combine with QM/MM method, and corrects energies from a low-level QM/MM method to a high-level QM/MM method:

$$E = E_{\mathrm{QM}}(\mathrm{R};\mathrm{P}) + E_{\mathrm{QM/MM}}(\mathrm{R};\mathrm{P}) + E_{\mathrm{MM}}(\mathrm{R}) + E_{\mathrm{DPRc}}(\mathrm{R})$$

5.16.1 Theory

Deep Potential - Range Correction (DPRc) was initially designed to correct the potential energy from a fast, linear-scaling low-level semiempirical QM/MM theory to a high-level ''ab initio'' QM/MM theory in a range-correction way to quantitatively correct short and mid-range non-bonded interactions leveraging the non-bonded lists routinely used in molecular dynamics simulations using molecular mechanical force fields such as AMBER. In this way, long-ranged electrostatic interactions can be modeled efficiently using the particle mesh Ewald method or its extensions for multipolar and QM/MM potentials. In a DPRc model, the switch function is modified to disable MM-MM interaction:

$$s_{\mathrm{DPRc}}(r_{ij}) = \begin{cases} 0, & \text{if } i \in \mathrm{MM} \wedge j \in \mathrm{MM}, \\ s(r_{ij}), & \text{otherwise}, \end{cases}$$

where $s_{\mathrm{DPRc}}(r_{ij})$ is the new switch function and $s(r_{ij})$ is the old one. This ensures the forces between MM atoms are zero, i.e.

$$\boldsymbol{F}_{ij} = -\frac{\partial E}{\partial \boldsymbol{r}_{ij}} = 0, \quad i \in \mathrm{MM} \wedge j \in \mathrm{MM}.$$

The fitting network is revised to remove energy bias from MM atoms:

$$E_i = \begin{cases} \mathcal{F}_0(\mathcal{D}^i), & \text{if } i \in \mathrm{QM}, \\ \mathcal{F}_0(\mathcal{D}^i) - \mathcal{F}_0(0), & \text{if } i \in \mathrm{MM}, \end{cases}$$

where 0 is a zero matrix. It is worth mentioning that usage of DPRc is not limited to its initial design for QM/MM correction and can be expanded to any similar interaction.[1]

See the JCTC paper for details.

5.16.2 Training data

Instead the normal ab initio data, one needs to provide the correction from a low-level QM/MM method to a high-level QM/MM method:

$$E = E_{\text{high-level QM/MM}} - E_{\text{low-level QM/MM}}$$

Two levels of data use the same MM method, so E_{MM} is eliminated.

5.16.3 Training the DPRc model

In a DPRc model, QM atoms and MM atoms have different atom types. Assuming we have 4 QM atom types (C, H, O, P) and 2 MM atom types (HW, OW):

```
"type_map": ["C", "H", "HW", "O", "OW", "P"]
```

As described in the paper, the DPRc model only corrects E_{QM} and $E_{\mathrm{QM/MM}}$ within the cutoff, so we use a hybrid descriptor to describe them separatedly:

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

TensorFlow

```
"descriptor" :{
    "type":             "hybrid",
    "list" : [
        {
            "type":     "se_a_ebd_v2",
            "sel":              [6, 11, 0, 6, 0, 1],
            "rcut_smth":        1.00,
            "rcut":             9.00,
            "neuron":           [12, 25, 50],
            "exclude_types":    [[2, 2], [2, 4], [4, 4], [0, 2], [0, 4], [1, 2], [1, 4], [3, 2],␣
↪[3, 4], [5, 2], [5, 4]],
            "axis_neuron":      12,
            "_comment": " QM/QM interaction"
        },
        {
            "type":     "se_a_ebd_v2",
            "sel":              [6, 11, 100, 6, 50, 1],
            "rcut_smth":        0.50,
            "rcut":             6.00,
            "neuron":           [12, 25, 50],
            "exclude_types":    [[0, 0], [0, 1], [0, 3], [0, 5], [1, 1], [1, 3], [1, 5], [3, 3],␣
↪[3, 5], [5, 5], [2, 2], [2, 4], [4, 4]],
            "axis_neuron":      12,
            "set_davg_zero":    true,
            "_comment": " QM/MM interaction"
        }
    ]
}
```

PyTorch

```
"descriptor" :{
    "type":             "hybrid",
    "list" : [
        {
            "type":     "se_e2_a",
            "sel":              [6, 11, 0, 6, 0, 1],
            "rcut_smth":        1.00,
            "rcut":             9.00,
            "neuron":           [12, 25, 50],
            "exclude_types":    [[2, 2], [2, 4], [4, 4], [0, 2], [0, 4], [1, 2], [1, 4], [3, 2],␣
↪[3, 4], [5, 2], [5, 4]],
            "axis_neuron":      12,
            "type_one_side":    true,
            "_comment": " QM/QM interaction"
        },
        {
            "type":     "se_e2_a",
            "sel":              [6, 11, 100, 6, 50, 1],
            "rcut_smth":        0.50,
            "rcut":             6.00,
            "neuron":           [12, 25, 50],
```

(continues on next page)

(continued from previous page)

```
            "exclude_types":    [[0, 0], [0, 1], [0, 3], [0, 5], [1, 1], [1, 3], [1, 5], [3, 3],␣
↪[3, 5], [5, 5], [2, 2], [2, 4], [4, 4]],
            "axis_neuron":      12,
            "set_davg_zero":    true,
            "type_one_side":    true,
            "_comment": " QM/MM interaction"
        }
    ]
}
```

exclude_types can be generated by the following Python script:

```
from itertools import combinations_with_replacement, product

qm = (0, 1, 3, 5)
mm = (2, 4)
print(
    "QM/QM:",
    list(map(list, list(combinations_with_replacement(mm, 2)) + list(product(qm, mm)))),
)
print(
    "QM/MM:",
    list(
        map(
            list,
            list(combinations_with_replacement(qm, 2))
            + list(combinations_with_replacement(mm, 2)),
        )
    ),
)
```

Also, DPRc assumes MM atom energies (atom_ener) are zero:

```
"fitting_net": {
   "neuron": [240, 240, 240],
   "resnet_dt": true,
   "atom_ener": [null, null, 0.0, null, 0.0, null]
}
```

Note that atom_ener only works when descriptor/set_davg_zero of the QM/MM part is `true`.

5.16.4 Run MD simulations

The DPRc model has the best practices with the AMBER QM/MM module. An example is given by GitLab RutgersLBSR/AmberDPRc. In theory, DPRc is able to be used with any QM/MM package, as long as the DeePMD-kit package accepts QM atoms and MM atoms within the cutoff range and returns energies and forces.

5.16.5 Pairwise DPRc

Note: Supported backends: TensorFlow

If one wants to correct from a low-level method into a full DFT level, and the system is too large to do full DFT calculation, one may try the experimental pairwise DPRc model. In a pairwise DPRc model, the total energy is divided into QM internal energy and the sum of QM/MM energy for each MM residue l:

$$E = E*\mathrm{QM} + \sum *lE_\mathrm{QM/MM}, l$$

In this way, the interaction between the QM region and each MM fragmentation can be computed and trained separately. Thus, the pairwise DPRc model is divided into two sub-DPRc models. `qm_model` is for the QM internal interaction and `qmmm_model` is for the QM/MM interaction. The configuration for these two models is similar to the non-pairwise DPRc model. It is noted that the *`se_atten`* descriptor should be used, as it is the only descriptor to support the mixed type.

```
{
  "model": {
    "type": "pairwise_dprc",
    "type_map": ["C", "P", "O", "H", "OW", "HW"],
    "type_embedding": {
      "neuron": [8],
      "precision": "float32"
    },
    "qm_model": {
      "descriptor": {
        "type": "se_atten_v2",
        "sel": 24,
        "rcut_smth": 0.5,
        "rcut": 9.0,
        "attn_layer": 0,
        "neuron": [25, 50, 100],
        "resnet_dt": false,
        "axis_neuron": 12,
        "precision": "float32",
        "seed": 1
      },
      "fitting_net": {
        "type": "ener",
        "neuron": [240, 240, 240],
        "resnet_dt": true,
        "precision": "float32",
        "atom_ener": [null, null, null, null, 0.0, 0.0],
        "seed": 1
      }
    },
```

(continues on next page)

(continued from previous page)

```
    "qmmm_model": {
      "descriptor": {
        "type": "se_atten_v2",
        "sel": 27,
        "rcut_smth": 0.5,
        "rcut": 6.0,
        "attn_layer": 0,
        "neuron": [25, 50, 100],
        "resnet_dt": false,
        "axis_neuron": 12,
        "set_davg_zero": true,
        "exclude_types": [
          [0, 0],
          [0, 1],
          [0, 2],
          [0, 3],
          [1, 1],
          [1, 2],
          [1, 3],
          [2, 2],
          [2, 3],
          [3, 3],
          [4, 4],
          [4, 5],
          [5, 5]
        ],
        "precision": "float32",
        "seed": 1
      },
      "fitting_net": {
        "type": "ener",
        "neuron": [240, 240, 240],
        "resnet_dt": true,
        "seed": 1,
        "precision": "float32",
        "atom_ener": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
      }
    }
  }
}
```

The pairwise model needs information for MM residues. The model uses *aparam* with the shape of `nframes x natoms` to get the residue index. The QM residue should always use `0` as the index. For example, `0 0 0 1 1 1 2 2 2` means these 9 atoms are grouped into one QM residue and two MM residues.

5.17 Linear model

Note: Supported backends: TensorFlow

One can linearly combine existing models with arbitrary coefficients:

```
"model": {
    "type": "linear_ener",
    "models": [
    {
        "type": "frozen",
        "model_file": "model0.pb"
    },
    {
        "type": "frozen",
        "model_file": "model1.pb"
    }
    ],
    "weights": [0.5, 0.5]
},
```

weights can be a list of floats, `mean`, or `sum`.

To obtain the model, one needs to execute `dp train` to do a zero-step training with numb_steps set to `0`, and then freeze the model with `dp freeze`.

5.18 Interpolation or combination with a pairwise potential

Note: Supported backends: TensorFlow, PyTorch, DP

5.18.1 Theory

In applications like the radiation damage simulation, the interatomic distance may become too close, so that the DFT calculations fail. In such cases, the DP model that is an approximation of the DFT potential energy surface is usually replaced by an empirical potential, like the Ziegler-Biersack-Littmark (ZBL) screened nuclear repulsion potential in the radiation damage simulations. The DeePMD-kit package supports the interpolation between DP and an empirical pairwise potential

$$E_i = (1 - w_i)E_i^{\mathrm{DP}} + w_i(E_i^0 + E_i^{\mathrm{pair}}),$$

where the w_i is the interpolation weight and the E_i^{pair} is the atomic contribution due to the pairwise potential $u^{\mathrm{pair}}(r)$, i.e.

$$E_i^{\mathrm{pair}} = \sum_{j \in n(i)} u^{\mathrm{pair}}(r_{ij}).$$

The interpolation weight w_i is defined by

$$w_i = \begin{cases} 1, & \sigma_i r_a, \\ u_i^3(-6u_i^2 + 15u_i - 10) + 1, & r_a \leq \sigma_i r_b, \\ 0, & \sigma_i \geq r_b, \end{cases}$$

where $u_i = (\sigma_i - r_a)/(r_b - r_a)$. E_i^0 is the atom energy bias. In the range $[r_a, r_b]$, the DP model smoothly switched off and the pairwise potential smoothly switched on from r_b to r_a. The σ_i is the softmin of the distance between atom i and its neighbors,

$$\sigma_i = \frac{\sum\limits_{j \in n(i)} r_{ij} e^{-r_{ij}/\alpha_s}}{\sum\limits_{j \in n(i)} e^{-r_{ij}/\alpha_s}},$$

where the scale α_s is a tunable scale of the interatomic distance r_{ij}. The pairwise potential $u^{\mathrm{pair}}(r)$ is defined by a user-defined table that provides the value of u^{pair} on an evenly discretized grid from 0 to the cutoff distance.[1]

DeePMD-kit also supports combination with a pairwise potential :

$$E_i = E_i^{\mathrm{DP}} + E_i^{\mathrm{pair}},$$

5.18.2 Table file

The table file should be a text file that can be read by `numpy.loadtxt()`. The first column is the distance between two atoms, where upper range should be larger than the cutoff radius. Other columns are two-body interaction energies for pairs of certain types, in the order of Type_0-Type_0, Type_0-Type_1, ..., Type_0-Type_N, Type_1-Type_1, ..., Type_1-Type_N, ..., and Type_N-Type_N.

The interaction should be smooth at the cut-off distance.

Note: In instances where the interaction at the cut-off distance is not delineated within the table file, extrapolation will be conducted utilizing the available interaction data. This extrapolative procedure guarantees a smooth transition from the table-provided value to `0` whenever feasible.

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

5.18.3 Interpolation with a short-range pairwise potential

```
"model": {
  "use_srtab": "H2O_tab_potential.txt",
  "smin_alpha": 0.1,
  "sw_rmin": 0.8,
  "sw_rmax": 1.0,
  "_comment": "Below uses a normal DP model"
}
```

sw_rmin and sw_rmax must be smaller than the cutoff radius of the DP model.

5.18.4 Combination with a pairwise potential

To combine with a pairwise potential, use the linear model:

```
"model": {
  "type": "linear_ener",
  "weights": "sum",
  "models": [
    {
      "_comment": "Here uses a normal DP model"
    },
    {
      "type": "pairtab",
      "tab_file": "dftd3.txt",
      "rcut": 10.0,
      "sel": 534
    }
  ]
}
```

The rcut can be larger than that of the DP model.

CHAPTER

SIX

TRAINING

6.1 Train a model

Several examples of training can be found in the `examples` directory:

```
$ cd $deepmd_source_dir/examples/water/se_e2_a/
```

After switching to that directory, the training can be invoked by

```
$ dp train input.json
```

where `input.json` is the name of the input script.

By default, the verbosity level of the DeePMD-kit is `INFO`, one may see a lot of important information on the code and environment showing on the screen. Among them two pieces of information regarding data systems are worth special notice.

```
DEEPMD INFO    ---Summary of DataSystem: training     -----------------------------------------------
↪--
DEEPMD INFO    found 3 system(s):
DEEPMD INFO                                        system  natoms  bch_sz   n_bch   prob  pbc
DEEPMD INFO                          ../data_water/data_0/     192       1      80  0.250    T
DEEPMD INFO                          ../data_water/data_1/     192       1     160  0.500    T
DEEPMD INFO                          ../data_water/data_2/     192       1      80  0.250    T
DEEPMD INFO    ------------------------------------------------------------------------------------
↪--
DEEPMD INFO    ---Summary of DataSystem: validation   -----------------------------------------------
↪--
DEEPMD INFO    found 1 system(s):
DEEPMD INFO                                        system  natoms  bch_sz   n_bch   prob  pbc
DEEPMD INFO                           ../data_water/data_3     192       1      80  1.000    T
DEEPMD INFO    ------------------------------------------------------------------------------------
↪--
```

The DeePMD-kit prints detailed information on the training and validation data sets. The data sets are defined by training_data and validation_data defined in the training section of the input script. The training data set is composed of three data systems, while the validation data set is composed by one data system. The number of atoms, batch size, the number of batches in the system and the probability of using the system are all shown on the screen. The last column presents if the periodic boundary condition is assumed for the system.

During the training, the error of the model is tested every disp_freq training steps with the batch used to train the model and with numb_btch batches from the validating data. The training error and validation error are printed correspondingly in the file disp_file (default is `lcurve.out`). The batch size can be set in the

input script by the key batch_size in the corresponding sections for the training and validation data set. An example of the output

```
#  step      rmse_val    rmse_trn    rmse_e_val  rmse_e_trn    rmse_f_val  rmse_f_trn         lr
      0      3.33e+01    3.41e+01      1.03e+01    1.03e+01      8.39e-01    8.72e-01    1.0e-03
    100      2.57e+01    2.56e+01      1.87e+00    1.88e+00      8.03e-01    8.02e-01    1.0e-03
    200      2.45e+01    2.56e+01      2.26e-01    2.21e-01      7.73e-01    8.10e-01    1.0e-03
    300      1.62e+01    1.66e+01      5.01e-02    4.46e-02      5.11e-01    5.26e-01    1.0e-03
    400      1.36e+01    1.32e+01      1.07e-02    2.07e-03      4.29e-01    4.19e-01    1.0e-03
    500      1.07e+01    1.05e+01      2.45e-03    4.11e-03      3.38e-01    3.31e-01    1.0e-03
```

The file contains 8 columns, from left to right, which are the training step, the validation loss, training loss, root mean square (RMS) validation error of energy, RMS training error of energy, RMS validation error of force, RMS training error of force and the learning rate. The RMS error (RMSE) of the energy is normalized by the number of atoms in the system. One can visualize this file with a simple Python script:

```
import numpy as np
import matplotlib.pyplot as plt

data = np.genfromtxt("lcurve.out", names=True)
for name in data.dtype.names[1:-1]:
    plt.plot(data["step"], data[name], label=name)
plt.legend()
plt.xlabel("Step")
plt.ylabel("Loss")
plt.xscale("symlog")
plt.yscale("log")
plt.grid()
plt.show()
```

Checkpoints will be written to files with the prefix save_ckpt every save_freq training steps.

Warning: It is warned that the example water data (in folder `examples/water/data`) is of very limited amount, is provided only for testing purposes, and should not be used to train a production model.

6.2 Advanced options

In this section, we will take `$deepmd_source_dir/examples/water/se_e2_a/input.json` as an example of the input file.

6.2.1 Learning rate

Theory

The learning rate γ decays exponentially:

$$\gamma(\tau) = \gamma^0 r^{\lfloor \tau / s \rfloor},$$

where $\tau \in \mathbb{N}$ is the index of the training step, $\gamma^0 \in \mathbb{R}$ is the learning rate at the first step, and the decay rate r is given by

$$r = \left(\frac{\gamma^{\mathrm{stop}}}{\gamma^0} \right)^{\frac{s}{\tau^{\mathrm{stop}}}},$$

where $\tau^{\text{stop}} \in \mathbb{N}$, $\gamma^{\text{stop}} \in \mathbb{R}$, and $s \in \mathbb{N}$ are the stopping step, the stopping learning rate, and the decay steps, respectively, all of which are hyperparameters provided in advance.[1]

Instructions

The learning_rate section in `input.json` is given as follows

```
    "learning_rate" :{
        "type":                "exp",
        "start_lr":          0.001,
        "stop_lr":          3.51e-8,
        "decay_steps":          5000,
        "_comment":          "that's all"
    }
```

- start_lr gives the learning rate at the beginning of the training.
- stop_lr gives the learning rate at the end of the training. It should be small enough to ensure that the network parameters satisfactorily converge.
- During the training, the learning rate decays exponentially from start_lr to stop_lr following the formula:

```
lr(t) = start_lr * decay_rate ^ ( t / decay_steps )
```

6.2.2 Training parameters

Other training parameters are given in the training section.

```
    "training": {
         "training_data": {
            "systems":                 ["../data_water/data_0/", "../data_water/data_1/", "../data_
↪water/data_2/"],
            "batch_size":          "auto"
        },
        "validation_data":{
            "systems":                 ["../data_water/data_3"],
            "batch_size":          1,
            "numb_btch":           3
        },
        "mixed_precision": {
            "output_prec":       "float32",
            "compute_prec":      "float16"
        },

        "numb_steps":          1000000,
        "seed":                  1,
        "disp_file":          "lcurve.out",
        "disp_freq":          100,
```

(continues on next page)

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

(continued from previous page)

```
    "save_freq":        1000
}
```

The sections training_data and validation_data give the training dataset and validation dataset, respectively. Taking the training dataset for example, the keys are explained below:

- systems provide paths of the training data systems. DeePMD-kit allows you to provide multiple systems with different numbers of atoms. This key can be a `list` or a `str`.
  - `list`: systems gives the training data systems.
  - `str`: systems should be a valid path. DeePMD-kit will recursively search all data systems in this path.
- At each training step, DeePMD-kit randomly picks batch_size frame(s) from one of the systems. The probability of using a system is by default in proportion to the number of batches in the system. More options are available for automatically determining the probability of using systems. One can set the key auto_prob to
  - `"prob_uniform"` all systems are used with the same probability.
  - `"prob_sys_size"` the probability of using a system is proportional to its size (number of frames).
  - `"prob_sys_size; sidx_0:eidx_0:w_0; sidx_1:eidx_1:w_1;..."` the `list` of systems is divided into blocks. Block `i` has systems ranging from `sidx_i` to `eidx_i`. The probability of using a system from block `i` is proportional to `w_i`. Within one block, the probability of using a system is proportional to its size.
- An example of using `"auto_prob"` is given below. The probability of using `systems[2]` is 0.4, and the sum of the probabilities of using `systems[0]` and `systems[1]` is 0.6. If the number of frames in `systems[1]` is twice of `system[0]`, then the probability of using `system[1]` is 0.4 and that of `system[0]` is 0.2.

```
    "training_data": {
        "systems":              ["../data_water/data_0/", "../data_water/data_1/", "../data_
↪water/data_2/"],
        "auto_prob":        "prob_sys_size; 0:2:0.6; 2:3:0.4",
        "batch_size":        "auto"
    }
```

- The probability of using systems can also be specified explicitly with key sys_probs which is a list having the length of the number of systems. For example

```
    "training_data": {
        "systems":              ["../data_water/data_0/", "../data_water/data_1/", "../data_
↪water/data_2/"],
        "sys_probs":        [0.5, 0.3, 0.2],
        "batch_size":        "auto:32"
    }
```

- The key batch_size specifies the number of frames used to train or validate the model in a training step. It can be set to
  - `list`: the length of which is the same as the systems. The batch size of each system is given by the elements of the list.
  - `int`: all systems use the same batch size.
  - `"auto"`: the same as `"auto:32"`, see `"auto:N"`

  - `"auto:N"`: automatically determines the batch size so that the batch_size times the number of atoms in the system is no less than `N`.

- The key numb_batch in validate_data gives the number of batches of model validation. Note that the batches may not be from the same system

The section mixed_precision specifies the mixed precision settings, which will enable the mixed precision training workflow for DeePMD-kit. The keys are explained below:

- output_prec precision used in the output tensors, only `float32` is supported currently.
- compute_prec precision used in the computing tensors, only `float16` is supported currently. Note there are several limitations about mixed precision training:
- Only se_e2_a type descriptor is supported by the mixed precision training workflow.
- The precision of the embedding net and the fitting net are forced to be set to `float32`.

Other keys in the training section are explained below:

- numb_steps The number of training steps.
- seed The random seed for getting frames from the training data set.
- disp_file The file for printing learning curve.
- disp_freq The frequency of printing learning curve. Set in the unit of training steps
- save_freq The frequency of saving checkpoint.

6.2.3 Options and environment variables

Several command line options can be passed to `dp train`, which can be checked with

```
$ dp train --help
```

An explanation will be provided

```
positional arguments:
  INPUT                 the input json database

optional arguments:
  -h, --help            show this help message and exit

  --init-model INIT_MODEL
                        Initialize a model by the provided checkpoint

  --restart RESTART     Restart the training from the provided checkpoint

  --init-frz-model INIT_FRZ_MODEL
                        Initialize the training from the frozen model.
  --skip-neighbor-stat  Skip calculating neighbor statistics. Sel checking, automatic sel, and␣
↪model compression will be disabled. (default: False)
```

`--init-model model.ckpt`, initializes the model training with an existing model that is stored in the path prefix of checkpoint files `model.ckpt`, the network architectures should match.

`--restart model.ckpt`, continues the training from the checkpoint `model.ckpt`.

`--init-frz-model frozen_model.pb`, initializes the training with an existing model that is stored in `frozen_model.pb`.

`--skip-neighbor-stat` will skip calculating neighbor statistics if one is concerned about performance. Some features will be disabled.

To maximize the performance, one should follow FAQ: How to control the parallelism of a job to control the number of threads.

One can set other environmental variables:

Environment variables	Allowed value	Default value	Usage
DP_INTERFACE_I	`high`, `low`	`high`	Control high (double) or low (float) precision of training.
DP_AUTO_PARAL	0, 1	0	Enable auto parallelization for CPU operators.
DP_JIT	0, 1	0	Enable JIT. Note that this option may either improve or decrease the performance. Requires TensorFlow supports JIT.

6.2.4 Adjust `sel` of a frozen model

One can use `--init-frz-model` features to adjust (increase or decrease) *`sel`* of a existing model. Firstly, one needs to adjust *`sel`* in `input.json`. For example, adjust from `[46, 92]` to `[23, 46]`.

```
"model": {
        "descriptor": {
                "sel": [23, 46]
        }
}
```

To obtain the new model at once, *`numb_steps`* should be set to zero:

```
"training": {
        "numb_steps": 0
}
```

Then, one can initialize the training from the frozen model and freeze the new model at once:

```
dp train input.json --init-frz-model frozen_model.pb
dp freeze -o frozen_model_adjusted_sel.pb
```

Two models should give the same result when the input satisfies both constraints.

Note: At this time, this feature is only supported by *`se_e2_a`* descriptor with *`set_davg_true`* enabled, or `hybrid` composed of the above descriptors.

6.3 Training Parameters

Note: One can load, modify, and export the input file by using our effective web-based tool DP-GUI online or hosted using the command line interface `dp gui`. All training parameters below can be set in DP-GUI. By clicking "SAVE JSON", one can download the input file for furthur training.

`model:`

type: `dict`
argument path: `model`

`type_map:`

type: `typing.List[str]`, optional
argument path: `model/type_map`

A list of strings. Give the name to each type of atoms. It is noted that the number of atom type of training system must be less than 128 in a GPU environment. If not given, type.raw in each system should use the same type indexes, and type_map.raw will take no effect.

`data_stat_nbatch:`

type: `int`, optional, default: `10`
argument path: `model/data_stat_nbatch`

The model determines the normalization from the statistics of the data. This key specifies the number of frames in each system used for statistics.

`data_stat_protect:`

type: `float`, optional, default: `0.01`
argument path: `model/data_stat_protect`

Protect parameter for atomic energy regression.

`data_bias_nsample:`

type: `int`, optional, default: `10`
argument path: `model/data_bias_nsample`

The number of training samples in a system to compute and change the energy bias.

`use_srtab:`

type: `str`, optional
argument path: `model/use_srtab`

(Supported Backend: TensorFlow) The table for the short-range pairwise interaction added on top of DP. The table is a text data file with (N_t + 1) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

`smin_alpha:`

type: `float`, optional
argument path: `model/smin_alpha`

(Supported Backend: TensorFlow) The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when use_srtab is provided.

sw_rmin:

type: `float`, optional

argument path: `model/sw_rmin`

(Supported Backend: TensorFlow) The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

sw_rmax:

type: `float`, optional

argument path: `model/sw_rmax`

(Supported Backend: TensorFlow) The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

pair_exclude_types:

type: `list`, optional, default: `[]`

argument path: `model/pair_exclude_types`

(Supported Backend: PyTorch) The atom pairs of the listed types are not treated to be neighbors, i.e. they do not see each other.

atom_exclude_types:

type: `list`, optional, default: `[]`

argument path: `model/atom_exclude_types`

(Supported Backend: PyTorch) Exclude the atomic contribution of the listed atom types

srtab_add_bias:

type: `bool`, optional, default: `True`

argument path: `model/srtab_add_bias`

(Supported Backend: TensorFlow) Whether add energy bias from the statistics of the data to short-range tabulated atomic energy. It only takes effect when use_srtab is provided.

type_embedding:

type: `dict`, optional

argument path: `model/type_embedding`

(Supported Backend: TensorFlow) The type embedding.

neuron:

type: `typing.List[int]`, optional, default: `[8]`

argument path: `model/type_embedding/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

activation_function:

type: str, optional, default: tanh

argument path: model/type_embedding/activation_function

The activation function in the embedding net. Supported activation functions are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”. Note that “gelu” denotes the custom operator version, and “gelu_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

resnet_dt:

type: bool, optional, default: False

argument path: model/type_embedding/resnet_dt

Whether to use a “Timestep” in the skip connection

precision:

type: str, optional, default: default

argument path: model/type_embedding/precision

The precision of the embedding net parameters, supported options are “default”, “float64”, “float16”, “float32”. Default follows the interface precision.

trainable:

type: bool, optional, default: True

argument path: model/type_embedding/trainable

If the parameters in the embedding net are trainable

seed:

type: int | NoneType, optional, default: None

argument path: model/type_embedding/seed

Random seed for parameter initialization

modifier:

type: dict, optional

argument path: model/modifier

(Supported Backend: TensorFlow) The modifier of model output.

Depending on the value of type, different sub args are accepted.

type:

type: str (flag key)

argument path: model/modifier/type

possible choices: *dipole_charge*

The type of modifier. See explanation below.

-dipole_charge: Use WFCC to model the electronic structure of the system. Correct the long-range interaction

When type is set to dipole_charge:

model_name:

type: str

argument path: `model/modifier[dipole_charge]/model_name`

The name of the frozen dipole model file.

model_charge_map:

type: `typing.List[float]`

argument path: `model/modifier[dipole_charge]/model_charge_map`

The charge of the WFCC. The list length should be the same as the \`sel_type <model/fitting_net[dipole]/sel_type_>\`_.

sys_charge_map:

type: `typing.List[float]`

argument path: `model/modifier[dipole_charge]/sys_charge_map`

The charge of real atoms. The list length should be the same as the type_map

ewald_beta:

type: `float`, optional, default: `0.4`

argument path: `model/modifier[dipole_charge]/ewald_beta`

The splitting parameter of Ewald sum. Unit is A^-1

ewald_h:

type: `float`, optional, default: `1.0`

argument path: `model/modifier[dipole_charge]/ewald_h`

The grid spacing of the FFT grid. Unit is A

compress:

type: `dict`, optional

argument path: `model/compress`

(Supported Backend: TensorFlow) Model compression configurations

spin:

type: `dict`, optional

argument path: `model/spin`

The settings for systems with spin.

use_spin:

type: `typing.List[bool]` | `typing.List[int]`

argument path: `model/spin/use_spin`

Whether to use atomic spin model for each atom type. List of boolean values with the shape of [ntypes] to specify which types use spin, or a list of integer values (Supported Backend: PyTorch) to indicate the index of the type that uses spin.

spin_norm:

type: `typing.List[float]`, optional

argument path: `model/spin/spin_norm`

(Supported Backend: TensorFlow) The magnitude of atomic spin for each atom type with spin

virtual_len:

type: typing.List[float], optional

argument path: model/spin/virtual_len

(Supported Backend: TensorFlow) The distance between virtual atom representing spin and its corresponding real atom for each atom type with spin

virtual_scale:

type: typing.List[float] | float, optional

argument path: model/spin/virtual_scale

(Supported Backend: PyTorch) The scaling factor to determine the virtual distance between a virtual atom representing spin and its corresponding real atom for each atom type with spin. This factor is defined as the virtual distance divided by the magnitude of atomic spin for each atom type with spin. The virtual coordinate is defined as the real coordinate plus spin * virtual_scale. List of float values with shape of [ntypes] or [ntypes_spin] or one single float value for all types, only used when use_spin is True for each atom type.

Depending on the value of type, different sub args are accepted.

type:

type: str (flag key), default: standard

argument path: model/type

possible choices: *standard*, *multi*, *frozen*, *pairtab*, *pairwise_dprc*, *linear_ener*

When type is set to standard:

Stardard model, which contains a descriptor and a fitting.

descriptor:

type: dict

argument path: model[standard]/descriptor

The descriptor of atomic environment.

Depending on the value of type, different sub args are accepted.

type:

type: str (flag key)

argument path: model[standard]/descriptor/type

possible choices: *loc_frame*, *se_e2_a*, *se_e3*, *se_a_tpe*, *se_e2_r*, *hybrid*, *se_atten*, *se_atten_v2*, *dpa2*, *se_a_ebd_v2*, *se_a_mask*

The type of the descritpor. See explanation below.

- loc_frame: Defines a local frame at each atom, and the compute the descriptor as local coordinates under this frame.
- se_e2_a: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor.
- se_e2_r: Used by the smooth edition of Deep Potential. Only the distance between atoms is used to construct the descriptor.
- se_e3: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Three-body embedding will be used by this descriptor.

- se_a_tpe: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Type embedding will be used by this descriptor.
- se_atten: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Attention mechanism will be used by this descriptor.
- se_atten_v2: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Attention mechanism with new modifications will be used by this descriptor.
- se_a_mask: Used by the smooth edition of Deep Potential. It can accept a variable number of atoms in a frame (Non-PBC system). aparam are required as an indicator matrix for the real/virtual sign of input atoms.
- hybrid: Concatenate of a list of descriptors as a new descriptor.

When type is set to `loc_frame`:

(Supported Backend: TensorFlow)

sel_a:

type: `typing.List[int]`

argument path: `model[standard]/descriptor[loc_frame]/sel_a`

A list of integers. The length of the list should be the same as the number of atom types in the system. sel_a[i] gives the selected number of type-i neighbors. The full relative coordinates of the neighbors are used by the descriptor.

sel_r:

type: `typing.List[int]`

argument path: `model[standard]/descriptor[loc_frame]/sel_r`

A list of integers. The length of the list should be the same as the number of atom types in the system. sel_r[i] gives the selected number of type-i neighbors. Only relative distance of the neighbors are used by the descriptor. sel_a[i] + sel_r[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius.

rcut:

type: `float`, optional, default: `6.0`

argument path: `model[standard]/descriptor[loc_frame]/rcut`

The cut-off radius. The default value is 6.0

axis_rule:

type: `typing.List[int]`

argument path:
`model[standard]/descriptor[loc_frame]/axis_rule`

A list of integers. The length should be 6 times of the number of types.

- axis_rule[i*6+0]: class of the atom defining the first axis of type-i atom. 0 for neighbors with full coordinates and 1 for neighbors only with relative distance.
- axis_rule[i*6+1]: type of the atom defining the first axis of type-i atom.
- axis_rule[i*6+2]: index of the axis atom defining the first axis. Note that the neighbors with the same class and type are sorted according to their relative distance.

- axis_rule[i*6+3]: class of the atom defining the second axis of type-i atom. 0 for neighbors with full coordinates and 1 for neighbors only with relative distance.
- axis_rule[i*6+4]: type of the atom defining the second axis of type-i atom.
- axis_rule[i*6+5]: index of the axis atom defining the second axis. Note that the neighbors with the same class and type are sorted according to their relative distance.

When type is set to `se_e2_a` (or its alias `se_a`):

`sel:`

type: `str` | `typing.List[int]`, optional, default: `auto`

argument path: `model[standard]/descriptor[se_e2_a]/sel`

This parameter set the number of selected neighbors for each type of atom. It can be:
- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. sel[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- str. Can be "auto:factor" or "auto". "factor" is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the "factor". Finally the number is wraped up to 4 divisible. The option "auto" is equivalent to "auto:1.1".

`rcut:`

type: `float`, optional, default: `6.0`

argument path: `model[standard]/descriptor[se_e2_a]/rcut`

The cut-off radius.

`rcut_smth:`

type: `float`, optional, default: `0.5`

argument path: `model[standard]/descriptor[se_e2_a]/rcut_smth`

Where to start smoothing. For example the 1/r term is smoothed from rcut to rcut_smth

`neuron:`

type: `typing.List[int]`, optional, default: `[10, 20, 40]`

argument path: `model[standard]/descriptor[se_e2_a]/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

`axis_neuron:`

type: `int`, optional, default: `4`, alias: n_axis_neuron

argument path:
`model[standard]/descriptor[se_e2_a]/axis_neuron`

Size of the submatrix of G (embedding matrix).

activation_function:

type: str, optional, default: tanh

argument path:
model[standard]/descriptor[se_e2_a]/activation_function

The activation function in the embedding net. Supported activation functions are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”. Note that “gelu” denotes the custom operator version, and “gelu_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

resnet_dt:

type: bool, optional, default: False

argument path: model[standard]/descriptor[se_e2_a]/resnet_dt

Whether to use a “Timestep” in the skip connection

type_one_side:

type: bool, optional, default: False

argument path:
model[standard]/descriptor[se_e2_a]/type_one_side

If true, the embedding network parameters vary by types of neighbor atoms only, so there will be $N_text{types}$ sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be $N_text{types}^2$ sets of embedding network parameters.

precision:

type: str, optional, default: default

argument path: model[standard]/descriptor[se_e2_a]/precision

The precision of the embedding net parameters, supported options are “default”, “float64”, “float16”, “float32”. Default follows the interface precision.

trainable:

type: bool, optional, default: True

argument path: model[standard]/descriptor[se_e2_a]/trainable

If the parameters in the embedding net is trainable

seed:

type: int | NoneType, optional

argument path: model[standard]/descriptor[se_e2_a]/seed

Random seed for parameter initialization

exclude_types:

type: typing.List[typing.List[int]], optional, default: []

argument path:
model[standard]/descriptor[se_e2_a]/exclude_types

The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

env_protection:

type: float, optional, default: 0.0

argument path:
`model[standard]/descriptor[se_e2_a]/env_protection`

(Supported Backend: TensorFlow) Protection parameter to prevent division by zero errors during environment matrix calculations. For example, when using paddings, there may be zero distances of neighbors, which may make division by zero error during environment matrix calculations without protection.

`set_davg_zero:`

type: `bool`, optional, default: `False`

argument path:
`model[standard]/descriptor[se_e2_a]/set_davg_zero`

Set the normalization average to zero. This option should be set when atom_ener in the energy fitting is used

When type is set to `se_e3` (or its aliases `se_at`, `se_a_3be`, `se_t`):

(Supported Backend: TensorFlow)

`sel:`

type: `str` | `typing.List[int]`, optional, default: `auto`

argument path: `model[standard]/descriptor[se_e3]/sel`

This parameter set the number of selected neighbors for each type of atom. It can be:

- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. sel[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- str. Can be "auto:factor" or "auto". "factor" is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the "factor". Finally the number is wraped up to 4 divisible. The option "auto" is equivalent to "auto:1.1".

`rcut:`

type: `float`, optional, default: `6.0`

argument path: `model[standard]/descriptor[se_e3]/rcut`

The cut-off radius.

`rcut_smth:`

type: `float`, optional, default: `0.5`

argument path: `model[standard]/descriptor[se_e3]/rcut_smth`

Where to start smoothing. For example the 1/r term is smoothed from rcut to rcut_smth

`neuron:`

type: `typing.List[int]`, optional, default: `[10, 20, 40]`

argument path: `model[standard]/descriptor[se_e3]/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

activation_function:

type: str, optional, default: tanh

argument path:
model[standard]/descriptor[se_e3]/activation_function

The activation function in the embedding net. Supported activation functions are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf". Note that "gelu" denotes the custom operator version, and "gelu_tf" denotes the TF standard version. If you set "None" or "none" here, no activation function will be used.

resnet_dt:

type: bool, optional, default: False

argument path: model[standard]/descriptor[se_e3]/resnet_dt

Whether to use a "Timestep" in the skip connection

precision:

type: str, optional, default: default

argument path: model[standard]/descriptor[se_e3]/precision

The precision of the embedding net parameters, supported options are "default", "float64", "float16", "float32". Default follows the interface precision.

trainable:

type: bool, optional, default: True

argument path: model[standard]/descriptor[se_e3]/trainable

If the parameters in the embedding net are trainable

seed:

type: int | NoneType, optional

argument path: model[standard]/descriptor[se_e3]/seed

Random seed for parameter initialization

set_davg_zero:

type: bool, optional, default: False

argument path:
model[standard]/descriptor[se_e3]/set_davg_zero

Set the normalization average to zero. This option should be set when atom_ener in the energy fitting is used

When type is set to se_a_tpe (or its alias se_a_ebd):

(Supported Backend: TensorFlow)

sel:

type: str | typing.List[int], optional, default: auto

argument path: model[standard]/descriptor[se_a_tpe]/sel

This parameter set the number of selected neighbors for each type of atom. It can be:

- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i

neighbors. sel[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- str. Can be "auto:factor" or "auto". "factor" is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the "factor". Finally the number is wraped up to 4 divisible. The option "auto" is equivalent to "auto:1.1".

rcut:

type: `float`, optional, default: `6.0`

argument path: `model[standard]/descriptor[se_a_tpe]/rcut`

The cut-off radius.

rcut_smth:

type: `float`, optional, default: `0.5`

argument path: `model[standard]/descriptor[se_a_tpe]/rcut_smth`

Where to start smoothing. For example the 1/r term is smoothed from rcut to rcut_smth

neuron:

type: `typing.List[int]`, optional, default: `[10, 20, 40]`

argument path: `model[standard]/descriptor[se_a_tpe]/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

axis_neuron:

type: `int`, optional, default: `4`, alias: n_axis_neuron

argument path:
`model[standard]/descriptor[se_a_tpe]/axis_neuron`

Size of the submatrix of G (embedding matrix).

activation_function:

type: `str`, optional, default: `tanh`

argument path:
`model[standard]/descriptor[se_a_tpe]/activation_function`

The activation function in the embedding net. Supported activation functions are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf". Note that "gelu" denotes the custom operator version, and "gelu_tf" denotes the TF standard version. If you set "None" or "none" here, no activation function will be used.

resnet_dt:

type: `bool`, optional, default: `False`

argument path: `model[standard]/descriptor[se_a_tpe]/resnet_dt`

Whether to use a "Timestep" in the skip connection

type_one_side:

type: `bool`, optional, default: `False`

argument path:
model[standard]/descriptor[se_a_tpe]/type_one_side

If true, the embedding network parameters vary by types of neighbor atoms only, so there will be $N_text{types}$ sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be $N_text{types}^2$ sets of embedding network parameters.

precision:

type: str, optional, default: default

argument path: model[standard]/descriptor[se_a_tpe]/precision

The precision of the embedding net parameters, supported options are "default", "float64", "float16", "float32". Default follows the interface precision.

trainable:

type: bool, optional, default: True

argument path: model[standard]/descriptor[se_a_tpe]/trainable

If the parameters in the embedding net is trainable

seed:

type: int | NoneType, optional

argument path: model[standard]/descriptor[se_a_tpe]/seed

Random seed for parameter initialization

exclude_types:

type: typing.List[typing.List[int]], optional, default: []

argument path:
model[standard]/descriptor[se_a_tpe]/exclude_types

The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

env_protection:

type: float, optional, default: 0.0

argument path:
model[standard]/descriptor[se_a_tpe]/env_protection

(Supported Backend: TensorFlow) Protection parameter to prevent division by zero errors during environment matrix calculations. For example, when using paddings, there may be zero distances of neighbors, which may make division by zero error during environment matrix calculations without protection.

set_davg_zero:

type: bool, optional, default: False

argument path:
model[standard]/descriptor[se_a_tpe]/set_davg_zero

Set the normalization average to zero. This option should be set when atom_ener in the energy fitting is used

type_nchanl:

type: int, optional, default: 4

argument path:
`model[standard]/descriptor[se_a_tpe]/type_nchanl`

number of channels for type embedding

type_nlayer:

type: `int`, optional, default: `2`

argument path:
`model[standard]/descriptor[se_a_tpe]/type_nlayer`

number of hidden layers of type embedding net

numb_aparam:

type: `int`, optional, default: `0`

argument path:
`model[standard]/descriptor[se_a_tpe]/numb_aparam`

dimension of atomic parameter. if set to a value > 0, the atomic parameters are embedded.

When type is set to `se_e2_r` (or its alias `se_r`):

sel:

type: `str` | `typing.List[int]`, optional, default: `auto`

argument path: `model[standard]/descriptor[se_e2_r]/sel`

This parameter set the number of selected neighbors for each type of atom. It can be:

- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. sel[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- str. Can be "auto:factor" or "auto". "factor" is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the "factor". Finally the number is wraped up to 4 divisible. The option "auto" is equivalent to "auto:1.1".

rcut:

type: `float`, optional, default: `6.0`

argument path: `model[standard]/descriptor[se_e2_r]/rcut`

The cut-off radius.

rcut_smth:

type: `float`, optional, default: `0.5`

argument path: `model[standard]/descriptor[se_e2_r]/rcut_smth`

Where to start smoothing. For example the 1/r term is smoothed from rcut to rcut_smth

neuron:

type: `typing.List[int]`, optional, default: `[10, 20, 40]`

argument path: `model[standard]/descriptor[se_e2_r]/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

activation_function:

type: `str`, optional, default: `tanh`

argument path:
`model[standard]/descriptor[se_e2_r]/activation_function`

The activation function in the embedding net. Supported activation functions are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”. Note that “gelu” denotes the custom operator version, and “gelu_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

resnet_dt:

type: `bool`, optional, default: `False`

argument path: `model[standard]/descriptor[se_e2_r]/resnet_dt`

Whether to use a “Timestep” in the skip connection

type_one_side:

type: `bool`, optional, default: `False`

argument path:
`model[standard]/descriptor[se_e2_r]/type_one_side`

If true, the embedding network parameters vary by types of neighbor atoms only, so there will be $N_text{types}$ sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be $N_text{types}^2$ sets of embedding network parameters.

precision:

type: `str`, optional, default: `default`

argument path: `model[standard]/descriptor[se_e2_r]/precision`

The precision of the embedding net parameters, supported options are “default”, “float64”, “float16”, “float32”. Default follows the interface precision.

trainable:

type: `bool`, optional, default: `True`

argument path: `model[standard]/descriptor[se_e2_r]/trainable`

If the parameters in the embedding net are trainable

seed:

type: `int` | `NoneType`, optional

argument path: `model[standard]/descriptor[se_e2_r]/seed`

Random seed for parameter initialization

exclude_types:

type: `typing.List[typing.List[int]]`, optional, default: `[]`

argument path:
`model[standard]/descriptor[se_e2_r]/exclude_types`

The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero:

type: bool, optional, default: False

argument path: model[standard]/descriptor[se_e2_r]/set_davg_zero

Set the normalization average to zero. This option should be set when atom_ener in the energy fitting is used

When type is set to hybrid:

list:

type: list

argument path: model[standard]/descriptor[hybrid]/list

A list of descriptor definitions

When type is set to se_atten (or its alias dpa1):

sel:

type: str | int | typing.List[int], optional, default: auto

argument path: model[standard]/descriptor[se_atten]/sel

This parameter set the number of selected neighbors. Note that this parameter is a little different from that in other descriptors. Instead of separating each type of atoms, only the summation matters. And this number is highly related with the efficiency, thus one should not make it too large. Usually 200 or less is enough, far away from the GPU limitation 4096. It can be:

- int. The maximum number of neighbor atoms to be considered. We recommend it to be less than 200.
- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. Only the summation of sel[i] matters, and it is recommended to be less than 200. - str. Can be "auto:factor" or "auto". "factor" is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the "factor". Finally the number is wraped up to 4 divisible. The option "auto" is equivalent to "auto:1.1".

rcut:

type: float, optional, default: 6.0

argument path: model[standard]/descriptor[se_atten]/rcut

The cut-off radius.

rcut_smth:

type: float, optional, default: 0.5

argument path: model[standard]/descriptor[se_atten]/rcut_smth

Where to start smoothing. For example the 1/r term is smoothed from rcut to rcut_smth

neuron:

type: typing.List[int], optional, default: [10, 20, 40]

argument path: model[standard]/descriptor[se_atten]/neuron

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

axis_neuron:

type: int, optional, default: 4, alias: n_axis_neuron

argument path:
model[standard]/descriptor[se_atten]/axis_neuron

Size of the submatrix of G (embedding matrix).

activation_function:

type: str, optional, default: tanh

argument path:
model[standard]/descriptor[se_atten]/activation_function

The activation function in the embedding net. Supported activation functions are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf". Note that "gelu" denotes the custom operator version, and "gelu_tf" denotes the TF standard version. If you set "None" or "none" here, no activation function will be used.

resnet_dt:

type: bool, optional, default: False

argument path: model[standard]/descriptor[se_atten]/resnet_dt

(Supported Backend: TensorFlow) Whether to use a "Timestep" in the skip connection

type_one_side:

type: bool, optional, default: False

argument path:
model[standard]/descriptor[se_atten]/type_one_side

(Supported Backend: TensorFlow) If 'False', type embeddings of both neighbor and central atoms are considered. If 'True', only type embeddings of neighbor atoms are considered. Default is 'False'.

precision:

type: str, optional, default: default

argument path: model[standard]/descriptor[se_atten]/precision

(Supported Backend: TensorFlow) The precision of the embedding net parameters, supported options are "default", "float64", "float16", "float32". Default follows the interface precision.

trainable:

type: bool, optional, default: True

argument path: model[standard]/descriptor[se_atten]/trainable

(Supported Backend: TensorFlow) If the parameters in the embedding net is trainable

seed:

type: int | NoneType, optional

argument path: model[standard]/descriptor[se_atten]/seed

Random seed for parameter initialization

exclude_types:

type: typing.List[typing.List[int]], optional, default: []

argument path: model[standard]/descriptor[se_atten]/exclude_types

(Supported Backend: TensorFlow) The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

attn:

type: int, optional, default: 128

argument path: model[standard]/descriptor[se_atten]/attn

The length of hidden vectors in attention layers

attn_layer:

type: int, optional, default: 2

argument path: model[standard]/descriptor[se_atten]/attn_layer

The number of attention layers. Note that model compression of se_atten is only enabled when attn_layer==0 and tebd_input_mode=='strip'

attn_dotr:

type: bool, optional, default: True

argument path: model[standard]/descriptor[se_atten]/attn_dotr

Whether to do dot product with the normalized relative coordinates

attn_mask:

type: bool, optional, default: False

argument path: model[standard]/descriptor[se_atten]/attn_mask

Whether to do mask on the diagonal in the attention matrix

stripped_type_embedding:

type: bool | NoneType, optional, default: None

argument path: model[standard]/descriptor[se_atten]/stripped_type_embedding

(Deprecated, kept only for compatibility.) Whether to strip the type embedding into a separate embedding network. Setting this parameter to True is equivalent to setting tebd_input_mode to 'strip'. Setting it to False is equivalent to setting tebd_input_mode to 'concat'.The default value is None, which means the tebd_input_mode setting will be used instead.

smooth_type_embedding:

type: bool, optional, default: False, alias: smooth_type_embdding

argument path: model[standard]/descriptor[se_atten]/smooth_type_embedding

Whether to use smooth process in attention weights calculation. (Supported Backend: TensorFlow) When using stripped type embedding, whether to dot smooth factor on the network output of type embedding to keep the network smooth, instead of setting set_davg_zero to be True.

set_davg_zero:

type: bool, optional, default: True

argument path:
model[standard]/descriptor[se_atten]/set_davg_zero

Set the normalization average to zero. This option should be set when se_atten descriptor or atom_ener in the energy fitting is used

trainable_ln:

type: bool, optional, default: True
argument path:
model[standard]/descriptor[se_atten]/trainable_ln

Whether to use trainable shift and scale weights in layer normalization.

ln_eps:

type: float | NoneType, optional, default: None
argument path: model[standard]/descriptor[se_atten]/ln_eps

The epsilon value for layer normalization. The default value for TensorFlow is set to 1e-3 to keep consistent with keras while set to 1e-5 in PyTorch and DP implementation.

tebd_dim:

type: int, optional, default: 8
argument path: model[standard]/descriptor[se_atten]/tebd_dim

(Supported Backend: PyTorch) The dimension of atom type embedding.

tebd_input_mode:

type: str, optional, default: concat
argument path:
model[standard]/descriptor[se_atten]/tebd_input_mode

The input mode of the type embedding. Supported modes are ['concat', 'strip'].- 'concat': Concatenate the type embedding with the smoothed radial information as the union input for the embedding network. When type_one_side is False, the input is input_ij = concat([r_ij, tebd_j, tebd_i]). When type_one_side is True, the input is input_ij = concat([r_ij, tebd_j]). The output is out_ij = embeding(input_ij) for the pair-wise representation of atom i with neighbor j.- 'strip': Use a separated embedding network for the type embedding and combine the output with the radial embedding network output. When type_one_side is False, the input is input_t = concat([tebd_j, tebd_i]). (Supported Backend: PyTorch) When type_one_side is True, the input is input_t = tebd_j. The output is out_ij = embeding_t(input_t) * embeding_s(r_ij) + embeding_s(r_ij) for the pair-wise representation of atom i with neighbor j.

scaling_factor:

type: float, optional, default: 1.0
argument path:
model[standard]/descriptor[se_atten]/scaling_factor

(Supported Backend: PyTorch) The scaling factor of normalization in calculations of attention weights, which is used to scale the matmul(Q, K). If temperature is None, the scaling of attention weights is (N_hidden_dim * scaling_factor)**0.5. Else, the scaling of attention weights is setting to temperature.

normalize:

type: bool, optional, default: True

argument path: model[standard]/descriptor[se_atten]/normalize

(Supported Backend: PyTorch) Whether to normalize the hidden vectors during attention calculation.

temperature:

type: float, optional

argument path: model[standard]/descriptor[se_atten]/temperature

(Supported Backend: PyTorch) The scaling factor of normalization in calculations of attention weights, which is used to scale the matmul(Q, K).

concat_output_tebd:

type: bool, optional, default: True

argument path: model[standard]/descriptor[se_atten]/concat_output_tebd

(Supported Backend: PyTorch) Whether to concat type embedding at the output of the descriptor.

When type is set to se_atten_v2:

(Supported Backend: TensorFlow)

sel:

type: str | int | typing.List[int], optional, default: auto

argument path: model[standard]/descriptor[se_atten_v2]/sel

This parameter set the number of selected neighbors. Note that this parameter is a little different from that in other descriptors. Instead of separating each type of atoms, only the summation matters. And this number is highly related with the efficiency, thus one should not make it too large. Usually 200 or less is enough, far away from the GPU limitation 4096. It can be:

- int. The maximum number of neighbor atoms to be considered. We recommend it to be less than 200.
- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. Only the summation of sel[i] matters, and it is recommended to be less than 200. - str. Can be "auto:factor" or "auto". "factor" is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the "factor". Finally the number is wraped up to 4 divisible. The option "auto" is equivalent to "auto:1.1".

rcut:

type: float, optional, default: 6.0

argument path: model[standard]/descriptor[se_atten_v2]/rcut

The cut-off radius.

rcut_smth:

type: float, optional, default: 0.5

argument path:
`model[standard]/descriptor[se_atten_v2]/rcut_smth`

Where to start smoothing. For example the 1/r term is smoothed from rcut to rcut_smth

neuron:

type: `typing.List[int]`, optional, default: `[10, 20, 40]`
argument path: `model[standard]/descriptor[se_atten_v2]/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

axis_neuron:

type: `int`, optional, default: `4`, alias: n_axis_neuron
argument path:
`model[standard]/descriptor[se_atten_v2]/axis_neuron`

Size of the submatrix of G (embedding matrix).

activation_function:

type: `str`, optional, default: `tanh`
argument path:
`model[standard]/descriptor[se_atten_v2]/activation_function`

The activation function in the embedding net. Supported activation functions are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”. Note that “gelu” denotes the custom operator version, and “gelu_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

resnet_dt:

type: `bool`, optional, default: `False`
argument path:
`model[standard]/descriptor[se_atten_v2]/resnet_dt`

(Supported Backend: TensorFlow) Whether to use a “Timestep” in the skip connection

type_one_side:

type: `bool`, optional, default: `False`
argument path:
`model[standard]/descriptor[se_atten_v2]/type_one_side`

(Supported Backend: TensorFlow) If ‘False’, type embeddings of both neighbor and central atoms are considered. If ‘True’, only type embeddings of neighbor atoms are considered. Default is ‘False’.

precision:

type: `str`, optional, default: `default`
argument path:
`model[standard]/descriptor[se_atten_v2]/precision`

(Supported Backend: TensorFlow) The precision of the embedding net parameters, supported options are “default”, “float64”, “float16”, “float32”. Default follows the interface precision.

trainable:

type: bool, optional, default: True

argument path: model[standard]/descriptor[se_atten_v2]/trainable

(Supported Backend: TensorFlow) If the parameters in the embedding net is trainable

seed:

type: int | NoneType, optional

argument path: model[standard]/descriptor[se_atten_v2]/seed

Random seed for parameter initialization

exclude_types:

type: typing.List[typing.List[int]], optional, default: []

argument path: model[standard]/descriptor[se_atten_v2]/exclude_types

(Supported Backend: TensorFlow) The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

attn:

type: int, optional, default: 128

argument path: model[standard]/descriptor[se_atten_v2]/attn

The length of hidden vectors in attention layers

attn_layer:

type: int, optional, default: 2

argument path: model[standard]/descriptor[se_atten_v2]/attn_layer

The number of attention layers. Note that model compression of se_atten is only enabled when attn_layer==0 and tebd_input_mode=='strip'

attn_dotr:

type: bool, optional, default: True

argument path: model[standard]/descriptor[se_atten_v2]/attn_dotr

Whether to do dot product with the normalized relative coordinates

attn_mask:

type: bool, optional, default: False

argument path: model[standard]/descriptor[se_atten_v2]/attn_mask

Whether to do mask on the diagonal in the attention matrix

set_davg_zero:

type: bool, optional, default: False

argument path: model[standard]/descriptor[se_atten_v2]/set_davg_zero

Set the normalization average to zero. This option should be set when se_atten descriptor or atom_ener in the energy fitting is used

When type is set to dpa2:

(Supported Backend: PyTorch)

repinit_rcut:

type: float

argument path: model[standard]/descriptor[dpa2]/repinit_rcut

The cut-off radius of the repinit block

repinit_rcut_smth:

type: float

argument path: model[standard]/descriptor[dpa2]/repinit_rcut_smth

From this position the inverse distance smoothly decays to 0 at the cut-off. Use in the repinit block.

repinit_nsel:

type: int

argument path: model[standard]/descriptor[dpa2]/repinit_nsel

Maximally possible number of neighbors for repinit block.

repformer_rcut:

type: float

argument path: model[standard]/descriptor[dpa2]/repformer_rcut

The cut-off radius of the repformer block

repformer_rcut_smth:

type: float

argument path: model[standard]/descriptor[dpa2]/repformer_rcut_smth

From this position the inverse distance smoothly decays to 0 at the cut-off. Use in the repformer block.

repformer_nsel:

type: int

argument path: model[standard]/descriptor[dpa2]/repformer_nsel

Maximally possible number of neighbors for repformer block.

tebd_dim:

type: int, optional, default: 8

argument path: model[standard]/descriptor[dpa2]/tebd_dim

The dimension of atom type embedding

concat_output_tebd:

type: bool, optional, default: True

argument path: model[standard]/descriptor[dpa2]/concat_output_tebd

Whether to concat type embedding at the output of the descriptor.

repinit_neuron:

type: list, optional, default: [25, 50, 100]
argument path:
model[standard]/descriptor[dpa2]/repinit_neuron

repinit block: the number of neurons in the embedding net.

repinit_axis_neuron:

type: int, optional, default: 16
argument path:
model[standard]/descriptor[dpa2]/repinit_axis_neuron

repinit block: the number of dimension of split in the symmetrization op.

repinit_set_davg_zero:

type: bool, optional, default: True
argument path:
model[standard]/descriptor[dpa2]/repinit_set_davg_zero

repinit_activation:

type: str, optional, default: tanh
argument path:
model[standard]/descriptor[dpa2]/repinit_activation

repinit block: the activation function in the embedding net

repformer_nlayers:

type: int, optional, default: 3
argument path:
model[standard]/descriptor[dpa2]/repformer_nlayers

repformers block: the number of repformer layers

repformer_g1_dim:

type: int, optional, default: 128
argument path:
model[standard]/descriptor[dpa2]/repformer_g1_dim

repformers block: the dimension of single-atom rep

repformer_g2_dim:

type: int, optional, default: 16
argument path:
model[standard]/descriptor[dpa2]/repformer_g2_dim

repformers block: the dimension of invariant pair-atom rep

repformer_axis_dim:

type: int, optional, default: 4
argument path:
model[standard]/descriptor[dpa2]/repformer_axis_dim

repformers block: the number of dimension of split in the symmetrization ops.

repformer_do_bn_mode:

type: str, optional, default: no

argument path:
`model[standard]/descriptor[dpa2]/repformer_do_bn_mode`

repformers block: do batch norm in the repformer layers

repformer_bn_momentum:

type: `float`, optional, default: `0.1`
argument path:
`model[standard]/descriptor[dpa2]/repformer_bn_momentum`

repformers block: moment in the batch normalization

repformer_update_g1_has_conv:

type: `bool`, optional, default: `True`
argument path: `model[standard]/descriptor[dpa2]/repformer_update_g1_has_conv`

repformers block: update the g1 rep with convolution term

repformer_update_g1_has_drrd:

type: `bool`, optional, default: `True`
argument path: `model[standard]/descriptor[dpa2]/repformer_update_g1_has_drrd`

repformers block: update the g1 rep with the drrd term

repformer_update_g1_has_grrg:

type: `bool`, optional, default: `True`
argument path: `model[standard]/descriptor[dpa2]/repformer_update_g1_has_grrg`

repformers block: update the g1 rep with the grrg term

repformer_update_g1_has_attn:

type: `bool`, optional, default: `True`
argument path: `model[standard]/descriptor[dpa2]/repformer_update_g1_has_attn`

repformers block: update the g1 rep with the localized self-attention

repformer_update_g2_has_g1g1:

type: `bool`, optional, default: `True`
argument path: `model[standard]/descriptor[dpa2]/repformer_update_g2_has_g1g1`

repformers block: update the g2 rep with the g1xg1 term

repformer_update_g2_has_attn:

type: `bool`, optional, default: `True`
argument path: `model[standard]/descriptor[dpa2]/repformer_update_g2_has_attn`

repformers block: update the g2 rep with the gated self-attention

repformer_update_h2:

type: `bool`, optional, default: `False`
argument path:
`model[standard]/descriptor[dpa2]/repformer_update_h2`

repformers block: update the h2 rep

repformer_attn1_hidden:

type: int, optional, default: 64

argument path: model[standard]/descriptor[dpa2]/repformer_attn1_hidden

repformers block: the hidden dimension of localized self-attention

repformer_attn1_nhead:

type: int, optional, default: 4

argument path: model[standard]/descriptor[dpa2]/repformer_attn1_nhead

repformers block: the number of heads in localized self-attention

repformer_attn2_hidden:

type: int, optional, default: 16

argument path: model[standard]/descriptor[dpa2]/repformer_attn2_hidden

repformers block: the hidden dimension of gated self-attention

repformer_attn2_nhead:

type: int, optional, default: 4

argument path: model[standard]/descriptor[dpa2]/repformer_attn2_nhead

repformers block: the number of heads in gated self-attention

repformer_attn2_has_gate:

type: bool, optional, default: False

argument path: model[standard]/descriptor[dpa2]/repformer_attn2_has_gate

repformers block: has gate in the gated self-attention

repformer_activation:

type: str, optional, default: tanh

argument path: model[standard]/descriptor[dpa2]/repformer_activation

repformers block: the activation function in the MLPs.

repformer_update_style:

type: str, optional, default: res_avg

argument path: model[standard]/descriptor[dpa2]/repformer_update_style

repformers block: style of update a rep. can be res_avg or res_incr. res_avg updates a rep u with: u = 1/sqrt{n+1} (u + u_1 + u_2 + … + u_n) res_incr updates a rep u with: u = u + 1/sqrt{n} (u_1 + u_2 + … + u_n)

repformer_set_davg_zero:

type: bool, optional, default: True

argument path: model[standard]/descriptor[dpa2]/repformer_set_davg_zero

repformers block: set the avg to zero in statistics

repformer_add_type_ebd_to_seq:

type: bool, optional, default: False

argument path: model[standard]/descriptor[dpa2]/repformer_add_type_ebd_to_seq

repformers block: concatenate the type embedding at the output

When type is set to se_a_ebd_v2 (or its alias se_a_tpe_v2):

(Supported Backend: TensorFlow)

sel:

type: str | typing.List[int], optional, default: auto

argument path: model[standard]/descriptor[se_a_ebd_v2]/sel

This parameter set the number of selected neighbors for each type of atom. It can be:

- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. sel[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- str. Can be "auto:factor" or "auto". "factor" is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the "factor". Finally the number is wraped up to 4 divisible. The option "auto" is equivalent to "auto:1.1".

rcut:

type: float, optional, default: 6.0

argument path: model[standard]/descriptor[se_a_ebd_v2]/rcut

The cut-off radius.

rcut_smth:

type: float, optional, default: 0.5

argument path: model[standard]/descriptor[se_a_ebd_v2]/rcut_smth

Where to start smoothing. For example the 1/r term is smoothed from rcut to rcut_smth

neuron:

type: typing.List[int], optional, default: [10, 20, 40]

argument path: model[standard]/descriptor[se_a_ebd_v2]/neuron

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

axis_neuron:

type: int, optional, default: 4, alias: n_axis_neuron

argument path: model[standard]/descriptor[se_a_ebd_v2]/axis_neuron

Size of the submatrix of G (embedding matrix).

activation_function:

type: str, optional, default: tanh

argument path:
model[standard]/descriptor[se_a_ebd_v2]/activation_function

The activation function in the embedding net. Supported activation functions are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”. Note that “gelu” denotes the custom operator version, and “gelu_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

resnet_dt:

type: bool, optional, default: False

argument path:
model[standard]/descriptor[se_a_ebd_v2]/resnet_dt

Whether to use a “Timestep” in the skip connection

type_one_side:

type: bool, optional, default: False

argument path:
model[standard]/descriptor[se_a_ebd_v2]/type_one_side

If true, the embedding network parameters vary by types of neighbor atoms only, so there will be $N_text{types}$ sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be $N_text{types}^2$ sets of embedding network parameters.

precision:

type: str, optional, default: default

argument path:
model[standard]/descriptor[se_a_ebd_v2]/precision

The precision of the embedding net parameters, supported options are “default”, “float64”, “float16”, “float32”. Default follows the interface precision.

trainable:

type: bool, optional, default: True

argument path:
model[standard]/descriptor[se_a_ebd_v2]/trainable

If the parameters in the embedding net is trainable

seed:

type: int | NoneType, optional

argument path: model[standard]/descriptor[se_a_ebd_v2]/seed

Random seed for parameter initialization

exclude_types:

type: typing.List[typing.List[int]], optional, default: []

argument path:
model[standard]/descriptor[se_a_ebd_v2]/exclude_types

The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

env_protection:

type: `float`, optional, default: `0.0`

argument path:
`model[standard]/descriptor[se_a_ebd_v2]/env_protection`

(Supported Backend: TensorFlow) Protection parameter to prevent division by zero errors during environment matrix calculations. For example, when using paddings, there may be zero distances of neighbors, which may make division by zero error during environment matrix calculations without protection.

set_davg_zero:

type: `bool`, optional, default: `False`

argument path:
`model[standard]/descriptor[se_a_ebd_v2]/set_davg_zero`

Set the normalization average to zero. This option should be set when atom_ener in the energy fitting is used

When type is set to `se_a_mask`:

(Supported Backend: TensorFlow)

sel:

type: `str` | `typing.List[int]`, optional, default: `auto`

argument path: `model[standard]/descriptor[se_a_mask]/sel`

This parameter sets the number of selected neighbors for each type of atom. It can be:

- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. sel[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- str. Can be "auto:factor" or "auto". "factor" is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the "factor". Finally the number is wraped up to 4 divisible. The option "auto" is equivalent to "auto:1.1".

neuron:

type: `typing.List[int]`, optional, default: `[10, 20, 40]`

argument path: `model[standard]/descriptor[se_a_mask]/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

axis_neuron:

type: `int`, optional, default: `4`, alias: n_axis_neuron

argument path:
`model[standard]/descriptor[se_a_mask]/axis_neuron`

Size of the submatrix of G (embedding matrix).

activation_function:

type: `str`, optional, default: `tanh`

argument path:
`model[standard]/descriptor[se_a_mask]/activation_function`

The activation function in the embedding net. Supported activation functions are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”. Note that “gelu” denotes the custom operator version, and “gelu_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

resnet_dt:

type: `bool`, optional, default: `False`

argument path:
`model[standard]/descriptor[se_a_mask]/resnet_dt`

Whether to use a “Timestep” in the skip connection

type_one_side:

type: `bool`, optional, default: `False`

argument path:
`model[standard]/descriptor[se_a_mask]/type_one_side`

If true, the embedding network parameters vary by types of neighbor atoms only, so there will be $N_text{types}$ sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be $N_text{types}^2$ sets of embedding network parameters.

exclude_types:

type: `typing.List[typing.List[int]]`, optional, default: `[]`

argument path:
`model[standard]/descriptor[se_a_mask]/exclude_types`

The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

precision:

type: `str`, optional, default: `default`

argument path:
`model[standard]/descriptor[se_a_mask]/precision`

The precision of the embedding net parameters, supported options are “default”, “float64”, “float16”, “float32”. Default follows the interface precision.

trainable:

type: `bool`, optional, default: `True`

argument path:
`model[standard]/descriptor[se_a_mask]/trainable`

If the parameters in the embedding net is trainable

seed:

type: `int` | `NoneType`, optional

argument path: `model[standard]/descriptor[se_a_mask]/seed`

Random seed for parameter initialization

fitting_net:

type: dict

argument path: model[standard]/fitting_net

The fitting of physical properties.

Depending on the value of type, different sub args are accepted.

type:

type: str (flag key), default: ener

argument path: model[standard]/fitting_net/type

possible choices: *ener*, *dos*, *polar*, *dipole*

The type of the fitting. See explanation below.
- ener: Fit an energy model (potential energy surface).
- dos : Fit a density of states model. The total density of states / site-projected density of states labels should be provided by dos.npy or atom_dos.npy in each data system. The file has number of frames lines and number of energy grid columns (times number of atoms in atom_dos.npy). See loss parameter.
- dipole: Fit an atomic dipole model. Global dipole labels or atomic dipole labels for all the selected atoms (see sel_type) should be provided by dipole.npy in each data system. The file either has number of frames lines and 3 times of number of selected atoms columns, or has number of frames lines and 3 columns. See loss parameter.
- polar: Fit an atomic polarizability model. Global polarizazbility labels or atomic polarizability labels for all the selected atoms (see sel_type) should be provided by polarizability.npy in each data system. The file eith has number of frames lines and 9 times of number of selected atoms columns, or has number of frames lines and 9 columns. See loss parameter.

When type is set to ener:

numb_fparam:

type: int, optional, default: 0

argument path: model[standard]/fitting_net[ener]/numb_fparam

The dimension of the frame parameter. If set to >0, file fparam.npy should be included to provided the input fparams.

numb_aparam:

type: int, optional, default: 0

argument path: model[standard]/fitting_net[ener]/numb_aparam

The dimension of the atomic parameter. If set to >0, file aparam.npy should be included to provided the input aparams.

neuron:

type: typing.List[int], optional, default: [120, 120, 120], alias: n_neuron

argument path: model[standard]/fitting_net[ener]/neuron

The number of neurons in each hidden layers of the fitting net. When two hidden layers are of the same size, a skip connection is built.

activation_function:

type: str, optional, default: tanh

argument path:
`model[standard]/fitting_net[ener]/activation_function`

The activation function in the fitting net. Supported activation functions are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”. Note that “gelu” denotes the custom operator version, and “gelu_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

precision:

type: `str`, optional, default: `default`

argument path: `model[standard]/fitting_net[ener]/precision`

The precision of the fitting net parameters, supported options are “default”, “float64”, “float16”, “float32”. Default follows the interface precision.

resnet_dt:

type: `bool`, optional, default: `True`

argument path: `model[standard]/fitting_net[ener]/resnet_dt`

Whether to use a “Timestep” in the skip connection

trainable:

type: `typing.List[bool]` | `bool`, optional, default: `True`

argument path: `model[standard]/fitting_net[ener]/trainable`

Whether the parameters in the fitting net are trainable. This option can be

- bool: True if all parameters of the fitting net are trainable, False otherwise.
- list of bool(Supported Backend: TensorFlow) : Specifies if each layer is trainable. Since the fitting net is composed by hidden layers followed by a output layer, the length of this list should be equal to len(neuron)+1.

rcond:

type: `float` | `NoneType`, optional, default: `None`

argument path: `model[standard]/fitting_net[ener]/rcond`

The condition number used to determine the inital energy shift for each type of atoms. See rcond in `numpy.linalg.lstsq()` for more details.

seed:

type: `int` | `NoneType`, optional

argument path: `model[standard]/fitting_net[ener]/seed`

Random seed for parameter initialization of the fitting net

atom_ener:

type: `typing.List[typing.Optional[float]]`, optional, default: `[]`

argument path: `model[standard]/fitting_net[ener]/atom_ener`

Specify the atomic energy in vacuum for each type

layer_name:

type: `typing.List[str]`, optional

argument path: `model[standard]/fitting_net[ener]/layer_name`

The name of the each layer. The length of this list should be equal to n_neuron + 1. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters. The shape of these layers should be the same. If null is given for a layer, parameters will not be shared.

use_aparam_as_mask:

type: `bool`, optional, default: `False`

argument path: `model[standard]/fitting_net[ener]/use_aparam_as_mask`

Whether to use the aparam as a mask in input.If True, the aparam will not be used in fitting net for embedding.When descrpt is se_a_mask, the aparam will be used as a mask to indicate the input atom is real/virtual. And use_aparam_as_mask should be set to True.

When type is set to `dos`:

(Supported Backend: TensorFlow)

numb_fparam:

type: `int`, optional, default: `0`

argument path: `model[standard]/fitting_net[dos]/numb_fparam`

The dimension of the frame parameter. If set to >0, file fparam.npy should be included to provided the input fparams.

numb_aparam:

type: `int`, optional, default: `0`

argument path: `model[standard]/fitting_net[dos]/numb_aparam`

The dimension of the atomic parameter. If set to >0, file aparam.npy should be included to provided the input aparams.

neuron:

type: `typing.List[int]`, optional, default: `[120, 120, 120]`

argument path: `model[standard]/fitting_net[dos]/neuron`

The number of neurons in each hidden layers of the fitting net. When two hidden layers are of the same size, a skip connection is built.

activation_function:

type: `str`, optional, default: `tanh`

argument path: `model[standard]/fitting_net[dos]/activation_function`

The activation function in the fitting net. Supported activation functions are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”. Note that “gelu” denotes the custom operator version, and “gelu_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

precision:

type: `str`, optional, default: `float64`

argument path: `model[standard]/fitting_net[dos]/precision`

The precision of the fitting net parameters, supported options are “default”, “float64”, “float16”, “float32”. Default follows the interface precision.

resnet_dt:

type: bool, optional, default: True

argument path: model[standard]/fitting_net[dos]/resnet_dt

Whether to use a "Timestep" in the skip connection

trainable:

type: typing.List[bool] | bool, optional, default: True

argument path: model[standard]/fitting_net[dos]/trainable

Whether the parameters in the fitting net are trainable. This option can be

- bool: True if all parameters of the fitting net are trainable, False otherwise.
- list of bool: Specifies if each layer is trainable. Since the fitting net is composed by hidden layers followed by a output layer, the length of tihs list should be equal to len(neuron)+1.

rcond:

type: float | NoneType, optional, default: None

argument path: model[standard]/fitting_net[dos]/rcond

The condition number used to determine the inital energy shift for each type of atoms. See rcond in numpy.linalg.lstsq() for more details.

seed:

type: int | NoneType, optional

argument path: model[standard]/fitting_net[dos]/seed

Random seed for parameter initialization of the fitting net

numb_dos:

type: int, optional, default: 300

argument path: model[standard]/fitting_net[dos]/numb_dos

The number of gridpoints on which the DOS is evaluated (NEDOS in VASP)

When type is set to polar:

neuron:

type: typing.List[int], optional, default: [120, 120, 120], alias: n_neuron

argument path: model[standard]/fitting_net[polar]/neuron

The number of neurons in each hidden layers of the fitting net. When two hidden layers are of the same size, a skip connection is built.

activation_function:

type: str, optional, default: tanh

argument path: model[standard]/fitting_net[polar]/activation_function

The activation function in the fitting net. Supported activation functions are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf". Note that "gelu" denotes the custom operator version, and "gelu_tf" denotes the TF standard version. If you set "None" or "none" here, no activation function will be used.

resnet_dt:

type: bool, optional, default: True

argument path: model[standard]/fitting_net[polar]/resnet_dt

Whether to use a "Timestep" in the skip connection

precision:

type: str, optional, default: default

argument path: model[standard]/fitting_net[polar]/precision

The precision of the fitting net parameters, supported options are "default", "float64", "float16", "float32". Default follows the interface precision.

fit_diag:

type: bool, optional, default: True

argument path: model[standard]/fitting_net[polar]/fit_diag

Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.

scale:

type: typing.List[float] | float, optional, default: 1.0

argument path: model[standard]/fitting_net[polar]/scale

The output of the fitting net (polarizability matrix) will be scaled by scale

shift_diag:

type: bool, optional, default: True

argument path: model[standard]/fitting_net[polar]/shift_diag

Whether to shift the diagonal of polar, which is beneficial to training. Default is true.

sel_type:

type: int | typing.List[int] | NoneType, optional, alias: pol_type

argument path: model[standard]/fitting_net[polar]/sel_type

The atom types for which the atomic polarizability will be provided. If not set, all types will be selected.(Supported Backend: TensorFlow)

seed:

type: int | NoneType, optional

argument path: model[standard]/fitting_net[polar]/seed

Random seed for parameter initialization of the fitting net

When type is set to dipole:

neuron:

type: typing.List[int], optional, default: [120, 120, 120], alias: n_neuron

argument path: model[standard]/fitting_net[dipole]/neuron

The number of neurons in each hidden layers of the fitting net. When two hidden layers are of the same size, a skip connection is built.

activation_function:

type: str, optional, default: tanh

argument path: model[standard]/fitting_net[dipole]/activation_function

The activation function in the fitting net. Supported activation functions are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf". Note that "gelu" denotes the custom operator version, and "gelu_tf" denotes the TF standard version. If you set "None" or "none" here, no activation function will be used.

resnet_dt:

type: bool, optional, default: True

argument path: model[standard]/fitting_net[dipole]/resnet_dt

Whether to use a "Timestep" in the skip connection

precision:

type: str, optional, default: default

argument path: model[standard]/fitting_net[dipole]/precision

The precision of the fitting net parameters, supported options are "default", "float64", "float16", "float32". Default follows the interface precision.

sel_type:

type: int | typing.List[int] | NoneType, optional, alias: dipole_type

argument path: model[standard]/fitting_net[dipole]/sel_type

The atom types for which the atomic dipole will be provided. If not set, all types will be selected.(Supported Backend: TensorFlow)

seed:

type: int | NoneType, optional

argument path: model[standard]/fitting_net[dipole]/seed

Random seed for parameter initialization of the fitting net

When type is set to multi:

(Supported Backend: TensorFlow) Multiple-task model.

descriptor:

type: dict

argument path: model[multi]/descriptor

The descriptor of atomic environment. See model[standard]/descriptor for details.

fitting_net_dict:

type: dict

argument path: model[multi]/fitting_net_dict

The dictionary of multiple fitting nets in multi-task mode. Each fitting_net_dict[fitting_key] is the single definition of fitting of physical properties with user-defined name fitting_key.

When type is set to frozen:

model_file:

type: str
argument path: model[frozen]/model_file

Path to the frozen model file.

When type is set to pairtab:

(Supported Backend: TensorFlow) Pairwise tabulation energy model.

tab_file:

type: str
argument path: model[pairtab]/tab_file

Path to the tabulation file.

rcut:

type: float
argument path: model[pairtab]/rcut

The cut-off radius.

sel:

type: str | int | typing.List[int]
argument path: model[pairtab]/sel

This parameter set the number of selected neighbors. Note that this parameter is a little different from that in other descriptors. Instead of separating each type of atoms, only the summation matters. And this number is highly related with the efficiency, thus one should not make it too large. Usually 200 or less is enough, far away from the GPU limitation 4096. It can be:

- int. The maximum number of neighbor atoms to be considered. We recommend it to be less than 200.
- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. Only the summation of sel[i] matters, and it is recommended to be less than 200. - str. Can be "auto:factor" or "auto". "factor" is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the "factor". Finally the number is wraped up to 4 divisible. The option "auto" is equivalent to "auto:1.1".

When type is set to pairwise_dprc:

(Supported Backend: TensorFlow)

qm_model:

type: dict
argument path: model[pairwise_dprc]/qm_model

qmmm_model:

type: dict
argument path: model[pairwise_dprc]/qmmm_model

When type is set to linear_ener:

(Supported Backend: TensorFlow)

models:

type: dict | list
argument path: model[linear_ener]/models

The sub-models.

weights:

type: list | str
argument path: model[linear_ener]/weights

If the type is list of float, a list of weights for each model. If "mean", the weights are set to be 1 / len(models). If "sum", the weights are set to be 1.

learning_rate:

type: dict, optional
argument path: learning_rate

The definitio of learning rate

scale_by_worker:

type: str, optional, default: linear
argument path: learning_rate/scale_by_worker

When parallel training or batch size scaled, how to alter learning rate. Valid values are linear`(default), `sqrt or none.

Depending on the value of type, different sub args are accepted.

type:

type: str (flag key), default: exp
argument path: learning_rate/type
possible choices: *exp*

The type of the learning rate.

When type is set to exp:

start_lr:

type: float, optional, default: 0.001
argument path: learning_rate[exp]/start_lr

The learning rate at the start of the training.

stop_lr:

type: float, optional, default: 1e-08
argument path: learning_rate[exp]/stop_lr

The desired learning rate at the end of the training. When decay_rate (Supported Backend: PyTorch) is explicitly set, this value will serve as the minimum learning rate during training. In other words, if the learning rate decays below stop_lr, stop_lr will be applied instead.

decay_steps:

type: int, optional, default: 5000
argument path: learning_rate[exp]/decay_steps

The learning rate is decaying every this number of training steps.

decay_rate:

type: `float` | `NoneType`, optional, default: `None`

argument path: `learning_rate[exp]/decay_rate`

(Supported Backend: PyTorch) The decay rate for the learning rate. If this is provided, it will be used directly as the decay rate for learning rate instead of calculating it through interpolation between start_lr and stop_lr.

learning_rate_dict:

type: `dict`, optional

argument path: `learning_rate_dict`

The dictionary of definitions of learning rates in multi-task mode. Each learning_rate_dict[fitting_key], with user-defined name fitting_key in model/fitting_net_dict, is the single definition of learning rate.

loss:

type: `dict`, optional

argument path: `loss`

The definition of loss function. The loss type should be set to tensor, ener or left unset.

Depending on the value of type, different sub args are accepted.

type:

type: `str` (flag key), default: `ener`

argument path: `loss/type`

possible choices: *`ener`*, *`ener_spin`*, *`dos`*, *`tensor`*

The type of the loss. When the fitting type is ener, the loss type should be set to ener or left unset. When the fitting type is dipole or polar, the loss type should be set to tensor.

When type is set to `ener`:

start_pref_e:

type: `int` | `float`, optional, default: `0.02`

argument path: `loss[ener]/start_pref_e`

The prefactor of energy loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the energy label should be provided by file energy.npy in each data system. If both start_pref_e and limit_pref_e are set to 0, then the energy will be ignored.

limit_pref_e:

type: `int` | `float`, optional, default: `1.0`

argument path: `loss[ener]/limit_pref_e`

The prefactor of energy loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

start_pref_f:

type: `int` | `float`, optional, default: `1000`

argument path: `loss[ener]/start_pref_f`

The prefactor of force loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the force label should be provided by file

force.npy in each data system. If both start_pref_f and limit_pref_f are set to 0, then the force will be ignored.

limit_pref_f:

type: `int` | `float`, optional, default: `1.0`

argument path: `loss[ener]/limit_pref_f`

The prefactor of force loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

start_pref_v:

type: `int` | `float`, optional, default: `0.0`

argument path: `loss[ener]/start_pref_v`

The prefactor of virial loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the virial label should be provided by file virial.npy in each data system. If both start_pref_v and limit_pref_v are set to 0, then the virial will be ignored.

limit_pref_v:

type: `int` | `float`, optional, default: `0.0`

argument path: `loss[ener]/limit_pref_v`

The prefactor of virial loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

start_pref_ae:

type: `int` | `float`, optional, default: `0.0`

argument path: `loss[ener]/start_pref_ae`

The prefactor of atomic energy loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the atom_ener label should be provided by file atom_ener.npy in each data system. If both start_pref_ae and limit_pref_ae are set to 0, then the atomic energy will be ignored.

limit_pref_ae:

type: `int` | `float`, optional, default: `0.0`

argument path: `loss[ener]/limit_pref_ae`

The prefactor of atomic energy loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

start_pref_pf:

type: `int` | `float`, optional, default: `0.0`

argument path: `loss[ener]/start_pref_pf`

The prefactor of atomic prefactor force loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the atom_pref label should be provided by file atom_pref.npy in each data system. If both start_pref_pf and limit_pref_pf are set to 0, then the atomic prefactor force will be ignored.

limit_pref_pf:

type: `int` | `float`, optional, default: `0.0`

argument path: `loss[ener]/limit_pref_pf`

The prefactor of atomic prefactor force loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

relative_f:

type: float | NoneType, optional

argument path: loss[ener]/relative_f

If provided, relative force error will be used in the loss. The difference of force will be normalized by the magnitude of the force in the label with a shift given by relative_f, i.e. DF_i / (|| F || + relative_f) with DF denoting the difference between prediction and label and || F || denoting the L2 norm of the label.

enable_atom_ener_coeff:

type: bool, optional, default: False

argument path: loss[ener]/enable_atom_ener_coeff

If true, the energy will be computed as sum_i c_i E_i. c_i should be provided by file atom_ener_coeff.npy in each data system, otherwise it's 1.

start_pref_gf:

type: float, optional, default: 0.0

argument path: loss[ener]/start_pref_gf

The prefactor of generalized force loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the drdq label should be provided by file drdq.npy in each data system. If both start_pref_gf and limit_pref_gf are set to 0, then the generalized force will be ignored.

limit_pref_gf:

type: float, optional, default: 0.0

argument path: loss[ener]/limit_pref_gf

The prefactor of generalized force loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

numb_generalized_coord:

type: int, optional, default: 0

argument path: loss[ener]/numb_generalized_coord

The dimension of generalized coordinates. Required when generalized force loss is used.

When type is set to **ener_spin**:

start_pref_e:

type: int | float, optional, default: 0.02

argument path: loss[ener_spin]/start_pref_e

The prefactor of energy loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the energy label should be provided by file energy.npy in each data system. If both start_pref_energy and limit_pref_energy are set to 0, then the energy will be ignored.

limit_pref_e:

type: int | float, optional, default: 1.0

argument path: loss[ener_spin]/limit_pref_e

The prefactor of energy loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

start_pref_fr:

type: int | float, optional, default: 1000

argument path: loss[ener_spin]/start_pref_fr

The prefactor of force_real_atom loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the force_real_atom label should be provided by file force_real_atom.npy in each data system. If both start_pref_force_real_atom and limit_pref_force_real_atom are set to 0, then the force_real_atom will be ignored.

limit_pref_fr:

type: int | float, optional, default: 1.0

argument path: loss[ener_spin]/limit_pref_fr

The prefactor of force_real_atom loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

start_pref_fm:

type: int | float, optional, default: 10000

argument path: loss[ener_spin]/start_pref_fm

The prefactor of force_magnetic loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the force_magnetic label should be provided by file force_magnetic.npy in each data system. If both start_pref_force_magnetic and limit_pref_force_magnetic are set to 0, then the force_magnetic will be ignored.

limit_pref_fm:

type: int | float, optional, default: 10.0

argument path: loss[ener_spin]/limit_pref_fm

The prefactor of force_magnetic loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

start_pref_v:

type: int | float, optional, default: 0.0

argument path: loss[ener_spin]/start_pref_v

The prefactor of virial loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the virial label should be provided by file virial.npy in each data system. If both start_pref_virial and limit_pref_virial are set to 0, then the virial will be ignored.

limit_pref_v:

type: int | float, optional, default: 0.0

argument path: loss[ener_spin]/limit_pref_v

The prefactor of virial loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

start_pref_ae:

type: int | float, optional, default: 0.0

argument path: loss[ener_spin]/start_pref_ae

The prefactor of atom_ener loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the atom_ener label should be provided by file atom_ener.npy in each data system. If both start_pref_atom_ener and limit_pref_atom_ener are set to 0, then the atom_ener will be ignored.

limit_pref_ae:

type: int | float, optional, default: 0.0

argument path: loss[ener_spin]/limit_pref_ae

The prefactor of atom_ener loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

start_pref_pf:

type: int | float, optional, default: 0.0

argument path: loss[ener_spin]/start_pref_pf

The prefactor of atom_pref loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the atom_pref label should be provided by file atom_pref.npy in each data system. If both start_pref_atom_pref and limit_pref_atom_pref are set to 0, then the atom_pref will be ignored.

limit_pref_pf:

type: int | float, optional, default: 0.0

argument path: loss[ener_spin]/limit_pref_pf

The prefactor of atom_pref loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

relative_f:

type: float | NoneType, optional

argument path: loss[ener_spin]/relative_f

If provided, relative force error will be used in the loss. The difference of force will be normalized by the magnitude of the force in the label with a shift given by relative_f, i.e. DF_i / (|| F || + relative_f) with DF denoting the difference between prediction and label and || F || denoting the L2 norm of the label.

enable_atom_ener_coeff:

type: bool, optional, default: False

argument path: loss[ener_spin]/enable_atom_ener_coeff

If true, the energy will be computed as sum_i c_i E_i. c_i should be provided by file atom_ener_coeff.npy in each data system, otherwise it's 1.

When type is set to dos:

(Supported Backend: TensorFlow)

start_pref_dos:

type: int | float, optional, default: 0.0

argument path: loss[dos]/start_pref_dos

The prefactor of Density of State (DOS) loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the Density of State (DOS) label should be provided by file Density of State (DOS).npy in each data system. If both start_pref_Density of State (DOS) and limit_pref_Density of State (DOS) are set to 0, then the Density of State (DOS) will be ignored.

limit_pref_dos:

type: int | float, optional, default: 0.0

argument path: loss[dos]/limit_pref_dos

The prefactor of Density of State (DOS) loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

start_pref_cdf:

type: int | float, optional, default: 0.0

argument path: loss[dos]/start_pref_cdf

The prefactor of Cumulative Distribution Function (cumulative intergral of DOS) loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the Cumulative Distribution Function (cumulative intergral of DOS) label should be provided by file Cumulative Distribution Function (cumulative intergral of DOS).npy in each data system. If both start_pref_Cumulative Distribution Function (cumulative intergral of DOS) and limit_pref_Cumulative Distribution Function (cumulative intergral of DOS) are set to 0, then the Cumulative Distribution Function (cumulative intergral of DOS) will be ignored.

limit_pref_cdf:

type: int | float, optional, default: 0.0

argument path: loss[dos]/limit_pref_cdf

The prefactor of Cumulative Distribution Function (cumulative intergral of DOS) loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

start_pref_ados:

type: int | float, optional, default: 1.0

argument path: loss[dos]/start_pref_ados

The prefactor of atomic DOS (site-projected DOS) loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the atomic DOS (site-projected DOS) label should be provided by file atomic DOS (site-projected DOS).npy in each data system. If both start_pref_atomic DOS (site-projected DOS) and limit_pref_atomic DOS (site-projected DOS) are set to 0, then the atomic DOS (site-projected DOS) will be ignored.

limit_pref_ados:

type: int | float, optional, default: 1.0

argument path: loss[dos]/limit_pref_ados

The prefactor of atomic DOS (site-projected DOS) loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

start_pref_acdf:

type: int | float, optional, default: 0.0

argument path: loss[dos]/start_pref_acdf

The prefactor of Cumulative integral of atomic DOS loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the Cumulative integral of atomic DOS label should be provided by file Cumulative integral of atomic DOS.npy in each data system. If both start_pref_Cumulative integral of atomic DOS and limit_pref_Cumulative integral of atomic DOS are set to 0, then the Cumulative integral of atomic DOS will be ignored.

limit_pref_acdf:

type: int | float, optional, default: 0.0

argument path: loss[dos]/limit_pref_acdf

The prefactor of Cumulative integral of atomic DOS loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

When type is set to `tensor`:

(Supported Backend: TensorFlow)

pref:

type: `int` | `float`

argument path: `loss[tensor]/pref`

The prefactor of the weight of global loss. It should be larger than or equal to 0. If controls the weight of loss corresponding to global label, i.e. 'polarizability.npy' or dipole.npy, whose shape should be #frames x [9 or 3]. If it's larger than 0.0, this npy should be included.

pref_atomic:

type: `int` | `float`

argument path: `loss[tensor]/pref_atomic`

The prefactor of the weight of atomic loss. It should be larger than or equal to 0. If controls the weight of loss corresponding to atomic label, i.e. atomic_polarizability.npy or atomic_dipole.npy, whose shape should be #frames x ([9 or 3] x #selected atoms). If it's larger than 0.0, this npy should be included. Both pref and pref_atomic should be provided, and either can be set to 0.0.

loss_dict:

type: `dict`, optional

argument path: `loss_dict`

The dictionary of definitions of multiple loss functions in multi-task mode. Each loss_dict[fitting_key], with user-defined name fitting_key in model/fitting_net_dict, is the single definition of loss function, whose type should be set to tensor, ener or left unset.

training:

type: `dict`

argument path: `training`

The training options.

training_data:

type: `dict`, optional

argument path: `training/training_data`

Configurations of training data.

systems:

type: `typing.List[str]` | `str`

argument path: `training/training_data/systems`

The data systems for training. This key can be provided with a list that specifies the systems, or be provided with a string by which the prefix of all systems are given and the list of the systems is automatically generated.

batch_size:

type: `str` | `int` | `typing.List[int]`, optional, default: `auto`

argument path: `training/training_data/batch_size`

This key can be

- list: the length of which is the same as the systems. The batch size of each system is given by the elements of the list.

- int: all systems use the same batch size.
- string "auto": automatically determines the batch size so that the batch_size times the number of atoms in the system is no less than 32.
- string "auto:N": automatically determines the batch size so that the batch_size times the number of atoms in the system is no less than N.
- string "mixed:N": the batch data will be sampled from all systems and merged into a mixed system with the batch size N. Only support the se_atten descriptor.

If MPI is used, the value should be considered as the batch size per task.

auto_prob:

type: `str`, optional, default: `prob_sys_size`, alias: auto_prob_style

argument path: `training/training_data/auto_prob`

Determine the probability of systems automatically. The method is assigned by this key and can be

- "prob_uniform" : the probability all the systems are equal, namely 1.0/self.get_nsystems()
- "prob_sys_size" : the probability of a system is proportional to the number of batches in the system
- "prob_sys_size;stt_idx:end_idx:weight;stt_idx:end_idx:weight;..." : the list of systems is devided into blocks. A block is specified by stt_idx:end_idx:weight, where stt_idx is the starting index of the system, end_idx is then ending (not including) index of the system, the probabilities of the systems in this block sums up to weight, and the relatively probabilities within this block is proportional to the number of batches in the system.

sys_probs:

type: `typing.List[float]` | `NoneType`, optional, default: `None`, alias: sys_weights

argument path: `training/training_data/sys_probs`

A list of float if specified. Should be of the same length as systems, specifying the probability of each system.

validation_data:

type: `dict` | `NoneType`, optional, default: `None`

argument path: `training/validation_data`

Configurations of validation data. Similar to that of training data, except that a numb_btch argument may be configured

systems:

type: `typing.List[str]` | `str`

argument path: `training/validation_data/systems`

The data systems for validation. This key can be provided with a list that specifies the systems, or be provided with a string by which the prefix of all systems are given and the list of the systems is automatically generated.

batch_size:

type: `str` | `int` | `typing.List[int]`, optional, default: `auto`

argument path: `training/validation_data/batch_size`

This key can be

- list: the length of which is the same as the systems. The batch size of each system is given by the elements of the list.
- int: all systems use the same batch size.
- string "auto": automatically determines the batch size so that the batch_size times the number of atoms in the system is no less than 32.
- string "auto:N": automatically determines the batch size so that the batch_size times the number of atoms in the system is no less than N.

auto_prob:

type: `str`, optional, default: `prob_sys_size`, alias: auto_prob_style

argument path: `training/validation_data/auto_prob`

Determine the probability of systems automatically. The method is assigned by this key and can be

- "prob_uniform" : the probability all the systems are equal, namely 1.0/self.get_nsystems()
- "prob_sys_size" : the probability of a system is proportional to the number of batches in the system
- "prob_sys_size;stt_idx:end_idx:weight;stt_idx:end_idx:weight;..." : the list of systems is devided into blocks. A block is specified by stt_idx:end_idx:weight, where stt_idx is the starting index of the system, end_idx is then ending (not including) index of the system, the probabilities of the systems in this block sums up to weight, and the relatively probabilities within this block is proportional to the number of batches in the system.

sys_probs:

type: `typing.List[float]` | `NoneType`, optional, default: `None`, alias: sys_weights

argument path: `training/validation_data/sys_probs`

A list of float if specified. Should be of the same length as systems, specifying the probability of each system.

numb_btch:

type: `int`, optional, default: `1`, alias: numb_batch

argument path: `training/validation_data/numb_btch`

An integer that specifies the number of batches to be sampled for each validation period.

mixed_precision:

type: `dict`, optional

argument path: `training/mixed_precision`

Configurations of mixed precision.

output_prec:

type: `str`, optional, default: `float32`

argument path: `training/mixed_precision/output_prec`

The precision for mixed precision params. " "The trainable variables precision during the mixed precision training process, " "supported options are float32 only currently.

compute_prec:

type: `str`

argument path: `training/mixed_precision/compute_prec`

The precision for mixed precision compute. " "The compute precision during the mixed precision training process, "" "supported options are float16 and bfloat16 currently.

numb_steps:

type: `int`, alias: stop_batch
argument path: `training/numb_steps`

Number of training batch. Each training uses one batch of data.

seed:

type: `int` | `NoneType`, optional
argument path: `training/seed`

The random seed for getting frames from the training data set.

disp_file:

type: `str`, optional, default: `lcurve.out`
argument path: `training/disp_file`

The file for printing learning curve.

disp_freq:

type: `int`, optional, default: `1000`
argument path: `training/disp_freq`

The frequency of printing learning curve.

save_freq:

type: `int`, optional, default: `1000`
argument path: `training/save_freq`

The frequency of saving check point.

save_ckpt:

type: `str`, optional, default: `model.ckpt`
argument path: `training/save_ckpt`

The path prefix of saving check point files.

max_ckpt_keep:

type: `int`, optional, default: `5`
argument path: `training/max_ckpt_keep`

The maximum number of checkpoints to keep. The oldest checkpoints will be deleted once the number of checkpoints exceeds max_ckpt_keep. Defaults to 5.

disp_training:

type: `bool`, optional, default: `True`
argument path: `training/disp_training`

Displaying verbose information during training.

time_training:

type: `bool`, optional, default: `True`
argument path: `training/time_training`

Timing durining training.

profiling:

type: bool, optional, default: False
argument path: training/profiling

(Supported Backend: TensorFlow) Profiling during training.

profiling_file:

type: str, optional, default: timeline.json
argument path: training/profiling_file

(Supported Backend: TensorFlow) Output file for profiling.

enable_profiler:

type: bool, optional, default: False
argument path: training/enable_profiler

Enable TensorFlow Profiler (available in TensorFlow 2.3) or PyTorch Profiler to analyze performance. The log will be saved to tensorboard_log_dir.

tensorboard:

type: bool, optional, default: False
argument path: training/tensorboard

Enable tensorboard

tensorboard_log_dir:

type: str, optional, default: log
argument path: training/tensorboard_log_dir

The log directory of tensorboard outputs

tensorboard_freq:

type: int, optional, default: 1
argument path: training/tensorboard_freq

The frequency of writing tensorboard events.

data_dict:

type: dict, optional
argument path: training/data_dict

The dictionary of multi DataSystems in multi-task mode. Each data_dict[fitting_key], with user-defined name fitting_key in model/fitting_net_dict, contains training data and optional validation data definitions.

fitting_weight:

type: dict, optional
argument path: training/fitting_weight

Each fitting_weight[fitting_key], with user-defined name fitting_key in model/fitting_net_dict, is the training weight of fitting net fitting_key. Fitting nets with higher weights will be selected with higher probabilities to be trained in one step. Weights will be normalized and minus ones will be ignored. If not set, each fitting net will be equally selected when training.

warmup_steps:

type: int, optional

argument path: `training/warmup_steps`

(Supported Backend: PyTorch) The number of steps for learning rate warmup. During warmup, the learning rate begins at zero and progressively increases linearly to start_lr, rather than starting directly from start_lr

gradient_max_norm:

type: `float`, optional
argument path: `training/gradient_max_norm`

(Supported Backend: PyTorch) Clips the gradient norm to a maximum value. If the gradient norm exceeds this value, it will be clipped to this limit. No gradient clipping will occur if set to 0.

stat_file:

type: `str`, optional
argument path: `training/stat_file`

(Supported Backend: PyTorch) The file path for saving the data statistics results. If set, the results will be saved and directly loaded during the next training session, avoiding the need to recalculate the statistics. If the file extension is .h5 or .hdf5, an HDF5 file is used to store the statistics; otherwise, a directory containing NumPy binary files are used.

Depending on the value of opt_type, different sub args are accepted.

opt_type:

type: `str` (flag key), default: `Adam`
argument path: `training/opt_type`
possible choices: *`Adam`*, *`LKF`*

(Supported Backend: PyTorch) The type of optimizer to use.

When opt_type is set to `Adam`:

When opt_type is set to `LKF`:

kf_blocksize:

type: `int`, optional
argument path: `training[LKF]/kf_blocksize`

(Supported Backend: PyTorch) The blocksize for the Kalman filter.

nvnmd:

type: `dict`, optional
argument path: `nvnmd`

The nvnmd options.

version:

type: `int`
argument path: `nvnmd/version`

configuration the nvnmd version (0 | 1), 0 for 4 types, 1 for 32 types

max_nnei:

type: `int`

argument path: `nvnmd/max_nnei`

configuration the max number of neighbors, 128|256 for version 0, 128 for version 1

net_size:

type: `int`
argument path: `nvnmd/net_size`

configuration the number of nodes of fitting_net, just can be set as 128

map_file:

type: `str`
argument path: `nvnmd/map_file`

A file containing the mapping tables to replace the calculation of embedding nets

config_file:

type: `str`
argument path: `nvnmd/config_file`

A file containing the parameters about how to implement the model in certain hardware

weight_file:

type: `str`
argument path: `nvnmd/weight_file`

a *.npy file containing the weights of the model

enable:

type: `bool`
argument path: `nvnmd/enable`

enable the nvnmd training

restore_descriptor:

type: `bool`
argument path: `nvnmd/restore_descriptor`

enable to restore the parameter of embedding_net from weight.npy

restore_fitting_net:

type: `bool`
argument path: `nvnmd/restore_fitting_net`

enable to restore the parameter of fitting_net from weight.npy

quantize_descriptor:

type: `bool`
argument path: `nvnmd/quantize_descriptor`

enable the quantizatioin of descriptor

quantize_fitting_net:

type: `bool`
argument path: `nvnmd/quantize_fitting_net`

enable the quantizatioin of fitting_net

6.4 Parallel training

Note: Supported backends: TensorFlow , PyTorch

6.4.1 TensorFlow Implementation

Currently, parallel training in tensorflow version is enabled in a synchronized way with help of Horovod. Depending on the number of training processes (according to MPI context) and the number of GPU cards available, DeePMD-kit will decide whether to launch the training in parallel (distributed) mode or in serial mode. Therefore, no additional options are specified in your JSON/YAML input file.

Tuning learning rate

Horovod works in the data-parallel mode, resulting in a larger global batch size. For example, the real batch size is 8 when batch_size is set to 2 in the input file and you launch 4 workers. Thus, learning_rate is automatically scaled by the number of workers for better convergence. Technical details of such heuristic rule are discussed at Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.

The number of decay steps required to achieve the same accuracy can decrease by the number of cards (e.g., 1/2 of steps in the above case), but needs to be scaled manually in the input file.

In some cases, it won't work well when scaling the learning rate by worker count in a `linear` way. Then you can try `sqrt` or `none` by setting argument scale_by_worker like below.

```
"learning_rate" :{
    "scale_by_worker": "none",
    "type": "exp"
}
```

Scaling test

Testing `examples/water/se_e2_a` on an 8-GPU host, linear acceleration can be observed with the increasing number of cards.

Num of GPU cards	Seconds every 100 samples	Samples per second	Speed up
1	1.4515	68.89	1.00
2	1.5962	62.65*2	1.82
4	1.7635	56.71*4	3.29
8	1.7267	57.91*8	6.72

How to use

Training workers can be launched with `horovodrun`. The following command launches 4 processes on the same host:

```
CUDA_VISIBLE_DEVICES=4,5,6,7 horovodrun -np 4 \
    dp train --mpi-log=workers input.json
```

Need to mention, the environment variable `CUDA_VISIBLE_DEVICES` must be set to control parallelism on the occupied host where one process is bound to one GPU card.

To maximize the performance, one should follow FAQ: How to control the parallelism of a job to control the number of threads.

When using MPI with Horovod, `horovodrun` is a simple wrapper around `mpirun`. In the case where fine-grained control over options is passed to `mpirun`, `mpirun` can be invoked directly, and it will be detected automatically by Horovod, e.g.,

```
CUDA_VISIBLE_DEVICES=4,5,6,7 mpirun -l -launcher=fork -hosts=localhost -np 4 \
    dp train --mpi-log=workers input.json
```

this is sometimes necessary for an HPC environment.

Whether distributed workers are initiated can be observed in the "Summary of the training" section in the log (`world size` > 1, and `distributed`).

```
[0] DEEPMD INFO    ---Summary of the training---------------------------------------
[0] DEEPMD INFO    distributed
[0] DEEPMD INFO    world size:           4
[0] DEEPMD INFO    my rank:              0
[0] DEEPMD INFO    node list:            ['exp-13-57']
[0] DEEPMD INFO    running on:           exp-13-57
[0] DEEPMD INFO    computing device:     gpu:0
[0] DEEPMD INFO    CUDA_VISIBLE_DEVICES: 0,1,2,3
[0] DEEPMD INFO    Count of visible GPU: 4
[0] DEEPMD INFO    num_intra_threads:    0
[0] DEEPMD INFO    num_inter_threads:    0
[0] DEEPMD INFO    -----------------------------------------------------------------
```

Logging

What's more, 2 command-line arguments are defined to control the logging behavior when performing parallel training with MPI.

```
optional arguments:
  -l LOG_PATH, --log-path LOG_PATH
                        set log file to log messages to disk, if not
                        specified, the logs will only be output to console
                        (default: None)
  -m {master,collect,workers}, --mpi-log {master,collect,workers}
                        Set the manner of logging when running with MPI.
                        'master' logs only on main process, 'collect'
                        broadcasts logs from workers to master and 'workers'
                        means each process will output its own log (default:
                        master)
```

6.4.2 PyTorch Implementation

Currently, parallel training in pytorch version is implemented in the form of PyTorch Distributed Data Parallelism DDP. DeePMD-kit will decide whether to launch the training in parallel (distributed) mode or in serial mode depending on your execution command.

Dataloader and Dataset

One of the major differences between two backends during training is that the PyTorch version employs a multi-threaded data loading utility DataLoader. We utilize the PyTorch framework and have designed and implemented a multiprocessing data processing and loading system called DpLoaderSet based on torch DataLoader and Dataset.

First, we establish a DeepmdData class for each system, which is consistent with the TensorFlow version in this level. Then, we create a dataloader for each system, resulting in the same number of dataloaders as the number of systems. Next, we create a dataset for the dataloaders obtained in the previous step. This allows us to query the data for each system through this dataset, while the iteration pointers for each system are maintained by their respective dataloaders. Finally, a dataloader is created for the outermost dataset.

We achieve custom sampling methods using a weighted sampler. The length of the sampler is set to total_batch_num * num_workers.The parameter "num_workers" defines the number of threads involved in multi-threaded loading, which can be modified by setting the environment variable NUM_WORKERS (default: min(8, ncpus)).

> Note The underlying dataloader will use a distributed sampler to ensure that each GPU receives batches with different content in parallel mode, which will use sequential sampler in serial mode. In the TensorFlow version, Horovod shuffles the dataset using different random seeds for the same purpose..

```
flowchart LR

    subgraph systems
        subgraph system1
            direction LR
            frame1[frame 1]
            frame2[frame 2]
        end

        subgraph system2
            direction LR
            frame3[frame 3]
            frame4[frame 4]
            frame5[frame 5]
        end
    end

    subgraph dataset
        dataset1[dataset 1]
        dataset2[dataset 2]
    end
    system1 -- frames --> dataset1
    system2 --> dataset2

    subgraph distribted sampler
        ds1[distributed sampler 1]
        ds2[distributed sampler 2]
```

(continues on next page)

(continued from previous page)

```
    end
    dataset1 --> ds1
    dataset2 --> ds2

    subgraph dataloader
        dataloader1[dataloader 1]
        dataloader2[dataloader 2]
    end
    ds1 -- mini batch --> dataloader1
    ds2 --> dataloader2

    subgraph index[index on Rank 0]
        dl11[dataloader 1, entry 1]
        dl21[dataloader 2, entry 1]
        dl22[dataloader 2, entry 2]
    end
    dataloader1 --> dl11
    dataloader2 --> dl21
    dataloader2 --> dl22

    index -- for each step, choose 1 system --> WeightedSampler
    --> dploaderset --> bufferedq[buffered queue] --> model
```

How to use

We use `torchrun` to launch a DDP training session.

To start training with multiple GPUs in one node, set parameter `nproc_per_node` as the number of it:

```
torchrun --nproc_per_node=4 --no-python dp --pt train input.json
# Not setting `nproc_per_node` uses only 1 GPU
torchrun --no-python dp --pt train input.json
```

To train a model with a cluster, one can manually launch the task using the commands below (usually this should be done by your job management system). Set `nnodes` as the number of available nodes, `node_rank` as the rank of the current node among all nodes (not the rank of processes!), and `nproc_per_node` as the number of available GPUs in one node. Please make sure that every node can access the rendezvous address and port (`rdzv_endpoint` in the command), and has a same amount of GPUs.

```
# Running DDP on 2 nodes with 4 GPUs each
# On node 0:
torchrun --rdzv_endpoint=node0:12321 --nnodes=2 --nproc_per_node=4 --node_rank=0 --no_python dp --
↪pt train tests/water/se_e2_a.json
# On node 1:
torchrun --rdzv_endpoint=node0:12321 --nnodes=2 --nproc_per_node=4 --node_rank=1 --no_python dp --
↪pt train tests/water/se_e2_a.json
```

> Note Set environment variables to tune CPU specific optimizations in advance.
>
> Note for developers: `torchrun` by default passes settings as environment variables (list here).
>
> To check forward, backward, and communication time, please set env var `TORCH_CPP_LOG_LEVEL=INFO TORCH_DISTRIBUTED_DEBUG=DETAIL`. More details can be found here.

6.5 Multi-task training

Note: Supported backends: TensorFlow

6.5.1 Theory

The multi-task training process can simultaneously handle different datasets with properties that cannot be fitted in one network (e.g. properties from DFT calculations under different exchange-correlation functionals or different basis sets). These datasets are denoted by $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n_t)}$. For each dataset, a training task is defined as

$$\min_{\boldsymbol{\theta}} L^{(t)}(\boldsymbol{x}^{(t)}; \boldsymbol{\theta}^{(t)}, \tau), \quad t = 1, \ldots, n_t.$$

During the multi-task training process, all tasks share one descriptor with trainable parameters $\boldsymbol{\theta}_d$, while each of them has its own fitting network with trainable parameters $\boldsymbol{\theta}_f^{(t)}$, thus $\boldsymbol{\theta}^{(t)} = \{\boldsymbol{\theta}_d, \boldsymbol{\theta}_f^{(t)}\}$. At each training step, a task is randomly picked from $1, \ldots, n_t$, and the Adam optimizer is executed to minimize $L^{(t)}$ for one step to update the parameter $\boldsymbol{\theta}^{(t)}$. If different fitting networks have the same architecture, they can share the parameters of some layers to improve training efficiency.[1]

6.5.2 Perform the multi-task training

Training on multiple data sets (each data set contains several data systems) can be performed in multi-task mode, with one common descriptor and multiple specific fitting nets for each data set. One can simply switch the following parameters in training input script to perform multi-task mode:

- fitting_net –> fitting_net_dict, each key of which can be one individual fitting net.
- training_data, validation_data –> data_dict, each key of which can be one individual data set contains several data systems for corresponding fitting net, the keys must be consistent with those in fitting_net_dict.
- loss –> loss_dict, each key of which can be one individual loss setting for corresponding fitting net, the keys must be consistent with those in fitting_net_dict, if not set, the corresponding fitting net will use the default loss.
- (Optional) fitting_weight, each key of which can be a non-negative integer or float, deciding the chosen probability for corresponding fitting net in training, if not set or invalid, the corresponding fitting net will not be used.

The training procedure will automatically choose single-task or multi-task mode, based on the above parameters. Note that parameters of single-task mode and multi-task mode can not be mixed.

An example input for training energy and dipole in water system can be found here: multi-task input on water.

The supported descriptors for multi-task mode are listed:

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

- se_a (se_e2_a)
- se_r (se_e2_r)
- se_at (se_e3)
- se_atten
- se_atten_v2
- hybrid

The supported fitting nets for multi-task mode are listed:

- ener
- dipole
- polar

The output of `dp freeze` command in multi-task mode can be seen in freeze command.

6.5.3 Initialization from pre-trained multi-task model

For advance training in multi-task mode, one can first train the descriptor on several upstream datasets and then transfer it on new downstream ones with newly added fitting nets. At the second step, you can also inherit some fitting nets trained on upstream datasets, by merely adding fitting net keys in fitting_net_dict and optional fitting net weights in fitting_weight.

Take multi-task input on water again for example. You can first train a multi-task model using input script with the following model part:

```
    "model": {
        "type_map": ["O", "H"],
        "descriptor": {
            "type":     "se_e2_a",
            "sel":      [46, 92],
            "rcut_smth":    0.5,
            "rcut":     6.0,
            "neuron":       [25, 50, 100],
            "type_one_side": true
        },
        "fitting_net_dict": {
            "water_dipole": {
                "type":         "dipole",
                "neuron":       [100, 100, 100]
            },
            "water_ener": {
                "neuron":       [240, 240, 240],
                "resnet_dt":    true
            }
        },
    }
```

After training, you can freeze this multi-task model into one unit graph:

```
$ dp freeze -o graph.pb --united-model
```

Then if you want to transfer the trained descriptor and some fitting nets (take `water_ener` for example) to newly added datasets with new fitting net `water_ener_2`, you can modify the model part of the new input script in a more simplified way:

```
    "model": {
        "type_map": ["O", "H"],
        "descriptor": {},
        "fitting_net_dict": {
            "water_ener": {},
            "water_ener_2": {
                "neuron":       [240, 240, 240],
                "resnet_dt":    true,
            }
        },
    }
```

It will autocomplete the configurations according to the frozen graph.

Note that for newly added fitting net keys, other parts in the input script, including data_dict and loss_dict (optionally fitting_weight), should be set explicitly. While for old fitting net keys, it will inherit the old configurations if not set.

Finally, you can perform the modified multi-task training from the frozen model with command:

```
$ dp train input.json --init_frz_model graph.pb
```

6.5.4 Share layers among energy fitting networks

The multi-task training can be used to train multiple levels of energies (e.g. DFT and CCSD(T)) at the same time. In this situation, one can set model/fitting_net[ener]/layer_name> to share some of layers among fitting networks. The architecture of the layers with the same name should be the same.

For example, if one want to share the first and the third layers for two three-hidden-layer fitting networks, the following parameters should be set.

```
"fitting_net_dict": {
    "ccsd": {
        "neuron": [
            240,
            240,
            240
        ],
        "layer_name": ["l0", null, "l2", null]
    },
    "wb97m": {
        "neuron": [
            240,
            240,
            240
        ],
        "layer_name": ["l0", null, "l2", null]
    }
}
```

6.6 Multi-task training

Note: Supported backends: PyTorch

6.6.1 Theory

The multi-task training process can simultaneously handle different datasets with properties that cannot be fitted in one network (e.g. properties from DFT calculations under different exchange-correlation functionals or different basis sets). These datasets are denoted by $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n_t)}$. For each dataset, a training task is defined as

$$\min_{\boldsymbol{\theta}} L^{(t)}(\boldsymbol{x}^{(t)}; \boldsymbol{\theta}^{(t)}, \tau), \quad t = 1, \ldots, n_t.$$

In the Pytorch implementation, during the multi-task training process, all tasks can share any portion of the model parameters. A typical scenario is that each task shares the same descriptor with trainable parameters $\boldsymbol{\theta}_d$, while each has its own fitting network with trainable parameters $\boldsymbol{\theta}_f^{(t)}$, thus $\boldsymbol{\theta}^{(t)} = \{\boldsymbol{\theta}_d, \boldsymbol{\theta}_f^{(t)}\}$. At each training step, a task will be randomly selected from $1, \ldots, n_t$ according to the user-specified probability, and the Adam optimizer is executed to minimize $L^{(t)}$ for one step to update the parameter $\boldsymbol{\theta}^{(t)}$. In the case of multi-GPU parallel training, different GPUs will independently select their tasks. In the DPA-2 model, this multi-task training framework is adopted.[^1]

[^1] Duo Zhang, Xinzijian Liu, Xiangyu Zhang, Chengqian Zhang, Chun Cai, Hangrui Bi, Yiming Du, Xuejian Qin, Jiameng Huang, Bowen Li, Yifan Shan, Jinzhe Zeng, Yuzhi Zhang, Siyuan Liu, Yifan Li, Junhan Chang, Xinyan Wang, Shuo Zhou, Jianchuan Liu, Xiaoshan Luo, Zhenyu Wang, Wanrun Jiang, Jing Wu, Yudi Yang, Jiyuan Yang, Manyi Yang, Fu-Qiang Gong, Linshuang Zhang, Mengchao Shi, Fu-Zhi Dai, Darrin M. York, Shi Liu, Tong Zhu, Zhicheng Zhong, Jian Lv, Jun Cheng, Weile Jia, Mohan Chen, Guolin Ke, Weinan E, Linfeng Zhang, Han Wang, arXiv preprint arXiv:2312.15492 (2023) licensed under a Creative Commons Attribution (CC BY) license.

Compared with the previous TensorFlow implementation, the new support in PyTorch is more flexible and efficient. In particular, it makes multi-GPU parallel training and even tasks beyond DFT possible, enabling larger-scale and more general multi-task training to obtain more general pre-trained models.

6.6.2 Perform the multi-task training using PyTorch

Training on multiple data sets (each data set contains several data systems) can be performed in multi-task mode, typically with one common descriptor and multiple specific fitting nets for each data set. To proceed, one need to change the representation of the model definition in the input script. The core idea is to replace the previous single model definition model with multiple model definitions model/model_dict/model_key, define the shared parameters of the model part shared_dict, and then expand other parts for multi-model settings. Specifically, there are several parts that need to be modified:

- model/shared_dict: The parameter definition of the shared part, including various descriptors, type maps (or even fitting nets can be shared). Each module can be defined with a user-defined `part_key`, such as `my_descriptor`. The content needs to align with the corresponding definition in the single-task training model component, such as the definition of the descriptor.
- model/model_dict: The core definition of the model part and the explanation of sharing rules, starting with user-defined model name keys `model_key`, such as `my_model_1`. Each model part needs to align with the components of the single-task training model, but with the following sharing rules:

- – If you want to share the current model component with other tasks, which should be part of the model/shared_dict, you can directly fill in the corresponding `part_key`, such as `"descriptor": "my_descriptor",` to replace the previous detailed parameters. Here, you can also specify the shared_level, such as `"descriptor": "my_descriptor:shared_level",` and use the user-defined integer `shared_level` in the code to share the corresponding module to varying degrees (default is to share all parameters, i.e., `shared_level`=0). The parts that are exclusive to each model can be written following the previous definition.

- loss_dict: The loss settings corresponding to each task model, specified by the `model_key`. Each loss_dict/model_key contains the corresponding loss settings, which are the same as the definition in single-task training <loss>.

- training/data_dict: The data settings corresponding to each task model, specified by the `model_key`. Each `training/data_dict/model_key` contains the corresponding `training_data` and `validation_data` settings, which are the same as the definition in single-task training training_data and validation_data.

- (Optional) training/model_prob: The sampling weight settings corresponding to each `model_key`, i.e., the probability weight in the training step. You can specify any positive real number weight for each task. The higher the weight, the higher the probability of being sampled in each training. This setting is optional, and if not set, tasks will be sampled with equal weights.

An example input for multi-task training two models in water system is shown as following:

```
{
  "_comment": "that's all",
  "model": {
    "shared_dict": {
      "type_map_all": [
        "O",
        "H"
      ],
      "sea_descriptor_1": {
        "type": "se_e2_a",
        "sel": [
          46,
          92
        ],
        "rcut_smth": 0.50,
        "rcut": 6.00,
        "neuron": [
          25,
          50,
          100
        ],
        "resnet_dt": false,
        "axis_neuron": 16,
        "type_one_side": true,
        "seed": 1,
        "_comment": " that's all"
      },
      "_comment": "that's all"
    },
    "model_dict": {
      "water_1": {
        "type_map": "type_map_all",
        "descriptor": "sea_descriptor_1",
        "fitting_net": {
```

(continues on next page)

(continued from previous page)

```
          "neuron": [
            240,
            240,
            240
          ],
          "resnet_dt": true,
          "seed": 1,
          "_comment": " that's all"
        }
      },
      "water_2": {
        "type_map": "type_map_all",
        "descriptor": "sea_descriptor_1",
        "fitting_net": {
          "neuron": [
            240,
            240,
            240
          ],
          "resnet_dt": true,
          "seed": 1,
          "_comment": " that's all"
        }
      }
    }
  },
  "learning_rate": {
    "type": "exp",
    "decay_steps": 5000,
    "start_lr": 0.0002,
    "decay_rate": 0.98,
    "stop_lr": 3.51e-08,
    "_comment": "that's all"
  },
  "loss_dict": {
    "_comment": " that's all",
    "water_1": {
      "type": "ener",
      "start_pref_e": 0.02,
      "limit_pref_e": 1,
      "start_pref_f": 1000,
      "limit_pref_f": 1,
      "start_pref_v": 0,
      "limit_pref_v": 0
    },
    "water_2": {
      "type": "ener",
      "start_pref_e": 0.02,
      "limit_pref_e": 1,
      "start_pref_f": 1000,
      "limit_pref_f": 1,
      "start_pref_v": 0,
      "limit_pref_v": 0
    }
  },
  "training": {
    "model_prob": {
```

(continues on next page)

(continued from previous page)

```
      "water_1": 0.5,
      "water_2": 0.5
    },
    "data_dict": {
      "water_1": {
        "training_data": {
          "systems": [
            "../../water/data/data_0/",
            "../../water/data/data_1/",
            "../../water/data/data_2/"
          ],
          "batch_size": 1,
          "_comment": "that's all"
        },
        "validation_data": {
          "systems": [
            "../../water/data/data_3/"
          ],
          "batch_size": 1,
          "_comment": "that's all"
        }
      },
      "water_2": {
        "training_data": {
          "systems": [
            "../../water/data/data_0/",
            "../../water/data/data_1/",
            "../../water/data/data_2/"
          ],
          "batch_size": 1,
          "_comment": "that's all"
        }
      }
    },
    "numb_steps": 100000,
    "seed": 10,
    "disp_file": "lcurve.out",
    "disp_freq": 100,
    "save_freq": 100,
    "_comment": "that's all"
  }
}
```

6.6.3 Finetune from the pre-trained multi-task model

To finetune based on the checkpoint `model.pt` after the multi-task pre-training is completed, users can refer to this section.

6.7 TensorBoard Usage

Note: Supported backends: TensorFlow, PyTorch

TensorBoard provides the visualization and tooling needed for machine learning experimentation. Full instructions for TensorBoard can be found here.

6.7.1 Highlighted features

DeePMD-kit can now use most of the interesting features enabled by TensorBoard!

- Tracking and visualizing metrics, such as l2_loss, l2_energy_loss and l2_force_loss
- Visualizing the model graph (ops and layers)
- Viewing histograms of weights, biases, or other tensors as they change over time.
- Viewing summaries of trainable variables

6.7.2 How to use Tensorboard with DeePMD-kit

Before running TensorBoard, make sure you have generated summary data in a log directory by modifying the input script, setting tensorboard to true in the training subsection will enable the TensorBoard data analysis. eg. water_se_a.json.

```
   "training" : {
       "systems":        ["../data/"],
       "stop_batch":       1000000,
       "batch_size":       1,

       "seed":               1,

       "_comment": " display and restart",
       "_comment": " frequencies counted in batch",
       "disp_file":       "lcurve.out",
       "disp_freq":       100,
       "numb_test":       10,
       "save_freq":       1000,
       "save_ckpt":       "model.ckpt",

       "disp_training":true,
       "time_training":true,
       "tensorboard":       true,
       "tensorboard_log_dir":"log",
       "tensorboard_freq": 1000,
       "profiling":       false,
       "profiling_file":"timeline.json",
       "_comment":       "that's all"
   }
```

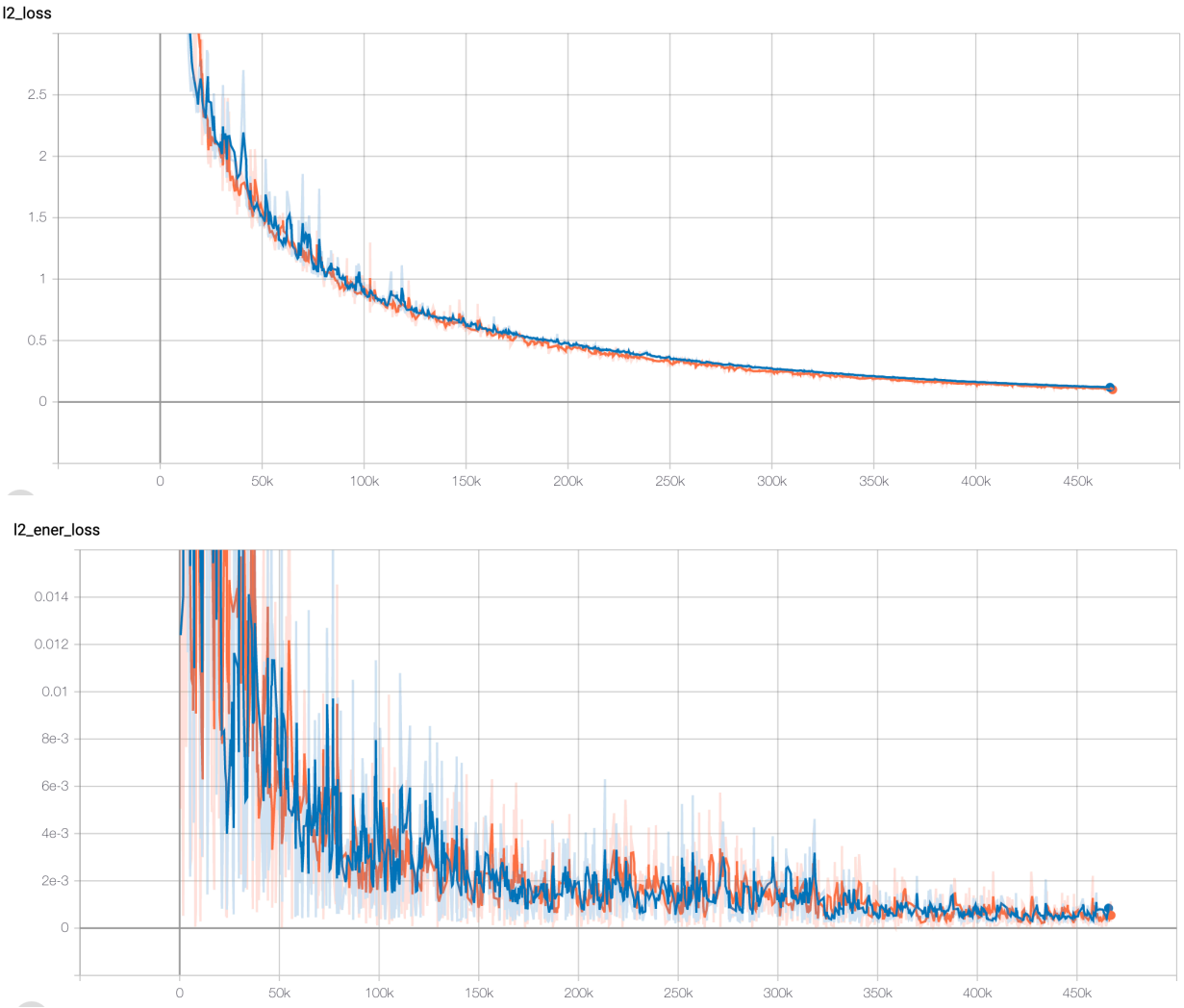
Once you have event files, run TensorBoard and provide the log directory. This should print that TensorBoard has started. Next, connect to http://tensorboard_server_ip:6006.

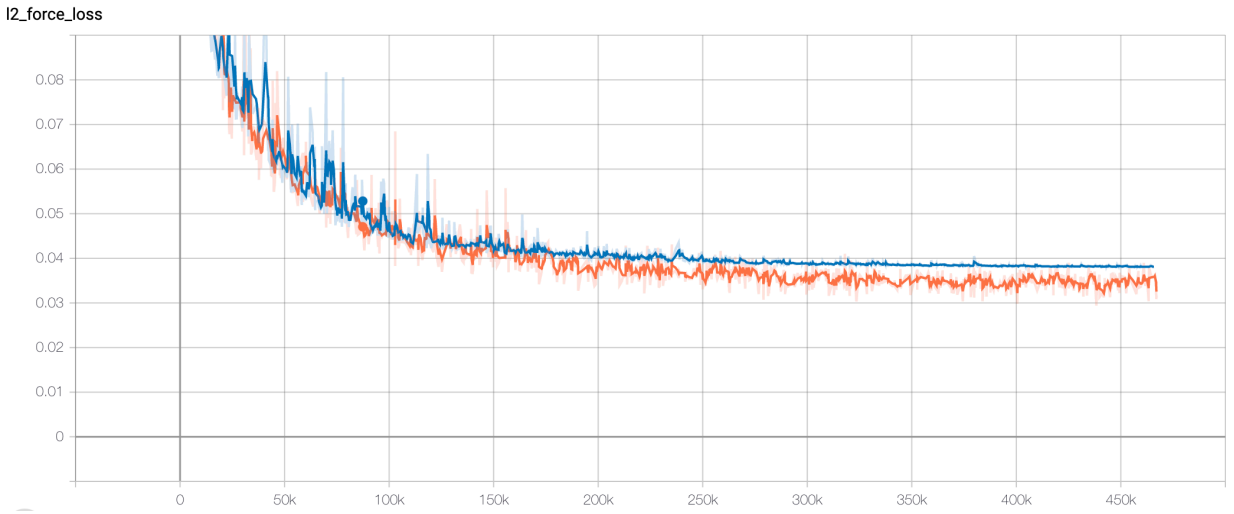
TensorBoard requires a logdir to read logs from. For info on configuring TensorBoard, run TensorBoard –help. One can easily change the log name with "tensorboard_log_dir" and the sampling frequency with "tensorboard_freq".

```
tensorboard --logdir path/to/logs
```

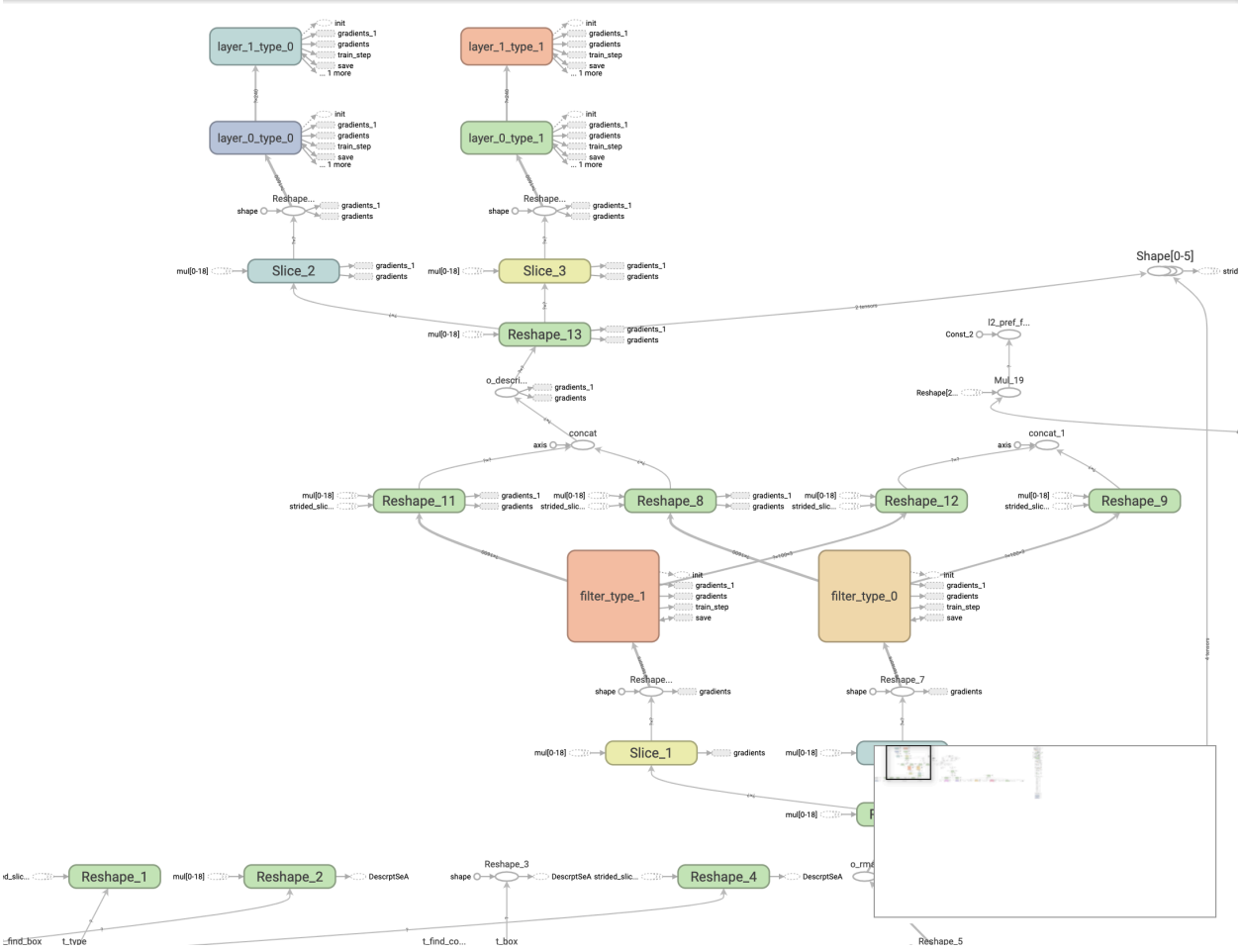
6.7.3 Examples

Tracking and visualizing loss metrics(red:train, blue:test)

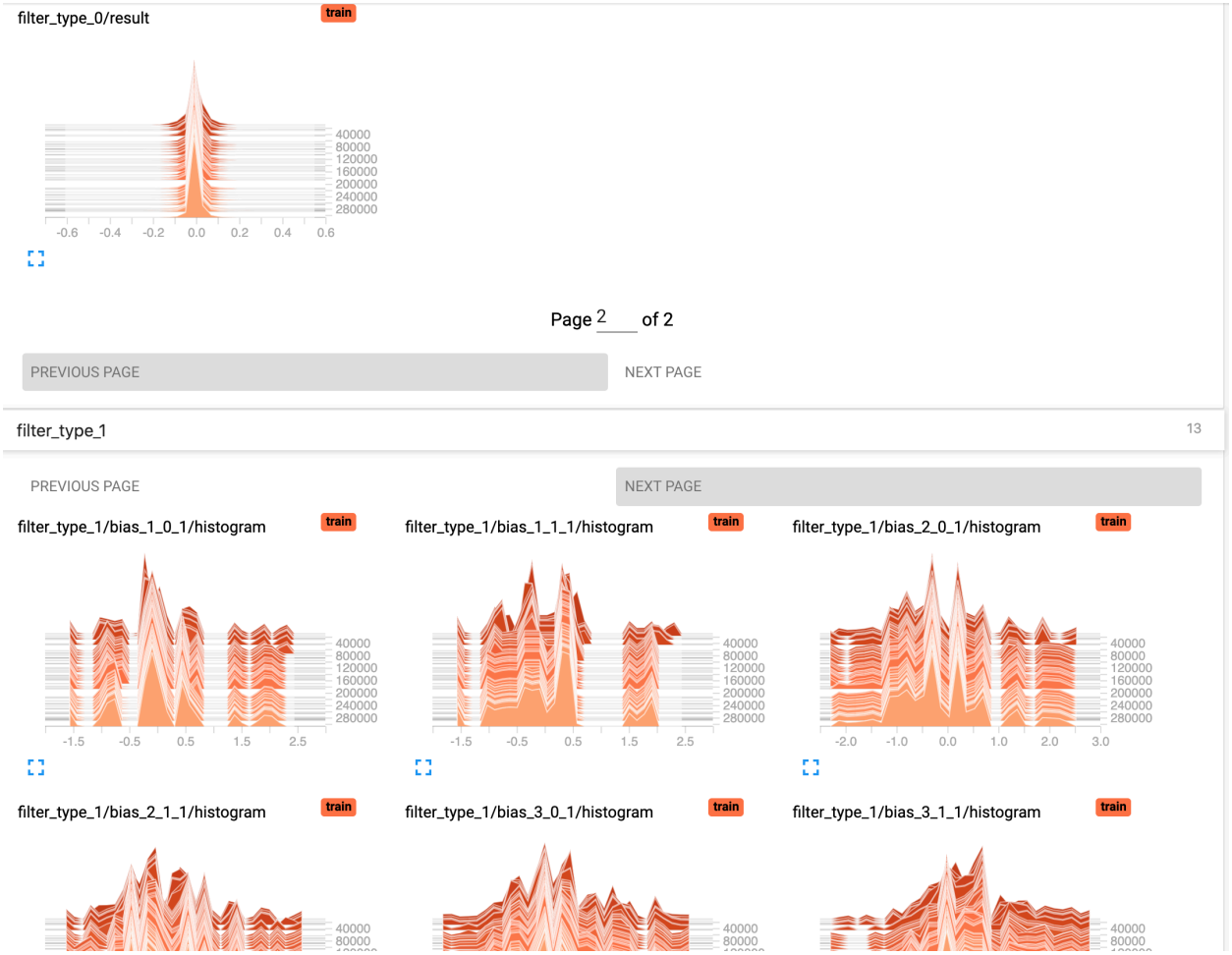


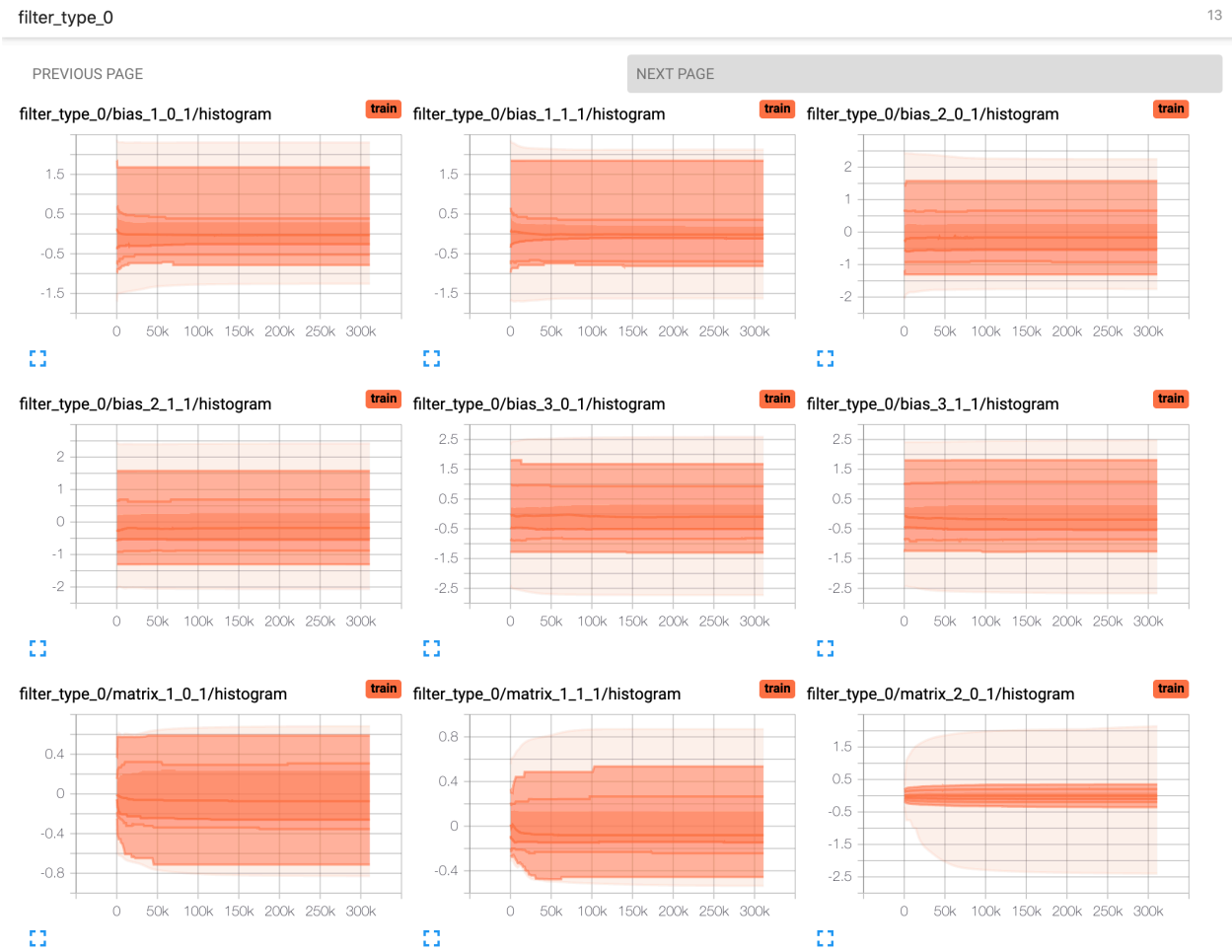


Visualizing DeePMD-kit model graph

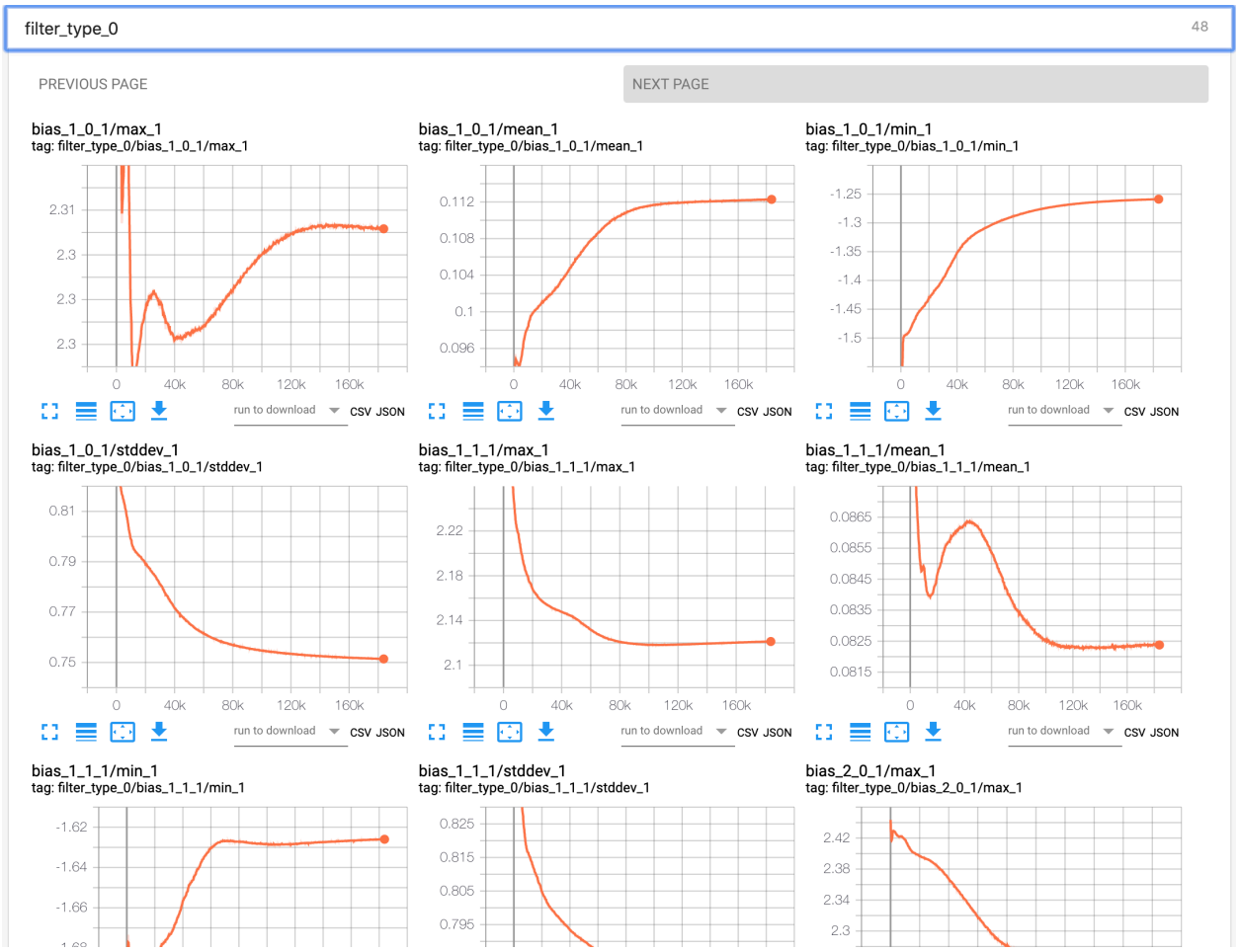


Viewing histograms of weights, biases, or other tensors as they change over time




filter_type_0
13
PREVIOUS PAGE
NEXT PAGE
filter_type_0/bias_1_0_1/histogram
train
filter_type_0/bias_1_1_1/histogram
train
filter_type_0/bias_2_0_1/histogram
train
filter_type_0/bias_2_1_1/histogram
train
filter_type_0/bias_3_0_1/histogram
train
filter_type_0/bias_3_1_1/histogram
train
filter_type_0/matrix_1_0_1/histogram
train
filter_type_0/matrix_1_1_1/histogram
train
filter_type_0/matrix_2_0_1/histogram
train
0
50k
100k
150k
200k
250k
300k

Viewing summaries of trainable variables



6.7.4 Attention

Allowing the tensorboard analysis will takes extra execution time.(eg, 15% increasing @Nvidia GTX 1080Ti double precision with default water sample)

TensorBoard can be used in Google Chrome or Firefox. Other browsers might work, but there may be bugs or performance issues.

6.8 Known limitations of using GPUs

If you use DeePMD-kit in a GPU environment, the acceptable value range of some variables is additionally restricted compared to the CPU environment due to the software's GPU implementations:

1. The number of atom types of a given system must be less than 128.
2. The maximum distance between an atom and its neighbors must be less than 128. It can be controlled by setting the rcut value of training parameters.

3. Theoretically, the maximum number of atoms that a single GPU can accept is about 10,000,000. However, this value is limited by the GPU memory size currently, usually within 1000,000 atoms even in the model compression mode.
4. The total sel value of training parameters(in `model/descriptor` section) must be less than 4096.
5. The size of the last layer of the embedding net must be less than 1024 during the model compression process.

6.9 Finetune the pre-trained model

Note: Supported backends: TensorFlow, PyTorch

Pretraining-and-finetuning is a widely used approach in other fields such as Computer Vision (CV) or Natural Language Processing (NLP) to vastly reduce the training cost, while it's not trivial in potential models. Compositions and configurations of data samples or even computational parameters in upstream software (such as VASP) may be different between the pre-trained and target datasets, leading to energy shifts or other diversities of training data.

Recently the emerging of methods such as DPA-1 has brought us to a new stage where we can perform similar pretraining-finetuning approaches. They can hopefully learn the common knowledge in the pre-trained dataset (especially the `force` information) and thus reduce the computational cost in downstream training tasks.

6.9.1 TensorFlow Implementation

If you have a pre-trained model `pretrained.pb` (here we support models using *`se_atten`* descriptor and *`ener`* fitting net) on a large dataset (for example, OC2M in DPA-1 paper), a finetuning strategy can be performed by simply running:

```
$ dp train input.json --finetune pretrained.pb
```

The command above will change the energy bias in the last layer of the fitting net in `pretrained.pb`, according to the training dataset in input.json.

Warning: Note that the elements in the training dataset must be contained in the pre-trained dataset.

The finetune procedure will inherit the model structures in `pretrained.pb`, and thus it will ignore the model parameters in `input.json`, such as descriptor, fitting_net, type_embedding and type_map. However, you can still set the `trainable` parameters in each part of `input.json` to control the training procedure.

To obtain a more simplified script, for example, you can change the model part in `input.json` to perform finetuning:

```
    "model": {
        "type_map":     ["O", "H"],
        "type_embedding": {"trainable":  true},
        "descriptor" :  {},
        "fitting_net" : {}
    }
```

6.9.2 PyTorch Implementation

In PyTorch version, we have introduced an updated, more adaptable approach to fine-tuning. This methodology encompasses two primary variations:

Single-task fine-tuning

Fine-tuning from a single-task pre-trained model

By saying "single-task pre-trained", we refer to a model pre-trained on one single dataset. This fine-tuning method is similar to the fine-tune approach supported by TensorFlow. It utilizes a single-task pre-trained model (`pretrained.pt`) and modifies the energy bias within its fitting net before continuing with training. The command for this operation is:

```
$ dp --pt train input.json --finetune pretrained.pt
```

Note: We do not support fine-tuning from a randomly initialized fitting net in this case, which is the same as implementations in TensorFlow.

The model section in input.json can be simplified as follows:

```
    "model": {
        "type_map":     ["O", "H"],
        "descriptor" :  {},
        "fitting_net" : {}
    }
```

Warning: The `type_map` will be overwritten based on that in the pre-trained model. Please ensure you are familiar with the `type_map` configuration in the pre-trained model before starting the fine-tuning process. This issue will be addressed in the future version.

Fine-tuning from a multi-task pre-trained model

Additionally, within the PyTorch implementation and leveraging the flexibility offered by the framework and the multi-task training capabilities provided by DPA2, we also support more general multitask pre-trained models, which includes multiple datasets for pre-training. These pre-training datasets share a common descriptor while maintaining their individual fitting nets, as detailed in the DPA2 paper.

For fine-tuning using this multitask pre-trained model (`multitask_pretrained.pt`), one can select a specific branch (e.g., `CHOOSEN_BRANCH`) included in `multitask_pretrained.pt` for fine-tuning with the following command:

```
$ dp --pt train input.json --finetune multitask_pretrained.pt --model-branch CHOOSEN_BRANCH
```

Note: To check the available model branches, you can typically refer to the documentation of the pre-trained model. If you're still unsure about the available branches, you can try inputting an arbitrary branch name. This will prompt an error message that displays a list of all the available model branches.

Please note that this feature will be improved in the upcoming version to provide a more user-friendly experience.

This command will start fine-tuning based on the pre-trained model's descriptor and the selected branch's fitting net. If –model-branch is not set, a randomly initialized fitting net will be used.

Multi-task fine-tuning

In typical scenarios, relying solely on single-task fine-tuning might gradually lead to the forgetting of information from the pre-trained datasets. In more advanced scenarios, it is desirable for the model to explicitly retain information from the pre-trained data during fine-tuning to prevent forgetting, which could be more beneficial for fine-tuning.

To achieve this, it is first necessary to clearly identify the datasets from which the pre-trained model originates and to download the corresponding datasets that need to be retained for subsequent multitask fine-tuning. Then, prepare a suitable input script for multitask fine-tuning `multi_input.json` as the following steps.

- Suppose the new dataset for fine-tuning is named `DOWNSTREAM_DATA`, and the datasets to be retained from multitask pre-trained model are `PRE_DATA1` and `PRE_DATA2`. One can:

1. Refer to the *multi-task-training* document to prepare a multitask training script for two systems, ideally extracting parts (i.e. model_dict, loss_dict, data_dict and model_prob parts) corresponding to `PRE_DATA1` and `PRE_DATA2` directly from the training script of the pre-trained model.
2. For `DOWNSTREAM_DATA`, select a desired branch to fine-tune from (e.g., `PRE_DATA1`), copy the configurations of `PRE_DATA1` as the configuration for `DOWNSTREAM_DATA` and insert the corresponding data path into the data_dict, thereby generating a three-system multitask training script.
3. In the model_dict for `DOWNSTREAM_DATA`, specify the branch from which `DOWNSTREAM_DATA` is to fine-tune using: `"finetune_head": "PRE_DATA1"`.

The complete `multi_input.json` should appear as follows ("..." means copied from input script of pre-trained model):

```
"model": {
  "shared_dict": {
    ...
  },
  "model_dict": {
    "PRE_DATA1": {
      "type_map": ...,
      "descriptor": ...,
      "fitting_net": ...
    },
    "PRE_DATA2": {
      "type_map": ...,
      "descriptor": ...,
      "fitting_net": ...
    },
    "DOWNSTREAM_DATA": {
      "finetune_head": "PRE_DATA1",
      "type_map": ...,
      "descriptor": ...,
      "fitting_net": ...
    },
  }
```

(continues on next page)

(continued from previous page)

```
  },
  "learning_rate": ...,
  "loss_dict": {
      "PRE_DATA1": ...,
      "PRE_DATA2": ...,
      "DOWNSTREAM_DATA": ...
  },
  "training": {
    "model_prob": {
      "PRE_DATA1": 0.5,
      "PRE_DATA2": 0.5,
      "DOWNSTREAM_DATA": 1.0
    },
    "data_dict": {
      "PRE_DATA1": ...,
      "PRE_DATA2": ...,
      "DOWNSTREAM_DATA": {
        "training_data": "training_data_config_for_DOWNSTREAM_DATA",
        "validation_data": "validation_data_config_for_DOWNSTREAM_DATA"
      }
    },
    ...
  }
```

Subsequently, run the command:

```
dp --pt train multi_input.json --finetune multitask_pretrained.pt
```

This will initiate multitask fine-tuning, where for branches `PRE_DATA1` and `PRE_DATA2`, it is akin to continuing training in `init-model` mode, whereas for `DOWNSTREAM_DATA`, fine-tuning will be based on the fitting net from `PRE_DATA1`. You can set `model_prob` for each dataset just the same as that in normal multitask training.

CHAPTER

SEVEN

FREEZE AND COMPRESS

7.1 Freeze a model

The trained neural network is extracted from a checkpoint and dumped into a protobuf(.pb) file. This process is called "freezing" a model. The idea and part of our code are from Morgan. To freeze a model, typically one does

TensorFlow

```
$ dp freeze -o model.pb
```

in the folder where the model is trained. The output model is called `model.pb`.

PyTorch

```
$ dp --pt freeze -o model.pth
```

in the folder where the model is trained. The output model is called `model.pth`.

In multi-task mode:

- This process will in default output several models, each of which contains the common descriptor and one of the user-defined fitting nets in fitting_net_dict, let's name it `fitting_key`, together frozen in `graph_{fitting_key}.pb`. Those frozen models are exactly the same as single-task output with fitting net `fitting_key`.
- If you add `--united-model` option in this situation, the total multi-task model will be frozen into one unit `graph.pb`, which is mainly for multi-task initialization and can not be used directly for inference.

7.2 Compress a model

Note: Supported backends: TensorFlow

7.2.1 Theory

The compression of the DP model uses three techniques, tabulated inference, operator merging, and precise neighbor indexing, to improve the performance of model training and inference when the model parameters are properly trained.

For better performance, the NN inference can be replaced by tabulated function evaluations if the input of the NN is of dimension one. The idea is to approximate the output of the NN by a piece-wise polynomial fitting. The input domain (a compact domain in $\mathbb{R}$) is divided into L_c equally spaced intervals, in which we apply a fifth-order polynomial $g_m^l(x)$ approximation of the m-th output component of the NN function:

$$g_m^l(x) = a_m^l x^5 + b_m^l x^4 + c_m^l x^3 + d_m^l x^2 + e_m^l x + f_m^l, \quad x \in [x_l, x_{l+1}),$$

where $l = 1, 2, \ldots, L_c$ is the index of the intervals, $x_1, \ldots, x_{L_c}, x_{L_c+1}$ are the endpoints of the intervals, and a_m^l, b_m^l, c_m^l, d_m^l, e_m^l, and f_m^l are the fitting parameters. The fitting parameters can be computed by the equations below:

$$a_m^l = \frac{1}{2\Delta x_l^5}[12h_{m,l} - 6(y'_{m,l+1} + y'_{m,l})\Delta x_l + (y''_{m,l+1} - y''_{m,l})\Delta x_l^2],$$

$$b_m^l = \frac{1}{2\Delta x_l^4}[-30h_{m,l} + (14y'_{m,l+1} + 16y'_{m,l})\Delta x_l + (-2y''_{m,l+1} + 3y''_{m,l})\Delta x_l^2],$$

$$c_m^l = \frac{1}{2\Delta x_l^3}[20h_{m,l} - (8y'_{m,l+1} + 12y'_{m,l})\Delta x_l + (y''_{m,l+1} - 3y''_{m,l})\Delta x_l^2],$$

$$d_m^l = \frac{1}{2}y''_{m,l},$$

$$e_m^l = y'_{m,l},$$

$$f_m^l = y_{m,l},$$

where $\Delta x_l = x_{l+1} - x_l$ denotes the size of the interval. $h_{m,l} = y_{m,l+1} - y_{m,l}$. $y_{m,l} = y_m(x_l)$, $y'_{m,l} = y'_m(x_l)$ and $y''_{m,l} = y''_m(x_l)$ are the value, the first-order derivative, and the second-order derivative of the m-th component of the target NN function at the interval point x_l, respectively. The first and second-order derivatives are easily calculated by the back-propagation of the NN functions.

In the standard DP model inference, taking the two-body embedding descriptor as an example, the matrix product $(\mathcal{G}^i)^T\mathcal{R}$ requires the transfer of the tensor $\mathcal{G}^i$ between the register and the host/device memories, which usually becomes the bottle-neck of the computation due to the relatively small memory bandwidth of the GPUs. The compressed DP model merges the matrix multiplication $(\mathcal{G}^i)^T\mathcal{R}$ with the tabulated inference step. More specifically, once one column of the $(\mathcal{G}^i)^T$ is evaluated, it is immediately multiplied with one row of the environment matrix in the register, and the outer product is deposited to the result of $(\mathcal{G}^i)^T\mathcal{R}$. By the operator merging technique, the allocation of $\mathcal{G}^i$ and the memory movement between register and host/device memories is avoided. The operator merging of the three-body embedding can be derived analogously.

The first dimension, N_c, of the environment ($\mathcal{R}^i$) and embedding ($\mathcal{G}^i$) matrices is the expected maximum number of neighbors. If the number of neighbors of an atom is smaller than N_c, the corresponding positions of the matrices are pad with zeros. In practice, if the real number of neighbors is significantly smaller than N_c,

a notable operation is spent on the multiplication of padding zeros. In the compressed DP model, the number of neighbors is precisely indexed at the tabulated inference stage, further saving computational costs.[1]

7.2.2 Instructions

Once the frozen model is obtained from DeePMD-kit, we can get the neural network structure and its parameters (weights, biases, etc.) from the trained model, and compress it in the following way:

```
dp compress -i graph.pb -o graph-compress.pb
```

where `-i` gives the original frozen model, `-o` gives the compressed model. Several other command line options can be passed to `dp compress`, which can be checked with

```
$ dp compress --help
```

An explanation will be provided

```
usage: dp compress [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
                   [-m {master,collect,workers}] [-i INPUT] [-o OUTPUT]
                   [-s STEP] [-e EXTRAPOLATE] [-f FREQUENCY]
                   [-c CHECKPOINT_FOLDER]

optional arguments:
  -h, --help            show this help message and exit
  -v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}, --log-level {DEBUG,3,INFO,2,WARNING,1,ERROR,0}
                        set verbosity level by string or number, 0=ERROR,
                        1=WARNING, 2=INFO and 3=DEBUG (default: INFO)
  -l LOG_PATH, --log-path LOG_PATH
                        set log file to log messages to disk, if not
                        specified, the logs will only be output to console
                        (default: None)
  -m {master,collect,workers}, --mpi-log {master,collect,workers}
                        Set the manner of logging when running with MPI.
                        'master' logs only on main process, 'collect'
                        broadcasts logs from workers to master and 'workers'
                        means each process will output its own log (default:
                        master)
  -i INPUT, --input INPUT
                        The original frozen model, which will be compressed by
                        the code (default: frozen_model.pb)
  -o OUTPUT, --output OUTPUT
                        The compressed model (default:
                        frozen_model_compressed.pb)
  -s STEP, --step STEP  Model compression uses fifth-order polynomials to
                        interpolate the embedding-net. It introduces two
                        tables with different step size to store the
                        parameters of the polynomials. The first table covers
                        the range of the training data, while the second table
                        is an extrapolation of the training data. The domain
                        of each table is uniformly divided by a given step
```

(continues on next page)

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

(continued from previous page)

```
                      size. And the step(parameter) denotes the step size of
                      the first table and the second table will use 10 *
                      step as it's step size to save the memory. Usually the
                      value ranges from 0.1 to 0.001. Smaller step means
                      higher accuracy and bigger model size (default: 0.01)
-e EXTRAPOLATE, --extrapolate EXTRAPOLATE
                      The domain range of the first table is automatically
                      detected by the code: [d_low, d_up]. While the second
                      table ranges from the first table's upper
                      boundary(d_up) to the extrapolate(parameter) * d_up:
                      [d_up, extrapolate * d_up] (default: 5)
-f FREQUENCY, --frequency FREQUENCY
                      The frequency of tabulation overflow check(Whether the
                      input environment matrix overflow the first or second
                      table range). By default do not check the overflow
                      (default: -1)
-c CHECKPOINT_FOLDER, --checkpoint-folder CHECKPOINT_FOLDER
                      path to checkpoint folder (default: .)
-t TRAINING_SCRIPT, --training-script TRAINING_SCRIPT
                      The training script of the input frozen model
                      (default: None)
```

Parameter explanation

Model compression, which includes tabulating the embedding net. The table is composed of fifth-order polynomial coefficients and is assembled from two sub-tables. For model descriptor with `se_e2_a` type, the first sub-table takes the stride(parameter) as its uniform stride, while the second sub-table takes 10 _ stride as its uniform stride; For model descriptor with `se_e3` type, the first sub-table takes 10 _ stride as it's uniform stride, while the second sub-table takes 100 _ stride as it's uniform stride. The range of the first table is automatically detected by DeePMD-kit, while the second table ranges from the first table's upper boundary(upper) to the extrapolate(parameter) _ upper. Finally, we added a check frequency parameter. It indicates how often the program checks for overflow(if the input environment matrix overflows the first or second table range) during the MD inference.

Justification of model compression

Model compression, with little loss of accuracy, can greatly speed up MD inference time. According to different simulation systems and training parameters, the speedup can reach more than 10 times at both CPU and GPU devices. At the same time, model compression can greatly change memory usage, reducing as much as 20 times under the same hardware conditions.

Acceptable original model version

The model compression interface requires the version of DeePMD-kit used in the original model generation should be `2.0.0-alpha.0` or above. If one has a frozen 1.2 or 1.3 model, one can upgrade it through the `dp convert-from` interface. (eg: `dp convert-from 1.2/1.3 -i old_frozen_model.pb -o new_frozen_model.pb`)

Acceptable descriptor type

Descriptors with `se_e2_a`, `se_e3`, `se_e2_r` and `se_atten_v2` types are supported by the model compression feature. `Hybrid` mixed with the above descriptors is also supported.

Notice: Model compression for the `se_atten_v2` descriptor is exclusively designed for models with the training parameter attn_layer set to 0.

Available activation functions for descriptor:

- tanh

- gelu
- relu
- relu6
- softplus
- sigmoid

CHAPTER

EIGHT

TEST

8.1 Test a model

The frozen model can be used in many ways. The most straightforward test can be performed using `dp test`. A typical usage of `dp test` is

```
dp test -m graph.pb -s /path/to/system -n 30
```

where `-m` gives the tested model, `-s` the path to the tested system and `-n` the number of tested frames. Several other command line options can be passed to `dp test`, which can be checked with

```
$ dp test --help
```

An explanation will be provided

```
usage: dp test [-h] [-m MODEL] [-s SYSTEM] [-S SET_PREFIX] [-n NUMB_TEST]
               [-r RAND_SEED] [--shuffle-test] [-d DETAIL_FILE]

optional arguments:
  -h, --help            show this help message and exit
  -m MODEL, --model MODEL
                        Frozen model file to import
  -s SYSTEM, --system SYSTEM
                        The system dir
  -S SET_PREFIX, --set-prefix SET_PREFIX
                        The set prefix
  -n NUMB_TEST, --numb-test NUMB_TEST
                        The number of data for test
  -r RAND_SEED, --rand-seed RAND_SEED
                        The random seed
  --shuffle-test        Shuffle test data
  -d DETAIL_FILE, --detail-file DETAIL_FILE
                        The prefix to files where details of energy, force and virial accuracy/
↪accuracy per atom will be written
  -a, --atomic          Test the accuracy of atomic label, i.e. energy / tensor (dipole, polar)
```

8.2 Calculate Model Deviation

8.2.1 Theory

Model deviation ϵ_y is the standard deviation of properties $\boldsymbol{y}$ inferred by an ensemble of models $\mathcal{M}_1, \ldots, \mathcal{M}_{n_m}$ that are trained by the same dataset(s) with the model parameters initialized independently. The DeePMD-kit supports $\boldsymbol{y}$ to be the atomic force $\boldsymbol{F}_i$ and the virial tensor $\boldsymbol{\Xi}$. The model deviation is used to estimate the error of a model at a certain data frame, denoted by $\boldsymbol{x}$, containing the coordinates and chemical species of all atoms. We present the model deviation of the atomic force and the virial tensor

$$\epsilon_{\boldsymbol{F},i}(\boldsymbol{x}) = \sqrt{\langle \| \boldsymbol{F}_i(\boldsymbol{x};\boldsymbol{\theta}_k) - \langle \boldsymbol{F}_i(\boldsymbol{x};\boldsymbol{\theta}_k) \rangle \|^2 \rangle},$$

$$\epsilon_{\boldsymbol{\Xi},\alpha\beta}(\boldsymbol{x}) = \frac{1}{N}\sqrt{\langle (\Xi_{\alpha\beta}(\boldsymbol{x};\boldsymbol{\theta}_k) - \langle \Xi_{\alpha\beta}(\boldsymbol{x};\boldsymbol{\theta}_k) \rangle)^2 \rangle},$$

where $\boldsymbol{\theta}_k$ is the parameters of the model $\mathcal{M}_k$, and the ensemble average $\langle \cdot \rangle$ is estimated by

$$\langle \boldsymbol{y}(\boldsymbol{x};\boldsymbol{\theta}_k) \rangle = \frac{1}{n_m}\sum_{k=1}^{n_m} \boldsymbol{y}(\boldsymbol{x};\boldsymbol{\theta}_k).$$

Small $\epsilon_{\boldsymbol{F},i}$ means the model has learned the given data; otherwise, it is not covered, and the training data needs to be expanded. If the magnitude of $\boldsymbol{F}_i$ or $\boldsymbol{\Xi}$ is quite large, a relative model deviation $\epsilon_{\boldsymbol{F},i,\mathrm{rel}}$ or $\epsilon_{\boldsymbol{\Xi},\alpha\beta,\mathrm{rel}}$ can be used instead of the absolute model deviation:

$$\epsilon_{\boldsymbol{F},i,\mathrm{rel}}(\boldsymbol{x}) = \frac{|\epsilon_{\boldsymbol{F},i}(\boldsymbol{x})|}{|\langle \boldsymbol{F}_i(\boldsymbol{x};\boldsymbol{\theta}_k) \rangle| + \nu},$$

$$\epsilon_{\boldsymbol{\Xi},\alpha\beta,\mathrm{rel}}(\boldsymbol{x}) = \frac{\epsilon_{\boldsymbol{\Xi},\alpha\beta}(\boldsymbol{x})}{|\langle \boldsymbol{\Xi}(\boldsymbol{x};\boldsymbol{\theta}_k) \rangle| + \nu},$$

where ν is a small constant used to protect an atom where the magnitude of $\boldsymbol{F}_i$ or $\boldsymbol{\Xi}$ is small from having a large model deviation.

Statistics of $\epsilon_{\boldsymbol{F},i}$ and $\epsilon_{\boldsymbol{\Xi},\alpha\beta}$ can be provided, including the maximum, average, and minimal model deviation over the atom index i and over the component index α, β, respectively. The maximum model deviation of forces $\epsilon_{\boldsymbol{F},\max}$ in a frame was found to be the best error indicator in a concurrent or active learning algorithm.[1]

8.2.2 Instructions

One can also use a subcommand to calculate the deviation of predicted forces or virials for a bunch of models in the following way:

```
dp model-devi -m graph.000.pb graph.001.pb graph.002.pb graph.003.pb -s ./data -o model_devi.out
```

where `-m` specifies graph files to be calculated, `-s` gives the data to be evaluated, `-o` the file to which model deviation results is dumped. Here is more information on this sub-command:

[1] This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

```
usage: dp model-devi [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}]
                     [-l LOG_PATH] [-m MODELS [MODELS ...]] [-s SYSTEM]
                     [-S SET_PREFIX] [-o OUTPUT] [-f FREQUENCY] [-i ITEMS]

optional arguments:
  -h, --help            show this help message and exit
  -v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}, --log-level {DEBUG,3,INFO,2,WARNING,1,ERROR,0}
                        set verbosity level by string or number, 0=ERROR,
                        1=WARNING, 2=INFO and 3=DEBUG (default: INFO)
  -l LOG_PATH, --log-path LOG_PATH
                        set log file to log messages to disk, if not
                        specified, the logs will only be output to console
                        (default: None)
  -m MODELS [MODELS ...], --models MODELS [MODELS ...]
                        Frozen models file to import (default:
                        ['graph.000.pb', 'graph.001.pb', 'graph.002.pb',
                        'graph.003.pb'])
  -s SYSTEM, --system SYSTEM
                        The system directory, not support recursive detection.
                        (default: .)
  -S SET_PREFIX, --set-prefix SET_PREFIX
                        The set prefix (default: set)
  -o OUTPUT, --output OUTPUT
                        The output file for results of model deviation
                        (default: model_devi.out)
  -f FREQUENCY, --frequency FREQUENCY
                        The trajectory frequency of the system (default: 1)
```

For more details concerning the definition of model deviation and its application, please refer to Yuzhi Zhang, Haidi Wang, Weijie Chen, Jinzhe Zeng, Linfeng Zhang, Han Wang, and Weinan E, DP-GEN: A concurrent learning platform for the generation of reliable deep learning based potential energy models, Computer Physics Communications, 2020, 253, 107206.

8.2.3 Relative model deviation

By default, the model deviation is output in absolute value. If the argument `--relative` is passed, then the relative model deviation of the force will be output, including values output by the argument `--atomic`. The relative model deviation of the force on atom i is defined by

$$E_{f_i} = \frac{|D_{f_i}|}{|f_i| + l}$$

where D_{f_i} is the absolute model deviation of the force on atom i, f_i is the norm of the force and l is provided as the parameter of the keyword `relative`. If the argument `--relative_v` is set, then the relative model deviation of the virial will be output instead of the absolute value, with the same definition of that of the force:

$$E_{v_i} = \frac{|D_{v_i}|}{|v_i| + l}$$

CHAPTER

NINE

INFERENCE

Note that the model for inference is required to be compatible with the DeePMD-kit package. See Model compatibility for details.

9.1 Python interface

One may use the python interface of DeePMD-kit for model inference, an example is given as follows

```
from deepmd.infer import DeepPot
import numpy as np

dp = DeepPot("graph.pb")
coord = np.array([[1, 0, 0], [0, 0, 1.5], [1, 0, 3]]).reshape([1, -1])
cell = np.diag(10 * np.ones(3)).reshape([1, -1])
atype = [1, 0, 1]
e, f, v = dp.eval(coord, cell, atype)
```

where `e`, `f` and `v` are predicted energy, force and virial of the system, respectively.

Furthermore, one can use the python interface to calculate model deviation.

```
from deepmd.infer import calc_model_devi
from deepmd.infer import DeepPot as DP
import numpy as np

coord = np.array([[1, 0, 0], [0, 0, 1.5], [1, 0, 3]]).reshape([1, -1])
cell = np.diag(10 * np.ones(3)).reshape([1, -1])
atype = [1, 0, 1]
graphs = [DP("graph.000.pb"), DP("graph.001.pb")]
model_devi = calc_model_devi(coord, cell, atype, graphs)
```

Note that if the model inference or model deviation is performed cyclically, one should avoid calling the same model multiple times. Otherwise, TensorFlow or PyTorch will never release the memory, and this may lead to an out-of-memory (OOM) error.

9.1.1 External neighbor list algorithm

Note: Supported backends: TensorFlow

The native neighbor list algorithm of the DeePMD-kit is in $O(N^2)$ complexity (N is the number of atoms). While this is not a problem for small systems that quantum methods can afford, the large systems for molecular dynamics have slow performance. In this case, one may pass an external neighbor list that has lower complexity to *DeepPot*, once it is compatible with `ase.neighborlist.NewPrimitiveNeighborList`.

```
import ase.neighborlist

neighbor_list = ase.neighborlist.NewPrimitiveNeighborList(
    cutoffs=6, bothways=True, self_interaction=False
)
dp = DeepPot("graph.pb", neighbor_list=neighbor_list)
```

The `update` and `build` methods will be called by *DeepPot*, and `first_neigh`, `pair_second`, and `offset_vec` properties will be used.

9.2 C/C++ interface

9.2.1 C++ interface

The C++ interface of DeePMD-kit is also available for the model interface, which is considered faster than the Python interface. An example `infer_water.cpp` is given below:

```
#include "deepmd/DeepPot.h"

int main(){
  deepmd::DeepPot dp ("graph.pb");
  std::vector<double > coord = {1., 0., 0., 0., 0., 1.5, 1. ,0. ,3.};
  std::vector<double > cell = {10., 0., 0., 0., 10., 0., 0., 0., 10.};
  std::vector<int > atype = {1, 0, 1};
  double e;
  std::vector<double > f, v;
  dp.compute (e, f, v, coord, atype, cell);
}
```

where `e`, `f` and `v` are predicted energy, force and virial of the system, respectively. See *deepmd::DeepPot* for details.

You can compile `infer_water.cpp` using `gcc`:

```
gcc infer_water.cpp -L $deepmd_root/lib -L $tensorflow_root/lib -I $deepmd_root/include -Wl,--no-
↪as-needed -ldeepmd_cc -lstdc++ -ltensorflow_cc -Wl,-rpath=$deepmd_root/lib -Wl,-rpath=
↪$tensorflow_root/lib -o infer_water
```

and then run the program:

```
./infer_water
```

9.2.2 C interface

Although C is harder to write, the C library will not be affected by different versions of C++ compilers.

An example `infer_water.c` is given below:

```c
#include <stdio.h>
#include <stdlib.h>
#include "deepmd/c_api.h"

int main(){
  const char* model = "graph.pb";
  double coord[] = {1., 0., 0., 0., 0., 1.5, 1. ,0. ,3.};
  double cell[] = {10., 0., 0., 0., 10., 0., 0., 0., 10.};
  int atype[] = {1, 0, 1};
  // init C pointers with given memory
  double* e = malloc(sizeof(*e));
  double* f = malloc(sizeof(*f) * 9); // natoms * 3
  double* v = malloc(sizeof(*v) * 9);
  double* ae = malloc(sizeof(*ae) * 9); // natoms
  double* av = malloc(sizeof(*av) * 27); // natoms * 9
  // DP model
  DP_DeepPot* dp = DP_NewDeepPot(model);
  DP_DeepPotCompute (dp, 3, coord, atype, cell, e, f, v, ae, av);
  // print results
  printf("energy: %f\n", *e);
  for (int ii = 0; ii < 9; ++ii)
    printf("force[%d]: %f\n", ii, f[ii]);
  for (int ii = 0; ii < 9; ++ii)
    printf("force[%d]: %f\n", ii, v[ii]);
  // free memory
  free(e);
  free(f);
  free(v);
  free(ae);
  free(av);
  DP_DeleteDeepPot(dp);
}
```

where `e`, `f` and `v` are predicted energy, force and virial of the system, respectively. `ae` and `av` are atomic energy and atomic virials, respectively. See *DP_DeepPotCompute()* for details.

You can compile `infer_water.c` using `gcc`:

```
gcc infer_water.c -L $deepmd_root/lib -L $tensorflow_root/lib -I $deepmd_root/include -Wl,--no-as-
↪needed -ldeepmd_c -Wl,-rpath=$deepmd_root/lib -Wl,-rpath=$tensorflow_root/lib -o infer_water
```

and then run the program:

```
./infer_water
```

9.2.3 Header-only C++ library interface (recommended)

The header-only C++ library is built based on the C library. Thus, it has the same ABI compatibility as the C library but provides a powerful C++ interface. To use it, include `deepmd/deepmd.hpp`.

```
#include "deepmd/deepmd.hpp"

int main(){
  deepmd::hpp::DeepPot dp ("graph.pb");
  std::vector<double > coord = {1., 0., 0., 0., 0., 1.5, 1. ,0. ,3.};
  std::vector<double > cell = {10., 0., 0., 0., 10., 0., 0., 0., 10.};
  std::vector<int > atype = {1, 0, 1};
  double e;
  std::vector<double > f, v;
  dp.compute (e, f, v, coord, atype, cell);
}
```

Note that the feature of the header-only C++ library is still limited compared to the original C++ library. See *deepmd::hpp::DeepPot* for details.

You can compile `infer_water_hpp.cpp` using `gcc`:

```
gcc infer_water_hpp.cpp -L $deepmd_root/lib -L $tensorflow_root/lib -I $deepmd_root/include -Wl,--
↪no-as-needed -ldeepmd_c -Wl,-rpath=$deepmd_root/lib -Wl,-rpath=$tensorflow_root/lib -o infer_
↪water_hpp
```

and then run the program:

```
./infer_water_hpp
```

In some cases, one may want to pass the custom neighbor list instead of the native neighbor list. The above code can be revised as follows:

```
  // neighbor list
  std::vector<std::vector<int >> nlist_vec = {
    {1, 2},
    {0, 2},
    {0, 1}
    };
  std::vector<int> ilist(3), numneigh(3);
  std::vector<int*> firstneigh(3);
  InputNlist nlist(3, &ilist[0], &numneigh[0], &firstneigh[0]);
  convert_nlist(nlist, nlist_vec);
  dp.compute (e, f, v, coord, atype, cell, 0, nlist, 0);
```

Here, `nlist_vec` means the neighbors of atom 0 are atom 1 and atom 2, the neighbors of atom 1 are atom 0 and atom 2, and the neighbors of atom 2 are atom 0 and atom 1.

9.3 Node.js interface

If Node.js interface is installed, one can use the Node.js interface for model inference, which is a wrapper of the header-only C++ API.

A simple example is shown below.

```
const deepmd = require("deepmd-kit");

const dp = new deepmd.DeepPot("graph.pb");

const coord = [1, 0, 0, 0, 0, 1.5, 1, 0, 3];
const atype = [1, 0, 1];
const cell = [10, 0, 0, 0, 10, 0, 0, 0, 10];

const v_coord = new deepmd.vectord(coord.length);
const v_atype = new deepmd.vectori(atype.length);
const v_cell = new deepmd.vectord(cell.length);
for (var i = 0; i < coord.length; i++) v_coord.set(i, coord[i]);
for (var i = 0; i < atype.length; i++) v_atype.set(i, atype[i]);
for (var i = 0; i < cell.length; i++) v_cell.set(i, cell[i]);

var energy = 0.0;
var v_forces = new deepmd.vectord();
var v_virials = new deepmd.vectord();

energy = dp.compute(energy, v_forces, v_virials, v_coord, v_atype, v_cell);

console.log("energy:", energy);
console.log(
  "forces:",
  [...Array(v_forces.size()).keys()].map((i) => v_forces.get(i)),
);
console.log(
  "virials:",
  [...Array(v_virials.size()).keys()].map((i) => v_virials.get(i)),
);
```

Energy, forces, and virials will be printed to the screen.

CHAPTER

TEN

COMMAND LINE INTERFACE

DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics

```
usage: dp [-h] [-b {tensorflow,tf,pytorch,pt} | --tensorflow | --pytorch]
          [--version]
          {transfer,train,freeze,test,compress,doc-train-input,model-devi,convert-from,neighbor-
↪stat,train-nvnmd,gui,convert-backend}
          ...
```

10.1 Named Arguments

-b, --backend	Possible choices: tensorflow, tf, pytorch, pt The backend of the model. Default can be set by environment variable DP_BACKEND. Default: "tensorflow"
--tensorflow, --tf	Alias for –backend tensorflow
--pytorch, --pt	Alias for –backend pytorch
--version	show program's version number and exit

10.2 Valid subcommands

command	Possible choices: transfer, train, freeze, test, compress, doc-train-input, model-devi, convert-from, neighbor-stat, train-nvnmd, gui, convert-backend

10.3 Sub-commands

10.3.1 transfer

(Supported backend: TensorFlow) pass parameters to another model

```
dp transfer [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
            [-r RAW_MODEL] [-O OLD_MODEL] [-o OUTPUT]
```

Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: "INFO"
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console
-r, --raw-model	the model receiving parameters Default: "raw_frozen_model.pb"
-O, --old-model	the model providing parameters Default: "old_frozen_model.pb"
-o, --output	the model after passing parameters Default: "frozen_model.pb"

10.3.2 train

train a model

```
dp train [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
         [-m {master,collect,workers}]
         [-i INIT_MODEL | -r RESTART | -f INIT_FRZ_MODEL | -t FINETUNE]
         [-o OUTPUT] [--skip-neighbor-stat] [--model-branch MODEL_BRANCH]
         [--force-load]
         INPUT
```

Positional Arguments

INPUT	the input parameter file in json or yaml format

Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: "INFO"
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console
-m, --mpi-log	Possible choices: master, collect, workers Set the manner of logging when running with MPI. 'master' logs only on main process, 'collect' broadcasts logs from workers to master and 'workers' means each process will output its own log Default: "master"
-i, --init-model	Initialize the model by the provided path prefix of checkpoint files.
-r, --restart	Restart the training from the provided path prefix of checkpoint files.
-f, --init-frz-model	Initialize the training from the frozen model.
-t, --finetune	Finetune the frozen pretrained model.
-o, --output	The output file of the parameters used in training. Default: "out.json"
--skip-neighbor-stat	Skip calculating neighbor statistics. Sel checking, automatic sel, and model compression will be disabled. Default: False
--model-branch	(Supported backend: PyTorch) Model branch chosen for fine-tuning if multi-task. If not specified, it will re-init the fitting net. Default: ""
--force-load	(Supported backend: PyTorch) Force load from ckpt, other missing tensors will init from scratch Default: False

examples:
dp train input.json dp train input.json –restart model.ckpt dp train input.json –init-model model.ckpt

10.3.3 freeze

freeze the model

```
dp freeze [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
          [-c CHECKPOINT_FOLDER] [-o OUTPUT] [-n NODE_NAMES] [-w NVNMD_WEIGHT]
          [--united-model] [--head HEAD]
```

Named Arguments

-v, --log-level
: Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0

  set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG

  Default: "INFO"

-l, --log-path
: set log file to log messages to disk, if not specified, the logs will only be output to console

-c, --checkpoint-folder, --checkpoint
: Path to checkpoint, either a folder containing checkpoint or the checkpoint prefix

  Default: "."

-o, --output
: Filename (prefix) of the output model file. TensorFlow backend: suffix is .pb; PyTorch backend: suffix is .pth

  Default: "frozen_model"

-n, --node-names
: (Supported backend: TensorFlow) the frozen nodes, if not set, determined from the model type

-w, --nvnmd-weight
: (Supported backend: TensorFlow) the name of weight file (.npy), if set, save the model's weight into the file

--united-model
: (Supported backend: TensorFlow) When in multi-task mode, freeze all nodes into one united model

  Default: False

--head
: (Supported backend: PyTorch) Task head to freeze if in multi-task mode.

examples:
    dp freeze dp freeze -o graph.pb

10.3.4 test

test the model

```
dp test [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH] [-m MODEL]
        [-s SYSTEM | -f DATAFILE] [-S SET_PREFIX] [-n NUMB_TEST]
        [-r RAND_SEED] [--shuffle-test] [-d DETAIL_FILE] [-a]
        [-i INPUT_SCRIPT] [--head HEAD]
```

Named Arguments

-v, --log-level
: Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0

  set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG

  Default: "INFO"

-l, --log-path
: set log file to log messages to disk, if not specified, the logs will only be output to console

-m, --model	Frozen model file (prefix) to import. TensorFlow backend: suffix is .pb; PyTorch backend: suffix is .pth. Default: "frozen_model"
-s, --system	The system dir. Recursively detect systems in this directory Default: "."
-f, --datafile	The path to file of test list.
-S, --set-prefix	[DEPRECATED] Deprecated argument.
-n, --numb-test	The number of data for test. 0 means all data. Default: 0
-r, --rand-seed	(Supported backend: TensorFlow) The random seed
--shuffle-test	Shuffle test data Default: False
-d, --detail-file	The prefix to files where details of energy, force and virial accuracy/accuracy per atom will be written
-a, --atomic	(Supported backend: TensorFlow) Test the accuracy of atomic label, i.e. energy / tensor (dipole, polar) Default: False
-i, --input_script	(Supported backend: PyTorch) The input script of the model
--head	(Supported backend: PyTorch) Task head to test if in multi-task mode.

examples:

dp test -m graph.pb -s /path/to/system -n 30

10.3.5 compress

(Supported backend: TensorFlow) compress a model

```
dp compress [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
            [-m {master,collect,workers}] [-i INPUT] [-o OUTPUT] [-s STEP]
            [-e EXTRAPOLATE] [-f FREQUENCY] [-c CHECKPOINT_FOLDER]
            [-t TRAINING_SCRIPT]
```

Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: "INFO"
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console

-m, --mpi-log
: Possible choices: master, collect, workers

  Set the manner of logging when running with MPI. 'master' logs only on main process, 'collect' broadcasts logs from workers to master and 'workers' means each process will output its own log

  Default: "master"

-i, --input
: The original frozen model, which will be compressed by the code

  Default: "frozen_model.pb"

-o, --output
: The compressed model

  Default: "frozen_model_compressed.pb"

-s, --step
: Model compression uses fifth-order polynomials to interpolate the embedding-net. It introduces two tables with different step size to store the parameters of the polynomials. The first table covers the range of the training data, while the second table is an extrapolation of the training data. The domain of each table is uniformly divided by a given step size. And the step(parameter) denotes the step size of the first table and the second table will use 10 * step as it's step size to save the memory. Usually the value ranges from 0.1 to 0.001. Smaller step means higher accuracy and bigger model size

  Default: 0.01

-e, --extrapolate
: The domain range of the first table is automatically detected by the code: [d_low, d_up]. While the second table ranges from the first table's upper boundary(d_up) to the extrapolate(parameter) * d_up: [d_up, extrapolate * d_up]

  Default: 5

-f, --frequency
: The frequency of tabulation overflow check(Whether the input environment matrix overflow the first or second table range). By default do not check the overflow

  Default: -1

-c, --checkpoint-folder
: path to checkpoint folder

  Default: "model-compression"

-t, --training-script
: The training script of the input frozen model

examples:

dp compress dp compress -i graph.pb -o compressed.pb

10.3.6 doc-train-input

print the documentation (in rst format) of input training parameters.

```
dp doc-train-input [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
                   [--out-type {rst,json}]
```

Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: "INFO"
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console
--out-type	Possible choices: rst, json The output type Default: "rst"

10.3.7 model-devi

calculate model deviation

```
dp model-devi [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
              [-m MODELS [MODELS ...]] [-s SYSTEM] [-S SET_PREFIX] [-o OUTPUT]
              [-f FREQUENCY] [--real_error] [--atomic] [--relative RELATIVE]
              [--relative_v RELATIVE_V]
```

Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: "INFO"
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console
-m, --models	Frozen models file (prefix) to import. TensorFlow backend: suffix is .pb; PyTorch backend: suffix is .pth. Default: ['graph.000', 'graph.001', 'graph.002', 'graph.003']
-s, --system	The system directory. Recursively detect systems in this directory. Default: "."
-S, --set-prefix	[DEPRECATED] Deprecated argument.
-o, --output	The output file for results of model deviation Default: "model_devi.out"
-f, --frequency	The trajectory frequency of the system Default: 1
--real_error	Calculate the RMS real error of the model. The real data should be given in the systems. Default: False

--atomic	Print the force model deviation of each atom. Default: False
--relative	Calculate the relative model deviation of force. The level parameter for computing the relative model deviation of the force should be given.
--relative_v	Calculate the relative model deviation of virial. The level parameter for computing the relative model deviation of the virial should be given.

examples:
dp model-devi -m graph.000.pb graph.001.pb graph.002.pb graph.003.pb -s ./data -o model_devi.out

10.3.8 convert-from

(Supported backend: TensorFlow) convert lower model version to supported version

```
dp convert-from [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
                [-i INPUT_MODEL] [-o OUTPUT_MODEL]
                [{auto,0.12,1.0,1.1,1.2,1.3,2.0,pbtxt}]
```

Positional Arguments

FROM	Possible choices: auto, 0.12, 1.0, 1.1, 1.2, 1.3, 2.0, pbtxt The original model compatibility Default: “auto”

Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: “INFO”
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console
-i, --input-model	the input model Default: “frozen_model.pb”
-o, --output-model	the output model If OUTPUT_MODEL ends with ‘.pbtxt’, the provided model will be converted to pbtxt format, without version conversion. Default: “convert_out.pb”

examples:
dp convert-from -i graph.pb -o graph_new.pb dp convert-from auto -i graph.pb -o graph_new.pb dp convert-from 1.0 -i graph.pb -o graph_new.pb

10.3.9 neighbor-stat

Calculate neighbor statistics

```
dp neighbor-stat [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
                 [-s SYSTEM] -r RCUT -t TYPE_MAP [TYPE_MAP ...] [--mixed-type]
```

Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: "INFO"
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console
-s, --system	The system dir. Recursively detect systems in this directory Default: "."
-r, --rcut	cutoff radius
-t, --type-map	type map
--mixed-type, --one-type	treat all types as a single type. Used with se_atten descriptor. Default: False

examples:
dp neighbor-stat -s data -r 6.0 -t O H

10.3.10 train-nvnmd

(Supported backend: TensorFlow) train nvnmd model

```
dp train-nvnmd [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
               [-i INIT_MODEL] [-r RESTART] [-s {s1,s2}]
               [--skip-neighbor-stat]
               INPUT
```

Positional Arguments

INPUT	the input parameter file in json format

Named Arguments

-v, --log-level
: Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0

  set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG

  Default: "INFO"

-l, --log-path
: set log file to log messages to disk, if not specified, the logs will only be output to console

-i, --init-model
: Initialize the model by the provided path prefix of checkpoint files.

-r, --restart
: Restart the training from the provided prefix of checkpoint files.

-s, --step
: Possible choices: s1, s2

  steps to train model of NVNMD: s1 (train CNN), s2 (train QNN)

  Default: "s1"

--skip-neighbor-stat
: Skip calculating neighbor statistics. Sel checking, automatic sel, and model compression will be disabled.

  Default: False

examples:
: dp train-nvnmd input_cnn.json -s s1 dp train-nvnmd input_qnn.json -s s2 dp train-nvnmd input_cnn.json -s s1 –restart model.ckpt dp train-nvnmd input_cnn.json -s s2 –init-model model.ckpt

10.3.11 gui

Serve DP-GUI.

```
dp gui [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH] [-p PORT]
       [--bind_all]
```

Named Arguments

-v, --log-level
: Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0

  set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG

  Default: "INFO"

-l, --log-path
: set log file to log messages to disk, if not specified, the logs will only be output to console

-p, --port
: The port to serve DP-GUI on.

  Default: 6042

--bind_all
: Serve on all public interfaces. This will expose your DP-GUI instance to the network on both IPv4 and IPv6 (where available).

  Default: False

10.3.12 convert-backend

Convert model to another backend.

```
dp convert-backend [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
                   INPUT OUTPUT
```

Positional Arguments

INPUT	The input model file.
OUTPUT	The output model file.

Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: "INFO"
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console

examples:
dp convert-backend model.pb model.pth dp convert-backend model.pb model.dp

Use –tf or –pt to choose the backend:
dp –tf train input.json dp –pt train input.json

CHAPTER

ELEVEN

INTEGRATE WITH THIRD-PARTY PACKAGES

Note that the model for inference is required to be compatible with the DeePMD-kit package. See Model compatibility for details.

11.1 Use deep potential with dpdata

DeePMD-kit provides a driver for dpdata >=0.2.7 via the plugin mechanism, making it possible to call the `predict` method for `System` class:

```
import dpdata

dsys = dpdata.LabeledSystem("OUTCAR")
dp_sys = dsys.predict("frozen_model_compressed.pb", driver="dp")
```

By inferring with the DP model `frozen_model_compressed.pb`, dpdata will generate a new labeled system `dp_sys` with inferred energies, forces, and virials.

11.2 Use deep potential with ASE

Deep potential can be set up as a calculator with ASE to obtain potential energies and forces.

```
from ase import Atoms
from deepmd.calculator import DP

water = Atoms(
    "H2O",
    positions=[(0.7601, 1.9270, 1), (1.9575, 1, 1), (1.0, 1.0, 1.0)],
    cell=[100, 100, 100],
    calculator=DP(model="frozen_model.pb"),
)
print(water.get_potential_energy())
print(water.get_forces())
```

Optimization is also available:

```
from ase.optimize import BFGS

dyn = BFGS(water)
dyn.run(fmax=1e-6)
print(water.get_positions())
```

11.3 Run MD with LAMMPS

11.3.1 units

All units in LAMMPS except `lj` are supported. `lj` is not supported.

The most commonly used units are `metal`, since the internal units of distance, energy, force, and charge in DeePMD-kit are `\AA`, `eV`, `eV / \AA`, and `proton charge`, respectively. These units are consistent with the `metal` units in LAMMPS.

If one wants to use other units like `real` or `si`, it is welcome to do so. There is no need to do the unit conversion mannualy. The unit conversion is done automatically by LAMMPS.

The only thing that one needs to take care is the unit of the output of `compute deeptensor/atom`. Working with `metal` units for `compute deeptensor/atom` is totally fine, since there is no unit conversion. For other unit styles, we currently assume that the output of the `compute deeptensor/atom` command has the unit of distance and have applied the unit conversion factor of distance. If a user wants to infer quantities with units other than distance, the user is encouraged to open a GitHub feature request, so that the unit conversion factor can be added.

11.3.2 Enable DeePMD-kit plugin (plugin mode)

If you are using the plugin mode, enable DeePMD-kit package in LAMMPS with `plugin` command:

```
plugin load libdeepmd_lmp.so
```

After LAMMPS version `patch_24Mar2022`, another way to load plugins is to set the environmental variable `LAMMPS_PLUGIN_PATH`:

```
LAMMPS_PLUGIN_PATH=$deepmd_root/lib/deepmd_lmp
```

where `$deepmd_root` is the directory to install C++ interface.

The built-in mode doesn't need this step.

11.3.3 pair_style `deepmd`

The DeePMD-kit package provides the pair_style `deepmd`

```
pair_style deepmd models ... keyword value ...
```

- deepmd = style of this pair_style
- models = frozen model(s) to compute the interaction. If multiple models are provided, then only the first model serves to provide energy and force prediction for each timestep of molecular dynamics, and the model deviation will be computed among all models every `out_freq` timesteps.
- keyword = out_file or out_freq or fparam or fparam_from_compute or aparam_from_compute or atomic or relative or relative_v or aparam or ttm

Examples

```
pair_style deepmd graph.pb
pair_style deepmd graph.pb fparam 1.2
pair_style deepmd graph_0.pb graph_1.pb graph_2.pb out_file md.out out_freq 10 atomic relative 1.0
pair_coeff * * O H

pair_style deepmd cp.pb fparam_from_compute TEMP
compute    TEMP all temp

pair_style deepmd ener.pb aparam_from_compute 1
compute    1 all ke/atom
```

Description

Evaluate the interaction of the system by using Deep Potential or Deep Potential Smooth Edition. It is noticed that deep potential is not a "pairwise" interaction, but a multi-body interaction.

This pair style takes the deep potential defined in a model file that usually has the .pb extension. The model can be trained and frozen by package DeePMD-kit, which can have either double or single float precision interface.

The model deviation evalulates the consistency of the force predictions from multiple models. By default, only the maximal, minimal and average model deviations are output. If the key `atomic` is set, then the model deviation of force prediction of each atom will be output. The unit follows LAMMPS units and the scale factor is not applied.

By default, the model deviation is output in absolute value. If the keyword `relative` is set, then the relative model deviation of the force will be output, including values output by the keyword `atomic`. The relative model deviation of the force on atom i is defined by

$$E_{f_i} = \frac{|D_{f_i}|}{|f_i| + l}$$

where D_{f_i} is the absolute model deviation of the force on atom i, f_i is the norm of the force and l is provided as the parameter of the keyword `relative`. If the keyword `relative_v` is set, then the relative model deviation of the virial will be output instead of the absolute value, with the same definition of that of the force:

$$E_{v_i} = \frac{|D_{v_i}|}{|v_i| + l}$$

If the keyword `fparam` is set, the given frame parameter(s) will be fed to the model. If the keyword `fparam_from_compute` is set, the global parameter(s) from compute command (e.g., temperature from compute temp command) will be fed to the model as the frame parameter(s). If the keyword `aparam_from_compute` is set, the atomic parameter(s) from compute command (e.g., per-atom translational kinetic energy from compute ke/atom command) will be fed to the model as the atom parameter(s). If the keyword `aparam` is set, the given atomic parameter(s) will be fed to the model, where each atom is assumed to have the same atomic parameter(s). If the keyword `ttm` is set, electronic temperatures from fix ttm command will be fed to the model as the atomic parameters.

Only a single `pair_coeff` command is used with the deepmd style which specifies atom names. These are mapped to LAMMPS atom types (integers from 1 to Ntypes) by specifying Ntypes additional arguments after `* *` in the `pair_coeff` command. If atom names are not set in the `pair_coeff` command, the training parameter type_map will be used by default. If a mapping value is specified as `NULL`, the mapping is not performed. This can be used when a deepmd potential is used as part of the hybrid pair style. The `NULL` values are placeholders for atom types that will be used with other potentials. If the training parameter

type_map is not set, atom names in the `pair_coeff` command cannot be set. In this case, atom type indexes in *`type.raw`* (integers from 0 to Ntypes-1) will map to LAMMPS atom types.

Spin is specified by keywords `virtual_len` and `spin_norm`. If the keyword `virtual_len` is set, the distance between virtual atom and its corresponding real atom for each type of magnetic atoms will be fed to the model as the spin parameters. If the keyword `spin_norm` is set, the magnitude of the magnetic moment for each type of magnetic atoms will be fed to the model as the spin parameters.

Restrictions

- The `deepmd` pair style is provided in the USER-DEEPMD package, which is compiled from the DeePMD-kit, visit the DeePMD-kit website for more information.

11.3.4 Compute tensorial properties

The DeePMD-kit package provides the compute `deeptensor/atom` for computing atomic tensorial properties.

```
compute ID group-ID deeptensor/atom model_file
```

- ID: user-assigned name of the computation
- group-ID: ID of the group of atoms to compute
- deeptensor/atom: the style of this compute
- model_file: the name of the binary model file.

At this time, the training parameter type_map will be mapped to LAMMPS atom types.

Examples

```
compute         dipole all deeptensor/atom dipole.pb
```

The result of the compute can be dumped to trajectory file by

```
dump            1 all custom 100 water.dump id type c_dipole[1] c_dipole[2] c_dipole[3]
```

Restrictions

- The `deeptensor/atom` compute is provided in the USER-DEEPMD package, which is compiled from the DeePMD-kit, visit the DeePMD-kit website for more information.
- For the issue of using a unit style for `compute deeptensor/atom`, refer to the discussions in units of this page.

11.3.5 Long-range interaction

The reciprocal space part of the long-range interaction can be calculated by LAMMPS command `kspace_style`. To use it with DeePMD-kit, one writes

```
pair_style        deepmd graph.pb
pair_coeff  * *
kspace_style        pppm 1.0e-5
kspace_modify        gewald 0.45
```

Please notice that the DeePMD does nothing to the direct space part of the electrostatic interaction, because this part is assumed to be fitted in the DeePMD model (the direct space cut-off is thus the cut-off of the DeePMD model). The splitting parameter `gewald` is modified by the `kspace_modify` command.

11.3.6 Use of the centroid/stress/atom to get the full 3x3 "atomic-virial"

The DeePMD-kit also allows the computation of per-atom stress tensor defined as:

$$dvatom = -\sum_m (\mathrm{r}_n - \mathrm{r}_m)\frac{de_m}{d\mathrm{r}_n}$$

Where r_n is the atomic position of nth atom, v_n velocity of the atom and $\frac{de_m}{d\mathrm{r}_n}$ the derivative of the atomic energy.

In LAMMPS one can get the per-atom stress using the command `centroid/stress/atom`:

```
compute ID group-ID centroid/stress/atom NULL virial
```

see LAMMPS doc page for more details on the meaning of the keywords.

Changed in version v2.2.3: v2.2.2 or previous versions passed per-atom stress (`cvatom`) with the per-atom pressure tensor, which is inconsistent with LAMMPS's definition. LAMMPS defines per-atom stress as the negative of the per-atom pressure tensor. Such behavior is corrected in v2.2.3.

Examples

In order of computing the 9-component per-atom stress

```
compute stress all centroid/stress/atom NULL virial
```

Thus `c_stress` is an array with 9 components in the order `xx,yy,zz,xy,xz,yz,yx,zx,zy`.

If you use this feature please cite D. Tisi, L. Zhang, R. Bertossa, H. Wang, R. Car, S. Baroni - arXiv preprint arXiv:2108.10850, 2021

11.3.7 Computation of heat flux

Using a per-atom stress tensor one can, for example, compute the heat flux defined as:

$$\mathrm{J} = \sum_n e_n \mathrm{v}_n + \sum_{n,m} (\mathrm{r}_m - \mathrm{r}_n)\frac{de_m}{d\mathrm{r}_n}\mathrm{v}_n$$

to compute the heat flux with LAMMPS:

```
compute ke_ID all ke/atom
compute pe_ID all pe/atom
compute stress_ID group-ID centroid/stress/atom NULL virial
compute flux_ID all heat/flux ke_ID pe_ID stress_ID
```

Examples

```
compute ke all ke/atom
compute pe all pe/atom
compute stress all centroid/stress/atom NULL virial
compute flux all heat/flux ke pe stress
```

`c_flux` is a global vector of length 6. The first three components are the x, y and z components of the full heat flux vector. The others are the components of the so-called convective portion, see LAMMPS doc page for more detailes.

If you use these features please cite D. Tisi, L. Zhang, R. Bertossa, H. Wang, R. Car, S. Baroni - arXiv preprint arXiv:2108.10850, 2021

11.4 Run path-integral MD with i-PI

The i-PI works in a client-server model. The i-PI provides the server for integrating the replica positions of atoms, while the DeePMD-kit provides a client named `dp_ipi` that computes the interactions (including energy, forces and virials). The server and client communicate via the Unix domain socket or the Internet socket. Installation instructions for i-PI can be found here. The client can be started by

```
i-pi input.xml &
dp_ipi water.json
```

It is noted that multiple instances of the client allow for computing, in parallel, the interactions of multiple replicas of the path-integral MD.

`water.json` is the parameter file for the client `dp_ipi`, and an example is provided:

```
{
  "verbose": false,
  "use_unix": true,
  "port": 31415,
  "host": "localhost",
  "graph_file": "graph.pb",
  "coord_file": "conf.xyz",
  "atom_type": {
    "OW": 0,
    "HW1": 1,
    "HW2": 1
  }
}
```

The option **`use_unix`** is set to `true` to activate the Unix domain socket, otherwise, the Internet socket is used.

The option **`port`** should be the same as that in input.xml:

```
<port>31415</port>
```

The option **graph_file** provides the file name of the frozen model. The model can have either double or single float precision interface.

The dp_ipi gets the atom names from an XYZ file provided by **coord_file** (meanwhile ignores all coordinates in it) and translates the names to atom types by rules provided by **atom_type**.

11.5 Running MD with GROMACS

11.5.1 DP/MM Simulation

This part gives a simple tutorial on how to run a DP/MM simulation for methane in water, which means using DP for methane and TIP3P for water. All relevant files can be found in examples/methane.

Topology Preparation

Similar to QM/MM simulation, the internal interactions (including bond, angle, dihedrals, LJ, Columb) of the region described by a neural network potential (NNP) have to be turned off. In GROMACS, bonded interactions can be turned off by modifying [bonds], [angles], [dihedrals] and [pairs] sections. And LJ and Columb interactions must be turned off by [exclusions] section.

For example, if one wants to simulate ethane in water, using DeepPotential for methane and TIP3P for water, the topology of methane should be like the following (as presented in examples/methane/methane.itp):

```
[ atomtypes ]
;name btype  mass  charge ptype    sigma  epsilon
  c3    c3   0.0     0.0     A 0.339771 0.451035
  hc    hc   0.0     0.0     A 0.260018 0.087027

[ moleculetype ]
;name            nrexcl
 methane          3

[ atoms ]
; nr type  resnr residue atom  cgnr  charge   mass
  1   c3      1     MOL   C1     1 -0.1068 12.010
  2   hc      1     MOL   H1     2  0.0267  1.008
  3   hc      1     MOL   H2     3  0.0267  1.008
  4   hc      1     MOL   H3     4  0.0267  1.008
  5   hc      1     MOL   H4     5  0.0267  1.008

[ bonds ]
; i  j  func  b0  kb
 1  2     5
 1  3     5
 1  4     5
 1  5     5

[ exclusions ]
; ai  aj1  aj2  aj3  aj4
  1    2    3    4    5
  2    1    3    4    5
  3    1    2    4    5
  4    1    2    3    5
  5    1    2    3    4
```

For comparison, the original topology file generated by `acpype` will be:

```
; methane_GMX.itp created by acpype (v: 2021-02-05T22:15:50CET) on Wed Sep  8 01:21:53 2021

[ atomtypes ]
;name   bond_type     mass     charge   ptype   sigma         epsilon       Amb
 c3       c3          0.00000  0.00000   A     3.39771e-01   4.51035e-01 ; 1.91  0.1078
 hc       hc          0.00000  0.00000   A     2.60018e-01   8.70272e-02 ; 1.46  0.0208

[ moleculetype ]
;name            nrexcl
 methane          3

[ atoms ]
;   nr  type  resi  res  atom  cgnr     charge      mass       ; qtot   bond_type
     1   c3     1   MOL    C1    1    -0.106800     12.01000 ; qtot -0.107
     2   hc     1   MOL    H1    2     0.026700      1.00800 ; qtot -0.080
     3   hc     1   MOL    H2    3     0.026700      1.00800 ; qtot -0.053
     4   hc     1   MOL    H3    4     0.026700      1.00800 ; qtot -0.027
     5   hc     1   MOL    H4    5     0.026700      1.00800 ; qtot 0.000

[ bonds ]
;   ai     aj funct   r             k
     1      2   1    1.0970e-01    3.1455e+05 ;     C1 - H1
     1      3   1    1.0970e-01    3.1455e+05 ;     C1 - H2
     1      4   1    1.0970e-01    3.1455e+05 ;     C1 - H3
     1      5   1    1.0970e-01    3.1455e+05 ;     C1 - H4

[ angles ]
;   ai     aj     ak    funct   theta         cth
     2      1      3      1    1.0758e+02    3.2635e+02 ;     H1 - C1     - H2
     2      1      4      1    1.0758e+02    3.2635e+02 ;     H1 - C1     - H3
     2      1      5      1    1.0758e+02    3.2635e+02 ;     H1 - C1     - H4
     3      1      4      1    1.0758e+02    3.2635e+02 ;     H2 - C1     - H3
     3      1      5      1    1.0758e+02    3.2635e+02 ;     H2 - C1     - H4
     4      1      5      1    1.0758e+02    3.2635e+02 ;     H3 - C1     - H4
```

DeepMD Settings

Before running simulations, we need to tell GROMACS to use DeepPotential by setting the environment variable `GMX_DEEPMD_INPUT_JSON`:

```
export GMX_DEEPMD_INPUT_JSON=input.json
```

Then, in your working directories, we have to write `input.json` file:

```
{
  "graph_file": "/path/to/graph.pb",
  "type_file": "type.raw",
  "index_file": "index.raw",
  "lambda": 1.0,
  "pbc": false
}
```

Here is an explanation for these settings:

- `graph_file` : The graph file (with suffix .pb) generated by `dp freeze` command

- `type_file` : File to specify DP atom types (in space-separated format). Here, `type.raw` looks like

```
1 0 0 0 0
```

- `index_file` : File containing indices of DP atoms (in space-separated format), which should be consistent with the indices' order in .gro file but starting from zero. Here, `index.raw` looks like

```
0 1 2 3 4
```

- `lambda`: Optional, default 1.0. Used in alchemical calculations.
- `pbc`: Optional, default true. If true, the GROMACS periodic condition is passed to DeepMD.

Run Simulation

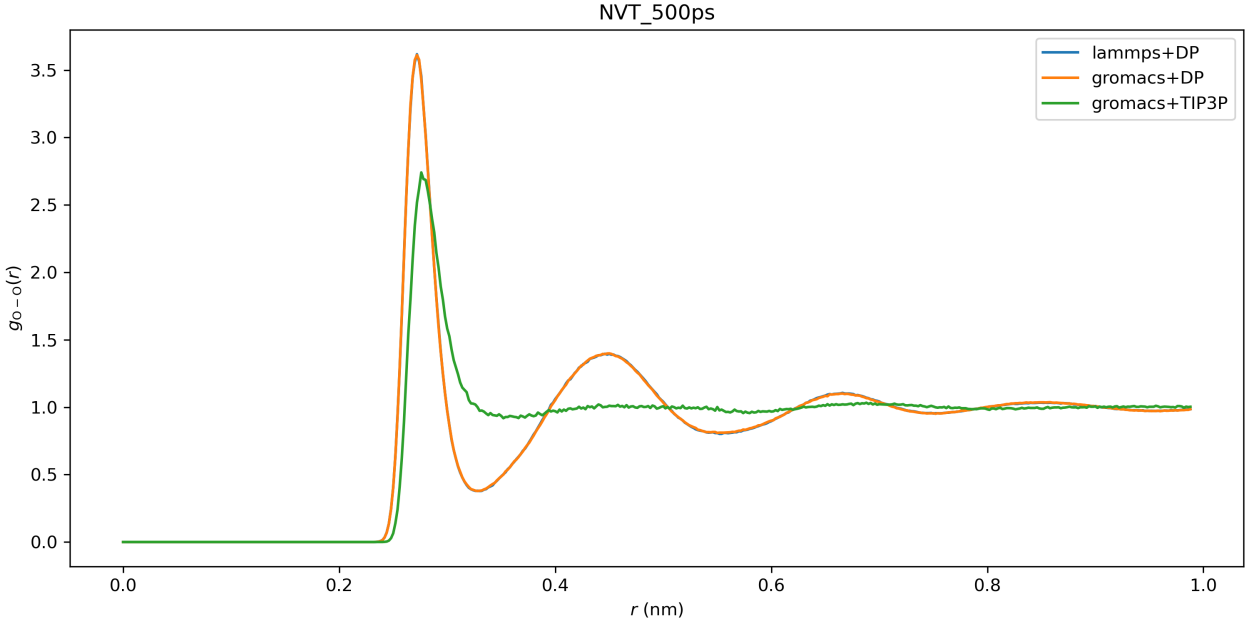
Finally, you can run GROMACS using `gmx mdrun` as usual.

11.5.2 All-atom DP Simulation

This part gives an example of how to simulate all atoms described by a DeepPotential with Gromacs, taking water as an example. Instead of using `[ exclusions ]` to turn off the non-bonded energies, we can simply do this by setting LJ parameters (i.e. epsilon and sigma) and partial charges to 0, as shown in `examples/water/gmx/water.top`:

```
[ atomtypes ]
; name      at.num  mass     charge ptype  sigma       epsilon
HW           1       1.008   0.0000  A   0.00000e+00  0.00000e+00
OW           8      16.00    0.0000  A   0.00000e+00  0.00000e+00
```

As mentioned in the above section, `input.json` and relevant files (`index.raw`, `type.raw`) should also be created. Then, we can start the simulation under the NVT ensemble and plot the radial distribution function (RDF) by `gmx rdf` command. We can see that the RDF given by Gromacs+DP matches perfectly with Lammps+DP, which further provides an evidence on the validity of our simulation.



However, we still recommend you run an all-atom DP simulation using LAMMPS since it is more stable and efficient.

11.6 Interfaces out of DeePMD-kit

The codes of the following interfaces are not a part of the DeePMD-kit package and maintained by other repositories. We list these interfaces here for user convenience.

11.6.1 OpenMM plugin for DeePMD-kit

An OpenMM plugin is provided from JingHuangLab/openmm_deepmd_plugin, written by the Huang Lab at Westlake University.

11.6.2 Amber interface to DeePMD-kit

Starting from AmberTools24, `sander` includes an interface to the DeePMD-kit, which implements the Deep Potential Range Corrected (DPRc) correction. The DPRc model and the interface were developed by the York Lab from Rutgers University. More details are available in

- Amber Reference Manuals, providing documentation for how to enable the interface and the `&dprc` namelist;
- GitLab RutgersLBSR/AmberDPRc, providing examples mdin files;
- DP-Amber, a tiny tool to convert Amber trajectory to DPRc training data;
- The original DPRc paper.

11.6.3 CP2K interface to DeePMD-kit

CP2K v2024.2 adds an interface to the DeePMD-kit for molecular dynamics. Read the CP2K manual for details.

11.6.4 DP-GEN

DP-GEN provides a workflow to generate accurate DP models by calling DeePMD-kit's command line interface (CLI) in the local or remote server. Details can be found in this paper.

11.6.5 MLatom

Mlatom provides an interface to the DeePMD-kit within MLatom's workflow by calling DeePMD-kit's CLI. Details can be found in this paper.

11.6.6 ABACUS

ABACUS can run molecular dynamics with a DP model. User is required to build ABACUS with DeePMD-kit.

CHAPTER

TWELVE

USE NVNMD

NVNMD stands for non-von Neumann molecular dynamics.

In this section, we will introduce how to use it.

- Use NVNMD

CHAPTER

THIRTEEN

FAQS

As a consequence of differences in computers or systems, problems may occur. Some common circumstances are listed as follows. In addition, some frequently asked questions are listed as follows. If other unexpected problems occur, you're welcome to contact us for help.

13.1 How to tune Fitting/embedding-net size ?

Here are some test forms on fitting-net size tuning or embedding-net size tuning performed on several different systems.

13.1.1 Al2O3

Fitting net size tuning form on Al2O3: (embedding-net size: [25,50,100])

Fitting-net size	Energy L2err(eV)	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[240,240,240]	1.742252e-02	7.259383e-05	4.014115e-02
[80,80,80]	1.799349e-02	7.497287e-05	4.042977e-02
[40,40,40]	1.799036e-02	7.495984e-05	4.068806e-02
[20,20,20]	1.834032e-02	7.641801e-05	4.094784e-02
[10,10,10]	1.913058e-02	7.971073e-05	4.154775e-02
[5,5,5]	1.932914e-02	8.053808e-05	4.188052e-02
[4,4,4]	1.944832e-02	8.103467e-05	4.217826e-02
[3,3,3]	2.068631e-02	8.619296e-05	4.300497e-02
[2,2,2]	2.267962e-02	9.449840e-05	4.413609e-02
[1,1,1]	2.813596e-02	1.172332e-04	4.781115e-02
[]	3.135002e-02	1.306251e-04	5.373120e-02

[] means no hidden layer, but there is still a linear output layer. This situation is equal to the linear regression.

Embedding net size tuning form on Al2O3: (Fitting-net size: [240,240,240])

Embedding-net size	Energy L2err(eV)	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[25,50,100]	1.742252e-02	7.259383e-05	4.014115e-02
[10,20,40]	2.909990e-02	1.212496e-04	4.734667e-02
[5,10,20]	3.357767e-02	1.399070e-04	5.706385e-02
[4,8,16]	6.060367e-02	2.525153e-04	7.333304e-02
[3,6,12]	5.656043e-02	2.356685e-04	7.793539e-02
[2,4,8]	5.277023e-02	2.198759e-04	7.459995e-02
[1,2,4]	1.302282e-01	5.426174e-04	9.672238e-02

13.1.2 Cu

Fitting net size tuning form on Cu: (embedding-net size: [25,50,100])

Fitting-net size	Energy L2err(eV)	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[240,240,240]	4.135548e-02	1.615449e-04	8.940946e-02
[20,20,20]	4.323858e-02	1.689007e-04	8.955762e-02
[10,10,10]	4.399364e-02	1.718502e-04	8.962891e-02
[5,5,5]	4.468404e-02	1.745470e-04	8.970111e-02
[4,4,4]	4.463580e-02	1.743586e-04	8.972011e-02
[3,3,3]	4.493758e-02	1.755374e-04	8.971303e-02
[2,2,2]	4.500736e-02	1.758100e-04	8.973878e-02
[1,1,1]	4.542073e-02	1.774247e-04	8.964761e-02
[]	4.545168e-02	1.775456e-04	8.983201e-02

Embedding net size tuning form on Cu: (Fitting-net size: [240,240,240])

Embedding-net size	Energy L2err(eV)	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[25,50,100]	4.135548e-02	1.615449e-04	8.940946e-02
[20,40,80]	4.203562e-02	1.642016e-04	8.925881e-02
[15,30,60]	4.146672e-02	1.619794e-04	8.936911e-02
[10,20,40]	4.263060e-02	1.665258e-04	8.955818e-02
[5,10,20]	4.994913e-02	1.951138e-04	9.007786e-02
[4,8,16]	1.022157e-01	3.992802e-04	9.532119e-02
[3,6,12]	1.362098e-01	5.320695e-04	1.073860e-01
[2,4,8]	7.061800e-02	2.758515e-04	9.126418e-02
[1,2,4] && seed = 1	9.843161e-02	3.844985e-04	9.348505e-02
[1,2,4] && seed = 2	9.404335e-02	3.673568e-04	9.304089e-02
[1,2,4] && seed = 3	1.508016e-01	5.890688e-04	1.382356e-01
[1,2,4] && seed = 4	9.686949e-02	3.783965e-04	9.294820e-02

13.1.3 Water

Fitting net size tuning form on water: (embedding-net size: [25,50,100])

Fitting-net size	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[240,240,240]	9.1589E-04	5.1540E-02
[200,200,200]	9.3221E-04	5.2366E-02
[160,160,160]	9.4274E-04	5.3403E-02
[120,120,120]	9.5407E-04	5.3093E-02
[80,80,80]	9.4605E-04	5.3402E-02
[40,40,40]	9.8533E-04	5.5790E-02
[20,20,20]	1.0057E-03	5.8232E-02
[10,10,10]	1.0466E-03	6.2279E-02
[5,5,5]	1.1154E-03	6.7994E-02
[4,4,4]	1.1289E-03	6.9613E-02
[3,3,3]	1.2368E-03	7.9786E-02
[2,2,2]	1.3558E-03	9.7042E-02
[1,1,1]	1.4633E-03	1.1265E-01
[]	1.5193E-03	1.2136E-01

Embedding net size tuning form on water: (Fitting-net size: [240,240,240])

Embedding-net size	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[25,50,100]	9.1589E-04	5.1540E-02
[20,40,80]	9.5080E-04	5.3593E-02
[15,30,60]	9.7996E-04	5.6338E-02
[10,20,40]	1.0353E-03	6.2776E-02
[5,10,20]	1.1254E-03	7.3195E-02
[4,8,16]	1.2495E-03	8.0371E-02
[3,6,12]	1.3604E-03	9.9883E-02
[2,4,8]	1.4358E-03	9.7389E-02
[1,2,4]	2.1765E-03	1.7276E-01

13.1.4 Mg-Al

Fitting net size tuning form on Mg-Al: (embedding-net size: [25,50,100])

Fitting-net size	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[240,240,240]	3.9606e-03	1.6289e-02
[200,200,200]	3.9449e-03	1.6471e-02
[160,160,160]	4.0947e-03	1.6413e-02
[120,120,120]	3.9234e-03	1.6283e-02
[80,80,80]	3.9758e-03	1.6506e-02
[40,40,40]	3.9142e-03	1.6348e-02
[20,20,20]	4.1302e-03	1.7006e-02
[10,10,10]	4.3433e-03	1.7524e-02
[5,5,5]	5.3154e-03	1.9716e-02
[4,4,4]	5.4210e-03	1.9710e-02
[2,2,2]	6.2667e-03	2.2568e-02
[1,1,1]	7.3676e-03	2.6375e-02
[]	7.3999e-03	2.6097e-02

Embedding net size tuning form on Mg-Al: (Fitting-net size: [240,240,240])

Embedding-net size	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[25,50,100]	3.9606e-03	1.6289e-02
[20,40,80]	4.0292e-03	1.6555e-02
[15,30,60]	4.1743e-03	1.7026e-02
[10,20,40]	4.8138e-03	1.8516e-02
[5,10,20]	5.6052e-03	2.0709e-02
[4,8,16]	6.1335e-03	2.1450e-02
[3,6,12]	6.6469e-03	2.3003e-02
[2,4,8]	6.8222e-03	2.6318e-02
[1,2,4]	1.0678e-02	3.9559e-02

13.2 How to control the parallelism of a job?

DeePMD-kit has three levels of parallelism. To get the best performance, one should control the number of threads used by DeePMD-kit. One should make sure the product of the parallel numbers is less than or equal to the number of cores available.

13.2.1 MPI (optional)

Parallelism for MPI is optional and used for multiple nodes, multiple GPU cards, or sometimes multiple CPU cores.

To enable MPI support for training, one should install horovod in advance. Note that the parallelism mode is data parallelism, so it is not expected to see the training time per batch decreases.

MPI support for inference is not directly supported by DeePMD-kit, but indirectly supported by the third-party software. For example, LAMMPS enables running simulations in parallel using the MPI parallel communication standard with distributed data. That software has to build against MPI.

Set the number of processes with:

```
mpirun -np $num_nodes dp
```

Note that `mpirun` here should be the same as the MPI used to build software. For example, one can use `mpirun --version` and `lmp -h` to see if `mpirun` and LAMMPS has the same MPI version.

Sometimes, `$num_nodes` and the nodes information can be directly given by the HPC scheduler system, if the MPI used here is the same as the MPI used to build the scheduler system. Otherwise, one have to manually assign these information.

13.2.2 Parallelism between independent operators

For CPU devices, TensorFlow and PyTorch use multiple streams to run independent operators (OP).

```
export DP_INTER_OP_PARALLELISM_THREADS=3
```

However, for GPU devices, TensorFlow uses only one compute stream and multiple copy streams. Note that some of DeePMD-kit OPs do not have GPU support, so it is still encouraged to set environmental variables even if one has a GPU.

13.2.3 Parallelism within an individual operators

For CPU devices, `DP_INTRA_OP_PARALLELISM_THREADS` controls parallelism within TensorFlow (when TensorFlow is built against Eigen) and PyTorch native OPs.

```
export DP_INTRA_OP_PARALLELISM_THREADS=2
```

`OMP_NUM_THREADS` is the number of threads for OpenMP parallelism. It controls parallelism within TensorFlow (when TensorFlow is built upon Intel OneDNN) and PyTorch (when PyTorch is built upon OpenMP) native OPs and DeePMD-kit custom CPU OPs. It may also control parallelism for NumPy when NumPy is built against OpenMP, so one who uses GPUs for training should also care this environmental variable.

```
export OMP_NUM_THREADS=2
```

There are several other environmental variables for OpenMP, such as `KMP_BLOCKTIME`.

TensorFlow

See Intel documentation for detailed information.

PyTorch

See PyTorch documentation for detailed information.

13.2.4 Tune the performance

There is no one general parallel configuration that works for all situations, so you are encouraged to tune parallel configurations yourself after empirical testing.

Here are some empirical examples. If you wish to use 3 cores of 2 CPUs on one node, you may set the environmental variables and run DeePMD-kit as follows:

```
export OMP_NUM_THREADS=3
export DP_INTRA_OP_PARALLELISM_THREADS=3
export DP_INTER_OP_PARALLELISM_THREADS=2
dp train input.json
```

For a node with 128 cores, it is recommended to start with the following variables:

```
export OMP_NUM_THREADS=16
export DP_INTRA_OP_PARALLELISM_THREADS=16
export DP_INTER_OP_PARALLELISM_THREADS=8
```

Again, in general, one should make sure the product of the parallel numbers is less than or equal to the number of cores available. In the above case, $16 \times 8 = 128$, so threads will not compete with each other.

13.3 Do we need to set rcut < half boxsize?

When seeking the neighbors of atom i under periodic boundary conditions, DeePMD-kit considers all j atoms within cutoff rcut from atom i in all mirror cells.

So, there is no limitation on the setting of rcut.

PS: The reason why some software requires rcut < half box size is that they only consider the nearest mirrors from the center cell. DeePMD-kit is different from them.

13.4 How to set sel?

`sel` is short for "selected number of atoms in `rcut`".

`sel_a[i]` is a list of integers. The length of the list should be the same as the number of atom types in the system.

`sel_a[i]` gives the number of the selected number of type `i` neighbors within `rcut`. To ensure that the results are strictly accurate, `sel_a[i]` should be larger than the largest number of type `i` neighbors in the `rcut`.

However, the computation overhead increases with `sel_a[i]`, therefore, `sel_a[i]` should be as small as possible.

The setting of `sel_a[i]` should balance the above two considerations.

13.5 Installation

13.5.1 Inadequate versions of gcc/g++

Sometimes you may use a gcc/g++ of version < 4.8. In this way, you can still compile all the parts of TensorFlow and most of the parts of DeePMD-kit, but i-Pi and GROMACS plugins will be disabled automatically. Or if you have a gcc/g++ of version > 4.8, say, 7.2.0, you may choose to use it by doing

```
export CC=/path/to/gcc-7.2.0/bin/gcc
export CXX=/path/to/gcc-7.2.0/bin/g++
```

13.5.2 Build files left in DeePMD-kit

When you try to build a second time when installing DeePMD-kit, files produced before may contribute to failure. Thus, you may clear them by

```
cd build
rm -r *
```

and redo the `cmake` process.

13.6 The temperature undulates violently during the early stages of MD

This is probably because your structure is too far from the equilibrium configuration.

To make sure the potential model is truly accurate, we recommend checking model deviation.

13.7 MD: cannot run LAMMPS after installing a new version of DeePMD-kit

This typically happens when you install a new version of DeePMD-kit and copy directly the generated `USER-DEEPMD` to a LAMMPS source code folder and re-install LAMMPS.

To solve this problem, it suffices to first remove `USER-DEEPMD` from the LAMMPS source code by

```
make no-user-deepmd
```

and then install the new `USER-DEEPMD`.

If this does not solve your problem, try to decompress the LAMMPS source tarball and install LAMMPS from scratch again, which typically should be very fast.

13.8 Model compatibility

When the version of DeePMD-kit used to train the model is different from the that of DeePMD-kit running MDs, one has the problem of model compatibility.

DeePMD-kit guarantees that the codes with the same major and minor revisions are compatible. That is to say, v0.12.5 is compatible with v0.12.0, but is not compatible with v0.11.0 or v1.0.0.

One can execute `dp convert-from` to convert an old model to a new one.

Model version	v0.12	v1.0	v1.1	v1.2	v1.3	v2.0	v2.1	v2.2
Compatibility	😊	😊	😊	😊	😊	😄	😄	😄

Legend:

- 😄: The model is compatible with the DeePMD-kit package.
- 😊: The model is incompatible with the DeePMD-kit package, but one can execute `dp convert-from` to convert an old model to v2.2.
- 😵: The model is incompatible with the DeePMD-kit package, and there is no way to convert models.

13.9 Why does a model have low precision?

Many phenomena are caused by model accuracy. For example, during simulations, temperatures explode, structures fall apart, and atoms are lost. One can test the model to confirm whether the model has the enough accuracy.

There are many reasons for a low-quality model. Some common reasons are listed below.

13.9.1 Data

Data units and signs

The unit of training data should follow what is listed in data section. Usually, the package to calculate the training data has different units from those of the DeePMD-kit. It is noted that some software label the energy gradient as forces, instead of the negative energy gradient. It is neccessary to check them carefully to avoid inconsistent data.

SCF coverage and data accuracy

The accuracy of models will not exceed the accuracy of training data, so the training data should reach enough accuracy. Here is a checklist for the accuracy of data:

- SCF should converge to a suitable threshold for all points in the training data.
- The convergence of the energy, force and virial with respect to the energy cutoff and k-spacing sample is checked.
- Sometimes, QM software may generate unstable outliers, which should be removed.
- The data should be extracted with enough digits and stored with the proper precision. Large energies may have low precision when they are stored as the single-precision floating-point format (FP32).

Enough data

If the model performs good on the training data, but has bad accuracy on another data, this means some data space is not covered by the training data. It can be validated by evaluting the model deviation with multiple models. If the model deviation of these data is high for some data, try to collect more data using DP-GEN.

Values of data

One should be aware that the errors of some data is also affected by the absolute values of this data. Stable structures tend to be more precise than unstable structures because unstable structures may have larger forces. Also, errors will be introduced in the Projector augmented wave (PAW) DFT calculations when the atoms are very close due to the overlap of pseudo-potentials. It is expected to see that data with large forces has larger errors and it is better to compare different models only with the same data.

13.9.2 Model

Enough `sel`

The *`sel`* of the descriptors must be enough for both training and test data. Otherwise, the model will be unreliable and give wrong results.

Cutoff radius

The model cannot fit the long-term interaction out of the cutoff radius. This is a designed approximation for performance, but one has to choose proper cutoff radius for the system.

Neural network size

The size of neural networks will affect the accuracy, but if one follows the parameters in the examples, this effect is insignificant. See FAQ: How to tune Fitting/embedding-net size for details.

Neural network precision

In some cases, one may want to use the FP32 precision to make the model faster. For some applications, FP32 is enough and thus is recommended, but one should still be aware that the precision of FP32 is not as high as that of FP64.

13.9.3 Training

Training steps

Generally speaking, the longer the number of training steps, the better the model. A balance between model accuracy and training time can be achieved. If one finds that model accuracy decreases with training time, there may be a problem with the data. See the data section for details.

Learning rate

Both too large and too small learning rate may affect the training. It is recommended to start with a large learning rate and end with a small learning rate. The learning rate from the examples is a good choice to start.

CHAPTER

FOURTEEN

FIND DEEPMD-KIT C/C++ LIBRARY FROM CMAKE

After DeePMD-kit C/C++ library is installed, one can find DeePMD-kit from CMake:

```
find_package(DeePMD REQUIRED)
```

Note that you may need to add ${deepmd_root} to the cached CMake variable `CMAKE_PREFIX_PATH`.

To link against the C interface library, using

```
target_link_libraries(some_library PRIVATE DeePMD::deepmd_c)
```

To link against the C++ interface library, using

```
target_link_libraries(some_library PRIVATE DeePMD::deepmd_cc)
```

CHAPTER

FIFTEEN

CREATE A MODEL IN TENSORFLOW

If you'd like to create a new model that isn't covered by the existing DeePMD-kit library, but reuse DeePMD-kit's other efficient modules such as data processing, trainner, etc, you may want to read this section.

To incorporate your custom model you'll need to:

1. Register and implement new components (e.g. descriptor) in a Python file. You may also want to register new TensorFlow OPs if necessary.
2. Register new arguments for user inputs.
3. Package new codes into a Python package.
4. Test new models.

15.1 Design a new component

When creating a new component, take descriptor as the example, one should inherit from the *deepmd.tf.descriptor.descriptor.Descriptor* class and override several methods. Abstract methods such as *deepmd.tf.descriptor.descriptor.Descriptor.build* must be implemented and others are not. You should keep arguments of these methods unchanged.

After implementation, you need to register the component with a key:

```
from deepmd.tf.descriptor import Descriptor


@Descriptor.register("some_descrpt")
class SomeDescript(Descriptor):
    def __init__(self, arg1: bool, arg2: float) -> None:
        pass
```

15.2 Register new arguments

To let someone uses your new component in their input file, you need to create a new method that returns some `Argument` of your new component, and then register new arguments. For example, the code below

```
from typing import List

from dargs import Argument
from deepmd.utils.argcheck import descrpt_args_plugin
```

(continues on next page)

(continued from previous page)

```
@descrpt_args_plugin.register("some_descrpt")
def descrpt_some_args() -> List[Argument]:
    return [
        Argument("arg1", bool, optional=False, doc="balabala"),
        Argument("arg2", float, optional=True, default=6.0, doc="haha"),
    ]
```

allows one to use your new descriptor as below:

```
"descriptor" :{
    "type": "some_descrpt",
    "arg1": true,
    "arg2": 6.0
}
```

The arguments here should be consistent with the class arguments of your new component.

15.3 Package new codes

You may use `setuptools` to package new codes into a new Python package. It's crucial to add your new component to `entry_points['deepmd']` in `setup.py`:

```
entry_points = (
    {
        "deepmd": [
            "some_descrpt=deepmd_some_descrtpt:SomeDescript",
        ],
    },
)
```

where `deepmd_some_descrtpt` is the module of your codes. It is equivalent to `from deepmd_some_descrtpt import SomeDescript`.

If you place `SomeDescript` and `descrpt_some_args` into different modules, you are also expected to add `descrpt_some_args` to `entry_points`.

After you install your new package, you can now use `dp train` to run your new model.

CHAPTER

SIXTEEN

CREATE A MODEL IN PYTORCH

If you'd like to create a new model that isn't covered by the existing DeePMD-kit library, but reuse DeePMD-kit's other efficient modules such as data processing, trainner, etc, you may want to read this section.

To incorporate your custom model you'll need to:

1. Register and implement new components (e.g. descriptor) in a Python file.
2. Register new arguments for user inputs.
3. Package new codes into a Python package.
4. Test new models.

16.1 Design a new component

With DeePMD-kit v3, we have expanded support to include two additional backends alongside TensorFlow: the PyTorch backend and the framework-independent backend (dpmodel). The PyTorch backend adopts a highly modularized design to provide flexibility and extensibility. It ensures a consistent experience for both training and inference, aligning with the TensorFlow backend.

The framework-independent backend is implemented in pure NumPy, serving as a reference backend to ensure consistency in tests. Its design pattern closely parallels that of the PyTorch backend.

16.1.1 New descriptors

When creating a new descriptor, it is essential to inherit from both the *deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor* class and the `torch.nn.Module` class. Abstract methods, including `deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor.forward`, must be implemented, while others remain optional. It is crucial to adhere to the original method arguments without any modifications. Once the implementation is complete, the next step involves registering the component with a designated key:

```
from deepmd.pt.model.descriptor.base_descriptor import (
    BaseDescriptor,
)


@BaseDescriptor.register("some_descrpt")
class SomeDescript(BaseDescriptor, torch.nn.Module):
    def __init__(self, arg1: bool, arg2: float) -> None:
        pass
```

(continues on next page)

(continued from previous page)

```
    def get_rcut(self) -> float:
        pass

    def get_nnei(self) -> int:
        pass

    def get_ntypes(self) -> int:
        pass

    def get_dim_out(self) -> int:
        pass

    def get_dim_emb(self) -> int:
        pass

    def mixed_types(self) -> bool:
        pass

    def forward(
        self,
        coord_ext: torch.Tensor,
        atype_ext: torch.Tensor,
        nlist: torch.Tensor,
        mapping: Optional[torch.Tensor] = None,
    ):
        pass

    def serialize(self) -> dict:
        pass

    def deserialize(cls, data: dict) -> "SomeDescript":
        pass

    def update_sel(cls, global_jdata: dict, local_jdata: dict):
        pass
```

The serialize and deserialize methods are important for cross-backend model conversion.

16.1.2 New fitting nets

In many instances, there is no requirement to create a new fitting net. For fitting user-defined scalar properties, the `deepmd.pt.model.task.ener.InvarFitting` class can be utilized. However, if there is a need for a new fitting net, one should inherit from both the *`deepmd.pt.model.task.base_fitting.BaseFitting`* class and the `torch.nn.Module` class. Alternatively, for a more straightforward approach, inheritance from the *`deepmd.pt.model.task.fitting.GeneralFitting`* class is also an option.

```
from deepmd.pt.model.task.fitting import (
    GeneralFitting,
)
from deepmd.dpmodel import (
    FittingOutputDef,
    fitting_check_output,
)
```

(continues on next page)

(continued from previous page)

```
@GeneralFitting.register("some_fitting")
@fitting_check_output
class SomeFittingNet(GeneralFitting):
    def __init__(self, arg1: bool, arg2: float) -> None:
        pass

    def forward(
        self,
        descriptor: torch.Tensor,
        atype: torch.Tensor,
        gr: Optional[torch.Tensor] = None,
        g2: Optional[torch.Tensor] = None,
        h2: Optional[torch.Tensor] = None,
        fparam: Optional[torch.Tensor] = None,
        aparam: Optional[torch.Tensor] = None,
    ):
        pass

    def output_def(self) -> FittingOutputDef:
        pass
```

16.1.3 New models

The PyTorch backend's model architecture is meticulously structured with multiple layers of abstraction, ensuring a high degree of flexibility. Typically, the process commences with an atomic model responsible for atom-wise property calculations. This atomic model inherits from both the *deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel* class and the `torch.nn.Module` class.

Subsequently, the `AtomicModel` is encapsulated using the `make_model(AtomicModel)` function, which leverages the `deepmd.pt.model.model.make_model.make_model` function. The purpose of the `make_model` wrapper is to facilitate the translation between atomic property predictions and the extended property predictions and differentiation , e.g. the reduction of atomic energy contribution and the autodiff for calculating the forces and virial. The developers usually need to implement an `AtomicModel` not a `Model`.

```
from deepmd.pt.model.atomic_model.base_atomic_model import (
    BaseAtomicModel,
)


class SomeAtomicModel(BaseAtomicModel, torch.nn.Module):
    def __init__(self, arg1: bool, arg2: float) -> None:
        pass

    def forward_atomic(self):
        pass
```

16.2 Register new arguments

To let someone uses your new component in their input file, you need to create a new method that returns some `Argument` of your new component, and then register new arguments. For example, the code below

```
from typing import List

from dargs import Argument
from deepmd.utils.argcheck import descrpt_args_plugin


@descrpt_args_plugin.register("some_descrpt")
def descrpt_some_args() -> List[Argument]:
    return [
        Argument("arg1", bool, optional=False, doc="balabala"),
        Argument("arg2", float, optional=True, default=6.0, doc="haha"),
    ]
```

allows one to use your new descriptor as below:

```
"descriptor" :{
    "type": "some_descrpt",
    "arg1": true,
    "arg2": 6.0
}
```

The arguments here should be consistent with the class arguments of your new component.

16.3 Unit tests

When transferring features from another backend to the PyTorch backend, it is essential to include a regression test in `/source/tests/consistent` to validate the consistency of the PyTorch backend with other backends. Presently, the regression tests cover self-consistency and cross-backend consistency between TensorFlow, PyTorch, and DP (Numpy) through the serialization/deserialization technique.

During the development of new components within the PyTorch backend, it is necessary to provide a DP (Numpy) implementation and incorporate corresponding regression tests. For PyTorch components, developers are also required to include a unit test using `torch.jit`.

CHAPTER

SEVENTEEN

ATOM TYPE EMBEDDING

17.1 Overview

Here is an overview of the DeePMD-kit algorithm. Given a specific centric atom, we can obtain the matrix describing its local environment, named $\mathcal{R}$. It consists of the distance between the centric atom and its neighbors, as well as a direction vector. We can embed each distance into a vector of M_1 dimension by an `embedding net`, so the environment matrix $\mathcal{R}$ can be embedded into matrix $\mathcal{G}$. We can thus extract a descriptor vector (of $M_1 \times M_2$ dim) of the centric atom from the $\mathcal{G}$ by some matrix multiplication, and put the descriptor into `fitting net` to get the predicted energy E. The vanilla version of DeePMD-kit builds `embedding net` and `fitting net` relying on the atom type, resulting in $O(N)$ memory usage. After applying atom type embedding, in DeePMD-kit v2.0, we can share one `embedding net` and one `fitting net` in total, which reduces training complexity largely.

17.2 Preliminary

In the following chart, you can find the meaning of symbols used to clarify the atom-type embedding algorithm.

i: Type of centric atom

j: Type of neighbor atom

s_{ij}: Distance between centric atom and neighbor atom

$\mathcal{G}_{ij}(\cdot)$: Origin embedding net, take s_{ij} as input and output embedding vector of M_1 dim

$\mathcal{G}(\cdot)$: Shared embedding net

Multi$(\cdot)$: Matrix multiplication and flattening, output the descriptor vector of $M_1 \times M_2$ dim

$F_i(\cdot)$: Origin fitting net, take the descriptor vector as input and output energy

$F(\cdot)$: Shared fitting net

$A(\cdot)$: Atom type embedding net, input is atom type, the output is type embedding vector of dim `nchanl`

So, we can formulate the training process as follows. Vanilla DeePMD-kit algorithm:

$$E = F_i(\text{Multi}(\mathcal{G}_{ij}(s_{ij})))$$

DeePMD-kit applying atom type embedding:

$$E = F([\text{Multi}(\mathcal{G}([s_{ij}, A(i), A(j)])), A(j)])$$

or

$$E = F([\text{Multi}(\mathcal{G}([s_{ij}, A(j)])), A(j)])$$

The difference between the two variants above is whether using the information of centric atom when generating the descriptor. Users can choose by modifying the `type_one_side` hyper-parameter in the input JSON file.

17.3 How to use

A detailed introduction can be found at *se_e2_a_tebd*. Looking for a fast start-up, you can simply add a `type_embedding` section in the input JSON file as displayed in the following, and the algorithm will adopt the atom type embedding algorithm automatically. An example of `type_embedding` is like

```
"type_embedding":{
   "neuron":    [2, 4, 8],
   "resnet_dt": false,
   "seed":      1
}
```

17.4 Code Modification

Atom-type embedding can be applied to varied `embedding net` and `fitting net`, as a result, we build a class `TypeEmbedNet` to support this free combination. In the following, we will go through the execution process of the code to explain our code modification.

17.4.1 trainer (train/trainer.py)

In trainer.py, it will parse the parameter from the input JSON file. If a `type_embedding` section is detected, it will build a `TypeEmbedNet`, which will be later input in the `model`. `model` will be built in the function `_build_network`.

17.4.2 model (model/ener.py)

When building the operation graph of the `model` in `model.build`. If a `TypeEmbedNet` is detected, it will build the operation graph of `type embed net`, `embedding net` and `fitting net` by order. The building process of `type embed net` can be found in `TypeEmbedNet.build`, which output the type embedding vector of each atom type (of [ntypes $\times$ nchanl] dimensions). We then save the type embedding vector into `input_dict`, so that they can be fetched later in `embedding net` and `fitting net`.

17.4.3 embedding net (descriptor/se*.py)

In `embedding net`, we shall take local environment $\mathcal{R}$ as input and output matrix $\mathcal{G}$. Functions called in this process by the order is

```
build -> _pass_filter -> _filter -> _filter_lower
```

`_pass_filter`: It will first detect whether an atom type embedding exists, if so, it will apply atom type embedding algorithm and doesn't divide the input by type.

`_filter`: It will call `_filter_lower` function to obtain the result of matrix multiplication ($\mathcal{G}^T \cdot \mathcal{R}$), do further multiplication involved in Multi($\cdot$), and finally output the result of descriptor vector of $M_1 \times M_2$ dim.

`_filter_lower`: The main function handling input modification. If type embedding exists, it will call `_concat_type_embedding` function to concat the first column of input $\mathcal{R}$ (the column of s_{ij}) with the atom type embedding information. It will decide whether to use the atom type embedding vector of the centric atom according to the value of `type_one_side` (if set True, then we only use the vector of the neighbor atom). The modified input will be put into the `fitting net` to get $\mathcal{G}$ for further matrix multiplication stage.

17.4.4 fitting net (fit/ener.py)

In `fitting net`, it takes the descriptor vector as input, whose dimension is [natoms, $M_1 \times M_2$]. Because we need to involve information on the centric atom in this step, we need to generate a matrix named `atype_embed` (of dim [natoms, nchanl]), in which each row is the type embedding vector of the specific centric atom. The input is sorted by type of centric atom, we also know the number of a particular atom type (stored in `natoms[2+i]`), thus we get the type vector of the centric atom. In the build phase of the fitting net, it will check whether type embedding exists in `input_dict` and fetch them. After that, call `embed_atom_type` function to look up the embedding vector for the type vector of the centric atom to obtain `atype_embed`, and concat input with it ([input, atype_embed]). The modified input goes through `fitting` net` to get predicted energy.

Note: You can't apply the compression method while using atom-type embedding.

CHAPTER

EIGHTEEN

CODING CONVENTIONS

18.1 Preface

The aim of these coding standards is to help create a codebase with a defined and consistent coding style that every contributor can get easily familiar with. This will in enhance code readability as there will be no different coding styles from different contributors and everything will be documented. Also, PR diffs will be smaller because of the unified coding style. Finally, static typing will help in hunting down potential bugs before the code is even run.

Contributed code will not be refused merely because it does not strictly adhere to these conditions; as long as it's internally consistent, clean, and correct, it probably will be accepted. But don't be surprised if the "offending" code gets fiddled with overtime to conform to these conventions.

There are also pre-commit CI checks for python code style which will automatically fix the PR.

18.2 Python

18.2.1 Rules

The code must be compatible with the oldest supported version of python which is 3.8.

The project follows the generic coding conventions as specified in the Style Guide for Python Code, Docstring Conventions and Typing Conventions PEPs, clarified and extended as follows:

- Do not use "`*`" imports such as `from module import *`. Instead, list imports explicitly.
- Use 4 spaces per indentation level. No tabs.
- No one-liner compound statements (i.e., no `if x: return`: use two lines).
- Maximum line length is 88 characters as recommended by black which is less strict than Docstring Conventions suggests.
- Use "StudlyCaps" for class names.
- Use "lowercase" or "lowercase_with_underscores" for function, method, variable names and module names. For short names, joined lowercase may be used (e.g. "tagname"). Choose what is most readable.
- No single-character variable names, except indices in loops that encompass a very small number of lines (`for i in range(5): ...`).
- Avoid lambda expressions. Use named functions instead.
- Avoid functional constructs (filter, map, etc.). Use list comprehensions instead.

- Use `"double quotes"` for string literals, and `"""triple double quotes"""` for docstring's. Single quotes are OK for something like

```
f"something {'this' if x else 'that'}"
```

- Use f-strings `s = f"{x:.2f}"` instead of old style formating with `"%f" % x`. string format method `"{x:.2f}".format()` may be used sparsely where it is more convenient than f-strings.

18.2.2 Whitespace

Python is not C/C++ so whitespace should be used sparingly to maintain code readability

- Read the Whitespace in Expressions and Statements section of PEP8.
- Avoid trailing whitespaces.
- Do not use excessive whitespace in your expressions and statements.
- You should have blank spaces after commas, colons, and semi-colons if it isn't trailing next to the end of a bracket, brace, or parentheses.
- With any operators you should use space on both sides of the operator.
- Colons for slicing are considered a binary operator, and should not have any spaces between them.
- You should have parentheses with no space, directly next to the function when calling functions `function()`.
- When indexing or slicing the brackets should be directly next to the collection with no space `collection["index"]`.
- Whitespace used to line up variable values is not recommended.
- Make sure you are consistent with the formats you choose when optional choices are available.

18.2.3 General advice

- Get rid of as many `break` and `continue` statements as possible.
- Write short functions. All functions should fit within a standard screen.
- Use descriptive variable names.

18.2.4 Writing documentation in the code

Here is an example of how to write good docstrings:

https://github.com/numpy/numpy/blob/master/doc/example.py

The NumPy docstring documentation can be found here

18.3 C++

The customized Clang Format style is used for C++ code formatting. The style is defined in `.clang-format` file in the root of the repository. The style is based on the Google C++ style with some modifications.

18.4 Run scripts to check the code

It's a good idea to install pre-commit on your repository:

```
$ pip install pre-commit
$ pre-commit install
```

The scripts will be run automatically before each commit and will fix the code style issues automatically.

CHAPTER

NINETEEN

CI/CD

19.1 CI

19.1.1 Test CUDA

`Test CUDA` action runs tests on a self-hosted runner with the NVIDIA card. It is not triggered by every PR. The developer who has the permission to manage the label can apply the label `Test CUDA` to a PR to trigger this action.

19.2 CD

GitHub Actions is used to build pre-compiled packages for each commit. See the Easy install the latest development version section to learn how to install the latest development version.

CHAPTER

TWENTY

DEEPMD

DeePMD-kit is a package written in Python/C++, designed to minimize the effort required to build deep learning-based model of interatomic potential energy and force field and to perform molecular dynamics (MD).

The top module (deepmd.__init__) should not import any third-party modules for performance.

20.1 Subpackages

20.1.1 deepmd.backend

Backends.

Avoid directly importing third-party libraries in this module for performance.

Submodules

deepmd.backend.backend

Module Contents

Classes

Backend	General backend class.

class deepmd.backend.backend.Backend

Bases: *deepmd.utils.plugin.PluginVariant*, make_plugin_registry('backend')

General backend class.

Examples

```
>>> @Backend.register("tf")
>>> @Backend.register("tensorflow")
>>> class TensorFlowBackend(Backend):
...     pass
```

class Feature(*args, **kwds)

Bases: enum.Flag

Feature flag to indicate whether the backend supports certain features.

ENTRY_POINT

Support entry point hook.

DEEP_EVAL

Support Deep Eval backend.

NEIGHBOR_STAT

Support neighbor statistics.

IO

Support IO hook.

abstract property entry_point_hook: Callable[[argparse.Namespace], None]

The entry point hook of the backend.

Returns

Callable[[Namespace], None]
The entry point hook of the backend.

abstract property deep_eval: Type[*deepmd.infer.deep_eval.DeepEvalBackend*]

The Deep Eval backend of the backend.

Returns

type[DeepEvalBackend]
The Deep Eval backend of the backend.

abstract property neighbor_stat: Type[*deepmd.utils.neighbor_stat.NeighborStat*]

The neighbor statistics of the backend.

Returns

type[NeighborStat]
The neighbor statistics of the backend.

abstract property serialize_hook: Callable[[str], dict]

The serialize hook to convert the model file to a dictionary.

Returns

Callable[[str], dict]
The serialize hook of the backend.

abstract property deserialize_hook: Callable[[str, dict], None]

The deserialize hook to convert the dictionary to a model file.

Returns

Callable[[str, dict], None]
The deserialize hook of the backend.

name: ClassVar[str] = 'Unknown'
The formal name of the backend.

To be consistent, this name should be also registered in the plugin system.

features: ClassVar[*Backend.Feature*]
The features of the backend.

suffixes: ClassVar[List[str]] = []
The supported suffixes of the saved model.

The first element is considered as the default suffix.

static get_backend(key: str) → Type[Backend]
Get the backend by key.

Parameters

key
[str] the key of a backend

Returns

Backend
the backend

static get_backends() → Dict[str, Type[Backend]]
Get all the registered backend names.

Returns

list
all the registered backends

static get_backends_by_feature(feature: Backend) → Dict[str, Type[Backend]]
Get all the registered backend names with a specific feature.

Parameters

feature
[*Backend.Feature*] the feature flag

Returns

list
all the registered backends with the feature

static detect_backend_by_model(filename: str) → Type[Backend]
Detect the backend of the given model file.

Parameters

filename
[str] The model file name

abstract is_available() → bool
Check if the backend is available.

Returns

bool
Whether the backend is available.

deepmd.backend.dpmodel

Module Contents

Classes

DPModelBackend	DPModel backend that uses NumPy as the reference implementation.

class deepmd.backend.dpmodel.DPModelBackend

Bases: *deepmd.backend.backend.Backend*

DPModel backend that uses NumPy as the reference implementation.

abstract property entry_point_hook: Callable[[argparse.Namespace], None]

The entry point hook of the backend.

Returns

Callable[[Namespace], None]
The entry point hook of the backend.

property deep_eval: Type[*deepmd.infer.deep_eval.DeepEvalBackend*]

The Deep Eval backend of the backend.

Returns

type[DeepEvalBackend]
The Deep Eval backend of the backend.

property neighbor_stat: Type[*deepmd.dpmodel.utils.neighbor_stat.NeighborStat*]

The neighbor statistics of the backend.

Returns

type[NeighborStat]
The neighbor statistics of the backend.

property serialize_hook: Callable[[str], dict]

The serialize hook to convert the model file to a dictionary.

Returns

Callable[[str], dict]
The serialize hook of the backend.

property deserialize_hook: Callable[[str, dict], None]

The deserialize hook to convert the dictionary to a model file.

Returns

Callable[[str, dict], None]
The deserialize hook of the backend.

name = 'DPModel'

The formal name of the backend.

features: ClassVar[*deepmd.backend.backend.Backend.Feature*]
The features of the backend.

suffixes: ClassVar[List[str]] = ['.dp']
The suffixes of the backend.

is_available() → bool
Check if the backend is available.

Returns

bool
Whether the backend is available.

deepmd.backend.pytorch

Module Contents

Classes

PyTorchBackend	PyTorch backend.

class deepmd.backend.pytorch.PyTorchBackend
Bases: *deepmd.backend.backend.Backend*

PyTorch backend.

property entry_point_hook: Callable[[argparse.Namespace], None]
The entry point hook of the backend.

Returns

Callable[[Namespace], None]
The entry point hook of the backend.

property deep_eval: Type[*deepmd.infer.deep_eval.DeepEvalBackend*]
The Deep Eval backend of the backend.

Returns

type[DeepEvalBackend]
The Deep Eval backend of the backend.

property neighbor_stat: Type[*deepmd.pt.utils.neighbor_stat.NeighborStat*]
The neighbor statistics of the backend.

Returns

type[NeighborStat]
The neighbor statistics of the backend.

property serialize_hook: Callable[[str], dict]
The serialize hook to convert the model file to a dictionary.

Returns

Callable[[str], dict]
The serialize hook of the backend.

property deserialize_hook: Callable[[str, dict], None]

The deserialize hook to convert the dictionary to a model file.

Returns

Callable[[str, dict], None]
The deserialize hook of the backend.

name = 'PyTorch'

The formal name of the backend.

features: ClassVar[*deepmd.backend.backend.Backend.Feature*]

The features of the backend.

suffixes: ClassVar[List[str]] = ['.pth', '.pt']

The suffixes of the backend.

is_available() → bool

Check if the backend is available.

Returns

bool
Whether the backend is available.

deepmd.backend.suffix

Module Contents

Functions

format_model_suffix(→ str)	Check and format the suffixes of a filename.

deepmd.backend.suffix.**format_model_suffix**(filename: str, feature: deepmd.backend.backend.Backend.Feature | None = None, preferred_backend: str | Type[deepmd.backend.backend.Backend] | None = None, strict_prefer: bool | None = None) → str

Check and format the suffixes of a filename.

When preferred_backend is not given, this method checks the suffix of the filename is within the suffixes of the any backends (with the given feature) and doesn't do formating. When preferred_backend is given, strict_prefer must be given. If strict_prefer is True and the suffix is not within the suffixes of the preferred backend, or strict_prefer is False and the suffix is not within the suffixes of the any backend with the given feature, the filename will be formatted with the preferred suffix of the preferred backend.

Parameters

filename
[str] The filename to be formatted.

feature
[Backend.Feature, optional] The feature of the backend, by default None

preferred_backend
[str or type of Backend, optional] The preferred backend, by default None

strict_prefer
: [bool, `optional`] Whether to strictly prefer the preferred backend, by default None

Returns

`str`
: The formatted filename with the correct suffix.

Raises

`ValueError`
: When preferred_backend is not given and the filename is not supported by any backend.

`deepmd.backend.tensorflow`

Module Contents

Classes

TensorFlowBackend	TensorFlow backend.

class deepmd.backend.tensorflow.**TensorFlowBackend**

Bases: *deepmd.backend.backend.Backend*

TensorFlow backend.

property entry_point_hook: Callable[[argparse.Namespace], None]

The entry point hook of the backend.

Returns

`Callable`[[`Namespace`], `None`]
: The entry point hook of the backend.

property deep_eval: Type[*deepmd.infer.deep_eval.DeepEvalBackend*]

The Deep Eval backend of the backend.

Returns

`type`[`DeepEvalBackend`]
: The Deep Eval backend of the backend.

property neighbor_stat: Type[*deepmd.tf.utils.neighbor_stat.NeighborStat*]

The neighbor statistics of the backend.

Returns

`type`[`NeighborStat`]
: The neighbor statistics of the backend.

property serialize_hook: Callable[[str], dict]

The serialize hook to convert the model file to a dictionary.

Returns

`Callable`[[`str`], `dict`]
: The serialize hook of the backend.

property deserialize_hook: Callable[[str, dict], None]

The deserialize hook to convert the dictionary to a model file.

Returns

Callable[[str, dict], None]
The deserialize hook of the backend.

name = 'TensorFlow'

The formal name of the backend.

features: ClassVar[*deepmd.backend.backend.Backend.Feature*]

The features of the backend.

suffixes: ClassVar[List[str]] = ['.pb']

The suffixes of the backend.

is_available() → bool

Check if the backend is available.

Returns

bool
Whether the backend is available.

Package Contents

deepmd.backend.PACKAGE_BASE = 'deepmd.backend'

deepmd.backend.NOT_LOADABLE = ('__init__.py',)

deepmd.backend.module_name

deepmd.backend.eps

deepmd.backend.plugin

20.1.2 deepmd.dpmodel

Subpackages

deepmd.dpmodel.atomic_model

The atomic model provides the prediction of some property on each atom. All the atomic models are not supposed to be directly accessed by users, but it provides a convenient interface for the implementation of models.

Taking the energy models for example, the developeres only needs to implement the atomic energy prediction via an atomic model, and the model can be automatically made by the deepmd.dpmodel.make_model method. The DPModel is made by ` DPModel = make_model(DPAtomicModel) `

Submodules

deepmd.dpmodel.atomic_model.base_atomic_model

Module Contents

Classes

BaseAtomicModel	Base Atomic Model provides the interfaces of an atomic model.

Attributes

*BaseAtomicModel_*

deepmd.dpmodel.atomic_model.base_atomic_model.**BaseAtomicModel_**

class deepmd.dpmodel.atomic_model.base_atomic_model.**BaseAtomicModel**(type_map: List[str], atom_exclude_types: List[int] = [], pair_exclude_types: List[Tuple[int, int]] = [], rcond: float | None = None, preset_out_bias: Dict[str, numpy.ndarray] | None = None)

Bases: *BaseAtomicModel_*

Base Atomic Model provides the interfaces of an atomic model.

init_out_stat()

Initialize the output bias.

__setitem__(key, value)

__getitem__(key)

get_type_map() → List[str]

Get the type map.

reinit_atom_exclude(exclude_types: List[int] = [])

reinit_pair_exclude(exclude_types: List[Tuple[int, int]] = [])

atomic_output_def() → deepmd.dpmodel.output_def.FittingOutputDef

Get the output def of the atomic model.

By default it is the same as FittingOutputDef, but it allows model level wrapper of the output defined by the developer.

forward_common_atomic(extended_coord: numpy.ndarray, extended_atype: numpy.ndarray, nlist: numpy.ndarray, mapping: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]

Common interface for atomic inference.

This method accept extended coordinates, extended atom typs, neighbor list, and predict the atomic contribution of the fit property.

Parameters

extended_coord
: extended coodinates, shape: nf x (nall x 3)

extended_atype
: extended atom typs, shape: nf x nall for a type < 0 indicating the atomic is virtual.

nlist
: neighbor list, shape: nf x nloc x nsel

mapping
: extended to local index mapping, shape: nf x nall

fparam
: frame parameters, shape: nf x dim_fparam

aparam
: atomic parameter, shape: nf x nloc x dim_aparam

Returns

ret_dict
: dict of output atomic properties. should implement the definition of fitting_output_def. ret_dict["mask"] of shape nf x nloc will be provided. ret_dict["mask"][ff,ii] == 1 indicating the ii-th atom of the ff-th frame is real. ret_dict["mask"][ff,ii] == 0 indicating the ii-th atom of the ff-th frame is virtual.

serialize() → dict

classmethod deserialize(data: dict) → BaseAtomicModel

apply_out_stat(ret: Dict[str, numpy.ndarray], atype: numpy.ndarray)

Apply the stat to each atomic output. The developer may override the method to define how the bias is applied to the atomic output of the model.

Parameters

ret
: The returned dict by the forward_atomic method

atype
: The atom types. nf x nloc

_varsize(shape: List[int]) → int

_get_bias_index(kk: str) → int

_fetch_out_stat(keys: List[str]) → Tuple[Dict[str, numpy.ndarray], Dict[str, numpy.ndarray]]

deepmd.dpmodel.atomic_model.dp_atomic_model

Module Contents

Classes

DPAtomicModel	Model give atomic prediction of some physical property.

class deepmd.dpmodel.atomic_model.dp_atomic_model.**DPAtomicModel**(descriptor, fitting, type_map: List[str], **kwargs)

Bases: *deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicModel*

Model give atomic prediction of some physical property.

Parameters

descriptor
Descriptor

fitting_net
Fitting net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

fitting_output_def() → deepmd.dpmodel.output_def.FittingOutputDef

Get the output def of the fitting net.

get_rcut() → float

Get the cut-off radius.

get_sel() → List[int]

Get the neighbor selection.

mixed_types() → bool

If true, the model 1. assumes total number of atoms aligned across frames; 2. uses a neighbor list that does not distinguish different atomic types.

If false, the model 1. assumes total number of atoms of each atom type aligned across frames; 2. uses a neighbor list that distinguishes different atomic types.

forward_atomic(extended_coord: numpy.ndarray, extended_atype: numpy.ndarray, nlist: numpy.ndarray, mapping: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]

Models' atomic predictions.

Parameters

extended_coord
coodinates in extended region

extended_atype
atomic type in extended region

nlist
neighbor list. nf x nloc x nsel

mapping
mapps the extended indices to local indices. nf x nall

fparam
frame parameter. nf x ndf

aparam
atomic parameter. nf x nloc x nda

Returns

result_dict
the result dict, defined by the FittingOutputDef.

serialize() → dict

classmethod deserialize(data) → DPAtomicModel

get_dim_fparam() → int
Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam() → int
Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]
Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool
Check whether the shape of atomic parameters is (nframes, nall, ndim).

If False, the shape is (nframes, nloc, ndim).

deepmd.dpmodel.atomic_model.linear_atomic_model

Module Contents

Classes

LinearEnergyAtomicModel	Linear model make linear combinations of several existing models.
DPZBLLinearEnergyAtomicModel	Model linearly combine a list of AtomicModels.

class deepmd.dpmodel.atomic_model.linear_atomic_model.**LinearEnergyAtomicModel**(models: List[deepmd.dpmodel.atomic_
type_map: List[str],
**kwargs)

Bases: *deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicModel*

Linear model make linear combinations of several existing models.

Parameters

models
[list[DPAtomicModel or PairTabAtomicModel]] A list of models to be combined. PairTabAtomicModel must be used together with a DPAtomicModel.

type_map
[list[str]] Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

mixed_types() → bool

If true, the model 1. assumes total number of atoms aligned across frames; 2. uses a neighbor list that does not distinguish different atomic types.

If false, the model 1. assumes total number of atoms of each atom type aligned across frames; 2. uses a neighbor list that distinguishes different atomic types.

get_rcut() → float

Get the cut-off radius.

get_type_map() → List[str]

Get the type map.

get_model_rcuts() → List[float]

Get the cut-off radius for each individual models.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_model_nsels() → List[int]

Get the processed sels for each individual models. Not distinguishing types.

get_model_sels() → List[int | List[int]]

Get the sels for each individual models.

_sort_rcuts_sels() → Tuple[List[float], List[int]]

forward_atomic(extended_coord, extended_atype, nlist, mapping: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]

Return atomic prediction.

Parameters

extended_coord
coodinates in extended region, (nframes, nall * 3)

extended_atype
atomic type in extended region, (nframes, nall)

nlist
neighbor list, (nframes, nloc, nsel).

mapping
mapps the extended indices to local indices.

fparam
frame parameter. (nframes, ndf)

aparam
atomic parameter. (nframes, nloc, nda)

Returns

result_dict
the result dict, defined by the fitting net output def.

static remap_atype(ori_map: List[str], new_map: List[str]) → numpy.ndarray

This method is used to map the atype from the common type_map to the original type_map of indivial AtomicModels.

Parameters

ori_map
[List[str]] The original type map of an AtomicModel.

new_map
[List[str]] The common type map of the DPZBLLinearEnergyAtomicModel, created by the get_type_map method, must be a subset of the ori_map.

Returns

np.ndarray

fitting_output_def() → deepmd.dpmodel.output_def.FittingOutputDef

Get the output def of developer implemented atomic models.

serialize() → dict

classmethod deserialize(data: dict) → LinearEnergyAtomicModel

_compute_weight(extended_coord: numpy.ndarray, extended_atype: numpy.ndarray, nlists_: List[numpy.ndarray]) → List[numpy.ndarray]

This should be a list of user defined weights that matches the number of models to be combined.

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim).

If False, the shape is (nframes, nloc, ndim).

class deepmd.dpmodel.atomic_model.linear_atomic_model.DPZBLLinearEnergyAtomicModel(dp_model: deepmd.dpmodel.atomic
zbl_model: deepmd.dpmodel.atomic
sw_rmin: float,
sw_rmax: float,
type_map: List[str],
smin_alpha: float | None = 0.1,
**kwargs)

Bases: *LinearEnergyAtomicModel*

Model linearly combine a list of AtomicModels.

Parameters

dp_model
The DPAtomicModel being combined.

zbl_model
The PairTable model being combined.

sw_rmin
The lower boundary of the interpolation between short-range tabulated interaction and DP.

sw_rmax
The upper boundary of the interpolation between short-range tabulated interaction and DP.

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

smin_alpha
The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin.

serialize() → dict

classmethod deserialize(data) → DPZBLLinearEnergyAtomicModel

_compute_weight(extended_coord: numpy.ndarray, extended_atype: numpy.ndarray, nlists_: List[numpy.ndarray]) → List[numpy.ndarray]

ZBL weight.

Returns

List[**np.ndarray**]
the atomic ZBL weight for interpolation. (nframes, nloc, 1)

deepmd.dpmodel.atomic_model.make_base_atomic_model

Module Contents

Functions

make_base_atomic_model(t_tensor[, fwd_method_name])	Make the base class for the atomic model.

deepmd.dpmodel.atomic_model.make_base_atomic_model.**make_base_atomic_model**(t_tensor, fwd_method_name: str = 'forward_atomic')

Make the base class for the atomic model.

Parameters

t_tensor
: The type of the tensor. used in the type hint.

fwd_method_name
: Name of the forward method. For dpmodels, it should be "call". For torch models, it should be "forward".

deepmd.dpmodel.atomic_model.pairtab_atomic_model

Module Contents

Classes

PairTabAtomicModel	Pairwise tabulation energy model.

class deepmd.dpmodel.atomic_model.pairtab_atomic_model.**PairTabAtomicModel**(tab_file: str, rcut: float, sel: int | List[int], type_map: List[str], rcond: float | None = None, atom_ener: List[float] | None = None, **kwargs)

Bases: *deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicModel*

Pairwise tabulation energy model.

This model can be used to tabulate the pairwise energy between atoms for either short-range or long-range interactions, such as D3, LJ, ZBL, etc. It should not be used alone, but rather as one submodel of a linear (sum) model, such as DP+D3.

Do not put the model on the first model of a linear model, since the linear model fetches the type map from the first model.

At this moment, the model does not smooth the energy at the cutoff radius, so one needs to make sure the energy has been smoothed to zero.

Parameters

tab_file
[str] The path to the tabulation file.

rcut
[float] The cutoff radius.

sel
[int or list[int]] The maxmum number of atoms in the cut-off radius.

type_map
[list[str]] Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

fitting_output_def() → deepmd.dpmodel.output_def.FittingOutputDef

Get the output def of developer implemented atomic models.

get_rcut() → float

Get the cut-off radius.

get_type_map() → List[str]

Get the type map.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_nsel() → int

Returns the total number of selected neighboring atoms in the cut-off radius.

mixed_types() → bool

If true, the model 1. assumes total number of atoms aligned across frames; 2. uses a neighbor list that does not distinguish different atomic types.

If false, the model 1. assumes total number of atoms of each atom type aligned across frames; 2. uses a neighbor list that distinguishes different atomic types.

serialize() → dict

classmethod deserialize(data) → PairTabAtomicModel

forward_atomic(extended_coord, extended_atype, nlist, mapping: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]

_pair_tabulated_inter(nlist: numpy.ndarray, i_type: numpy.ndarray, j_type: numpy.ndarray, rr: numpy.ndarray) → numpy.ndarray

Pairwise tabulated energy.

Parameters

nlist
[np.ndarray] The unmasked neighbour list. (nframes, nloc)

i_type
[np.ndarray] The integer representation of atom type for all local atoms for all frames. (nframes, nloc)

j_type
[np.ndarray] The integer representation of atom type for all neighbour atoms of all local atoms for all frames. (nframes, nloc, nnei)

rr
[np.ndarray] The salar distance vector between two atoms. (nframes, nloc, nnei)

Returns

np.ndarray
The masked atomic energy for all local atoms for all frames. (nframes, nloc, nnei)

Raises

Exception
If the distance is beyond the table.

Notes

This function is used to calculate the pairwise energy between two atoms. It uses a table containing cubic spline coefficients calculated in PairTab.

static _get_pairwise_dist(coords: numpy.ndarray, nlist: numpy.ndarray) → numpy.ndarray

Get pairwise distance dr.

Parameters

coords
[np.ndarray] The coordinate of the atoms, shape of (nframes, nall, 3).

nlist
The masked nlist, shape of (nframes, nloc, nnei).

Returns

np.ndarray
The pairwise distance between the atoms (nframes, nloc, nnei).

static _extract_spline_coefficient(i_type: numpy.ndarray, j_type: numpy.ndarray, idx: numpy.ndarray, tab_data: numpy.ndarray, nspline: int) → numpy.ndarray

Extract the spline coefficient from the table.

Parameters

i_type
[np.ndarray] The integer representation of atom type for all local atoms for all frames. (nframes, nloc)

j_type
[np.ndarray] The integer representation of atom type for all neighbour atoms of all local atoms for all frames. (nframes, nloc, nnei)

idx
[np.ndarray] The index of the spline coefficient. (nframes, nloc, nnei)

tab_data
[np.ndarray] The table storing all the spline coefficient. (ntype, ntype, nspline, 4)

nspline
[int] The number of splines in the table.

Returns

np.ndarray
The spline coefficient. (nframes, nloc, nnei, 4), shape may be squeezed.

static _calculate_ener(coef: numpy.ndarray, uu: numpy.ndarray) → numpy.ndarray

Calculate energy using spline coeeficients.

Parameters

coef
[np.ndarray] The spline coefficients. (nframes, nloc, nnei, 4)

uu
[np.ndarray] The atom displancemnt used in interpolation and extrapolation (nframes, nloc, nnei)

Returns

np.ndarray
The atomic energy for all local atoms for all frames. (nframes, nloc, nnei)

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim).

If False, the shape is (nframes, nloc, ndim).

Package Contents

Classes

BaseAtomicModel	Base Atomic Model provides the interfaces of an atomic model.
DPAtomicModel	Model give atomic prediction of some physical property.
DPZBLLinearEnergyAtomicModel	Model linearly combine a list of AtomicModels.
LinearEnergyAtomicModel	Linear model make linear combinations of several existing models.
PairTabAtomicModel	Pairwise tabulation energy model.

Functions

make_base_atomic_model(t_tensor[, fwd_method_name])	Make the base class for the atomic model.

class deepmd.dpmodel.atomic_model.BaseAtomicModel(type_map: List[str], atom_exclude_types: List[int] = [], pair_exclude_types: List[Tuple[int, int]] = [], rcond: float | None = None, preset_out_bias: Dict[str, numpy.ndarray] | None = None)

Bases: BaseAtomicModel_

Base Atomic Model provides the interfaces of an atomic model.

init_out_stat()

Initialize the output bias.

__setitem__(key, value)

__getitem__(key)

get_type_map() → List[str]

Get the type map.

reinit_atom_exclude(exclude_types: List[int] = [])

reinit_pair_exclude(exclude_types: List[Tuple[int, int]] = [])

atomic_output_def() → deepmd.dpmodel.output_def.FittingOutputDef

Get the output def of the atomic model.

By default it is the same as FittingOutputDef, but it allows model level wrapper of the output defined by the developer.

forward_common_atomic(extended_coord: numpy.ndarray, extended_atype: numpy.ndarray, nlist: numpy.ndarray, mapping: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]

Common interface for atomic inference.

This method accept extended coordinates, extended atom typs, neighbor list, and predict the atomic contribution of the fit property.

Parameters

extended_coord
: extended coodinates, shape: nf x (nall x 3)

extended_atype
: extended atom typs, shape: nf x nall for a type < 0 indicating the atomic is virtual.

nlist
: neighbor list, shape: nf x nloc x nsel

mapping
: extended to local index mapping, shape: nf x nall

fparam
: frame parameters, shape: nf x dim_fparam

aparam
atomic parameter, shape: nf x nloc x dim_aparam

Returns

ret_dict
dict of output atomic properties. should implement the definition of fitting_output_def. ret_dict["mask"] of shape nf x nloc will be provided. ret_dict["mask"][ff,ii] == 1 indicating the ii-th atom of the ff-th frame is real. ret_dict["mask"][ff,ii] == 0 indicating the ii-th atom of the ff-th frame is virtual.

serialize() → dict

classmethod deserialize(data: dict) → BaseAtomicModel

apply_out_stat(ret: Dict[str, numpy.ndarray], atype: numpy.ndarray)

Apply the stat to each atomic output. The developer may override the method to define how the bias is applied to the atomic output of the model.

Parameters

ret
The returned dict by the forward_atomic method

atype
The atom types. nf x nloc

_varsize(shape: List[int]) → int

_get_bias_index(kk: str) → int

_fetch_out_stat(keys: List[str]) → Tuple[Dict[str, numpy.ndarray], Dict[str, numpy.ndarray]]

class deepmd.dpmodel.atomic_model.DPAtomicModel(descriptor, fitting, type_map: List[str], **kwargs)

Bases: *deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicModel*

Model give atomic prediction of some physical property.

Parameters

descriptor
Descriptor

fitting_net
Fitting net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

fitting_output_def() → deepmd.dpmodel.output_def.FittingOutputDef

Get the output def of the fitting net.

get_rcut() → float

Get the cut-off radius.

get_sel() → List[int]

Get the neighbor selection.

mixed_types() → bool

If true, the model 1. assumes total number of atoms aligned across frames; 2. uses a neighbor list that does not distinguish different atomic types.

If false, the model 1. assumes total number of atoms of each atom type aligned across frames; 2. uses a neighbor list that distinguishes different atomic types.

forward_atomic(extended_coord: numpy.ndarray, extended_atype: numpy.ndarray, nlist: numpy.ndarray, mapping: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]

Models' atomic predictions.

Parameters

extended_coord
: coodinates in extended region

extended_atype
: atomic type in extended region

nlist
: neighbor list. nf x nloc x nsel

mapping
: mapps the extended indices to local indices. nf x nall

fparam
: frame parameter. nf x ndf

aparam
: atomic parameter. nf x nloc x nda

Returns

result_dict
: the result dict, defined by the FittingOutputDef.

serialize() → dict

classmethod deserialize(data) → DPAtomicModel

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim).

If False, the shape is (nframes, nloc, ndim).

class deepmd.dpmodel.atomic_model.DPZBLLinearEnergyAtomicModel(dp_model: deepmd.dpmodel.atomic_model.dp_atomic_mod
zbl_model: deepmd.dpmodel.atomic_model.pairtab_atomic_
sw_rmin: float, sw_rmax: float,
type_map: List[str],
smin_alpha: float | None = 0.1,
**kwargs)

Bases: *LinearEnergyAtomicModel*

Model linearly combine a list of AtomicModels.

Parameters

dp_model
The DPAtomicModel being combined.

zbl_model
The PairTable model being combined.

sw_rmin
The lower boundary of the interpolation between short-range tabulated interaction and DP.

sw_rmax
The upper boundary of the interpolation between short-range tabulated interaction and DP.

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

smin_alpha
The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin.

serialize() → dict

classmethod deserialize(data) → DPZBLLinearEnergyAtomicModel

_compute_weight(extended_coord: numpy.ndarray, extended_atype: numpy.ndarray, nlists_: List[numpy.ndarray]) → List[numpy.ndarray]

ZBL weight.

Returns

List[**np.ndarray**]
the atomic ZBL weight for interpolation. (nframes, nloc, 1)

class deepmd.dpmodel.atomic_model.LinearEnergyAtomicModel(models: List[deepmd.dpmodel.atomic_model.base_atomic_mod
type_map: List[str], **kwargs)

Bases: *deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicModel*

Linear model make linear combinations of several existing models.

Parameters

models
[**list**[*DPAtomicModel* or *PairTabAtomicModel*]] A list of models to be combined. PairTabAtomicModel must be used together with a DPAtomicModel.

type_map
[list[str]] Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

mixed_types() → bool

If true, the model 1. assumes total number of atoms aligned across frames; 2. uses a neighbor list that does not distinguish different atomic types.

If false, the model 1. assumes total number of atoms of each atom type aligned across frames; 2. uses a neighbor list that distinguishes different atomic types.

get_rcut() → float

Get the cut-off radius.

get_type_map() → List[str]

Get the type map.

get_model_rcuts() → List[float]

Get the cut-off radius for each individual models.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_model_nsels() → List[int]

Get the processed sels for each individual models. Not distinguishing types.

get_model_sels() → List[int | List[int]]

Get the sels for each individual models.

_sort_rcuts_sels() → Tuple[List[float], List[int]]

forward_atomic(extended_coord, extended_atype, nlist, mapping: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]

Return atomic prediction.

Parameters

extended_coord
coodinates in extended region, (nframes, nall * 3)

extended_atype
atomic type in extended region, (nframes, nall)

nlist
neighbor list, (nframes, nloc, nsel).

mapping
mapps the extended indices to local indices.

fparam
frame parameter. (nframes, ndf)

aparam
atomic parameter. (nframes, nloc, nda)

Returns

result_dict
the result dict, defined by the fitting net output def.

static remap_atype(ori_map: List[str], new_map: List[str]) → numpy.ndarray

This method is used to map the atype from the common type_map to the original type_map of indivial AtomicModels.

Parameters

ori_map
[List[str]] The original type map of an AtomicModel.

new_map
[List[str]] The common type map of the DPZBLLinearEnergyAtomicModel, created by the get_type_map method, must be a subset of the ori_map.

Returns

np.ndarray

fitting_output_def() → deepmd.dpmodel.output_def.FittingOutputDef

Get the output def of developer implemented atomic models.

serialize() → dict

classmethod deserialize(data: dict) → LinearEnergyAtomicModel

_compute_weight(extended_coord: numpy.ndarray, extended_atype: numpy.ndarray, nlists_: List[numpy.ndarray]) → List[numpy.ndarray]

This should be a list of user defined weights that matches the number of models to be combined.

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim).

If False, the shape is (nframes, nloc, ndim).

deepmd.dpmodel.atomic_model.**make_base_atomic_model**(t_tensor, fwd_method_name: str = 'forward_atomic')

Make the base class for the atomic model.

Parameters

t_tensor
The type of the tensor. used in the type hint.

fwd_method_name
Name of the forward method. For dpmodels, it should be "call". For torch models, it should be "forward".

class deepmd.dpmodel.atomic_model.**PairTabAtomicModel**(tab_file: str, rcut: float, sel: int | List[int], type_map: List[str], rcond: float | None = None, atom_ener: List[float] | None = None, **kwargs)

Bases: *deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicModel*

Pairwise tabulation energy model.

This model can be used to tabulate the pairwise energy between atoms for either short-range or long-range interactions, such as D3, LJ, ZBL, etc. It should not be used alone, but rather as one submodel of a linear (sum) model, such as DP+D3.

Do not put the model on the first model of a linear model, since the linear model fetches the type map from the first model.

At this moment, the model does not smooth the energy at the cutoff radius, so one needs to make sure the energy has been smoothed to zero.

Parameters

tab_file
: [`str`] The path to the tabulation file.

rcut
: [`float`] The cutoff radius.

sel
: [`int` or `list`[`int`]] The maxmum number of atoms in the cut-off radius.

type_map
: [`list`[`str`]] Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

fitting_output_def() → deepmd.dpmodel.output_def.FittingOutputDef

Get the output def of developer implemented atomic models.

get_rcut() → float

Get the cut-off radius.

get_type_map() → List[str]

Get the type map.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_nsel() → int

Returns the total number of selected neighboring atoms in the cut-off radius.

mixed_types() → bool

If true, the model 1. assumes total number of atoms aligned across frames; 2. uses a neighbor list that does not distinguish different atomic types.

If false, the model 1. assumes total number of atoms of each atom type aligned across frames; 2. uses a neighbor list that distinguishes different atomic types.

serialize() → dict

classmethod deserialize(data) → PairTabAtomicModel

forward_atomic(extended_coord, extended_atype, nlist, mapping: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]

_pair_tabulated_inter(nlist: numpy.ndarray, i_type: numpy.ndarray, j_type: numpy.ndarray, rr: numpy.ndarray) → numpy.ndarray

Pairwise tabulated energy.

Parameters

nlist
[np.ndarray] The unmasked neighbour list. (nframes, nloc)

i_type
[np.ndarray] The integer representation of atom type for all local atoms for all frames. (nframes, nloc)

j_type
[np.ndarray] The integer representation of atom type for all neighbour atoms of all local atoms for all frames. (nframes, nloc, nnei)

rr
[np.ndarray] The salar distance vector between two atoms. (nframes, nloc, nnei)

Returns

np.ndarray
The masked atomic energy for all local atoms for all frames. (nframes, nloc, nnei)

Raises

Exception
If the distance is beyond the table.

Notes

This function is used to calculate the pairwise energy between two atoms. It uses a table containing cubic spline coefficients calculated in PairTab.

static _get_pairwise_dist(coords: numpy.ndarray, nlist: numpy.ndarray) → numpy.ndarray

Get pairwise distance dr.

Parameters

coords
[np.ndarray] The coordinate of the atoms, shape of (nframes, nall, 3).

nlist
The masked nlist, shape of (nframes, nloc, nnei).

Returns

np.ndarray
The pairwise distance between the atoms (nframes, nloc, nnei).

static _extract_spline_coefficient(i_type: numpy.ndarray, j_type: numpy.ndarray, idx: numpy.ndarray, tab_data: numpy.ndarray, nspline: int) → numpy.ndarray

Extract the spline coefficient from the table.

Parameters

i_type
[np.ndarray] The integer representation of atom type for all local atoms for all frames. (nframes, nloc)

j_type
[np.ndarray] The integer representation of atom type for all neighbour atoms of all local atoms for all frames. (nframes, nloc, nnei)

idx
[np.ndarray] The index of the spline coefficient. (nframes, nloc, nnei)

tab_data
[np.ndarray] The table storing all the spline coefficient. (ntype, ntype, nspline, 4)

nspline
[int] The number of splines in the table.

Returns

np.ndarray
The spline coefficient. (nframes, nloc, nnei, 4), shape may be squeezed.

static _calculate_ener(coef: numpy.ndarray, uu: numpy.ndarray) → numpy.ndarray

Calculate energy using spline coeeficients.

Parameters

coef
[np.ndarray] The spline coefficients. (nframes, nloc, nnei, 4)

uu
[np.ndarray] The atom displancemnt used in interpolation and extrapolation (nframes, nloc, nnei)

Returns

np.ndarray
The atomic energy for all local atoms for all frames. (nframes, nloc, nnei)

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim).

If False, the shape is (nframes, nloc, ndim).

deepmd.dpmodel.descriptor

Submodules

deepmd.dpmodel.descriptor.base_descriptor

Module Contents

deepmd.dpmodel.descriptor.base_descriptor.**BaseDescriptor**

deepmd.dpmodel.descriptor.dpa1

Module Contents

Classes

DescrptDPA1	Attention-based descriptor which is proposed in the pretrainable DPA-1[1] model.
NeighborGatedAttention	The unit operation of a native model.
NeighborGatedAttentionLayer	The unit operation of a native model.
GatedAttentionLayer	The unit operation of a native model.

Functions

np_softmax(x[, axis])	
np_normalize(x[, axis])	

Attributes

__version__

deepmd.dpmodel.descriptor.dpa1.**__version__** = 'unknown'

deepmd.dpmodel.descriptor.dpa1.**np_softmax**(x, axis=-1)

deepmd.dpmodel.descriptor.dpa1.**np_normalize**(x, axis=-1)

class deepmd.dpmodel.descriptor.dpa1.DescrptDPA1(rcut: float, rcut_smth: float, sel: List[int] | int, ntypes: int, neuron: List[int] = [25, 50, 100], axis_neuron: int = 8, tebd_dim: int = 8, tebd_input_mode: str = 'concat', resnet_dt: bool = False, trainable: bool = True, type_one_side: bool = False, attn: int = 128, attn_layer: int = 2, attn_dotr: bool = True, attn_mask: bool = False, exclude_types: List[List[int]] = [], env_protection: float = 0.0, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, scaling_factor=1.0, normalize: bool = True, temperature: float | None = None, trainable_ln: bool = True, ln_eps: float | None = 1e-05, smooth_type_embedding: bool = True, concat_output_tebd: bool = True, spin: Any | None = None, stripped_type_embedding: bool | None = None, seed: int | None = None)

Bases: *deepmd.dpmodel.NativeOP*, *deepmd.dpmodel.descriptor.base_descriptor.BaseDescriptor*

Attention-based descriptor which is proposed in the pretrainable DPA-1[1] model.

This descriptor, $\mathcal{D}^i \in \mathbb{R}^{M \times M_<}$, is given by

$$\mathcal{D}^i = \frac{1}{N_c^2}(\hat{\mathcal{G}}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \hat{\mathcal{G}}^i_<,$$

where $\hat{\mathcal{G}}^i$ represents the embedding matrix:math:mathcal{G}^i after additional self-attention mechanism and $\mathcal{R}^i$ is defined by the full case in the se_e2_a descriptor. Note that we obtain $\mathcal{G}^i$ using the type embedding method by default in this descriptor.

To perform the self-attention mechanism, the queries $\mathcal{Q}^{i,l} \in \mathbb{R}^{N_c \times d_k}$, keys $\mathcal{K}^{i,l} \in \mathbb{R}^{N_c \times d_k}$, and values $\mathcal{V}^{i,l} \in \mathbb{R}^{N_c \times d_v}$ are first obtained:

$$\left(\mathcal{Q}^{i,l}\right)_j = Q_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$

$$\left(\mathcal{K}^{i,l}\right)_j = K_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$

$$\left(\mathcal{V}^{i,l}\right)_j = V_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$

where Q_l, K_l, V_l represent three trainable linear transformations that output the queries and keys of dimension d_k and values of dimension d_v, and l is the index of the attention layer. The input embedding matrix to the attention layers, denoted by $\mathcal{G}^{i,0}$, is chosen as the two-body embedding matrix.

Then the scaled dot-product attention method is adopted:

$$A(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{V}^{i,l}, \mathcal{R}^{i,l}) = \varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right)\mathcal{V}^{i,l},$$

where $\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right) \in \mathbb{R}^{N_c \times N_c}$ is attention weights. In the original attention method, one typically has $\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}\right) = \mathrm{softmax}\left(\frac{\mathcal{Q}^{i,l}(\mathcal{K}^{i,l})^T}{\sqrt{d_k}}\right)$, with $\sqrt{d_k}$ being the normalization temperature. This is slightly modified to incorporate the angular information:

$$\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right) = \mathrm{softmax}\left(\frac{\mathcal{Q}^{i,l}(\mathcal{K}^{i,l})^T}{\sqrt{d_k}}\right) \odot \hat{\mathcal{R}}^i(\hat{\mathcal{R}}^i)^T,$$

where $\hat{\mathcal{R}}^i \in \mathbb{R}^{N_c \times 3}$ denotes normalized relative coordinates,
$\hat{\mathcal{R}}^i_j = \frac{\boldsymbol{r}_{ij}}{\|\boldsymbol{r}_{ij}\|}$ and $\odot$ means element-wise multiplication.

Then layer normalization is added in a residual way to finally obtain the self-attention local embedding matrix
$\hat{\mathcal{G}}^i = \mathcal{G}^{i,L_a}$ after L_a attention layers:[^1]

$$\mathcal{G}^{i,l} = \mathcal{G}^{i,l-1} + \text{LayerNorm}(A(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{V}^{i,l}, \mathcal{R}^{i,l})).$$

Parameters

rcut: float
The cut-off radius r_c

rcut_smth: float
From where the environment matrix should be smoothed r_s

sel
[`list`[`int`], `int`] list[int]: sel[i] specifies the maxmum number of type i atoms in the cut-off radius int: the total maxmum number of atoms in the cut-off radius

ntypes
[`int`] Number of element types

neuron
[`list`[`int`]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron: int
Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

tebd_dim: int
Dimension of the type embedding

tebd_input_mode: str
The input mode of the type embedding. Supported modes are [“concat”, “strip”]. - “concat”: Concatenate the type embedding with the smoothed radial information as the union input for the embedding network. - “strip”: Use a separated embedding network for the type embedding and combine the output with the radial embedding network output.

resnet_dt: bool
Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable: bool
If the weights of this descriptors are trainable.

trainable_ln: bool
Whether to use trainable shift and scale weights in layer normalization.

ln_eps: float, Optional
The epsilon value for layer normalization.

type_one_side: bool
If ‘False’, type embeddings of both neighbor and central atoms are considered. If ‘True’, only type embeddings of neighbor atoms are considered. Default is ‘False’.

attn: int
Hidden dimension of the attention vectors

attn_layer: int
Number of attention layers

attn_dotr: bool
: If dot the angular gate to the attention weights

attn_mask: bool
: (Only support False to keep consistent with other backend references.) (Not used in this version. True option is not implemented.) If mask the diagonal of attention weights

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

env_protection: float
: Protection parameter to prevent division by zero errors during environment matrix calculations.

set_davg_zero: bool
: Set the shift of embedding net input to zero.

activation_function: str
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision: str
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

scaling_factor: float
: The scaling factor of normalization in calculations of attention weights. If temperature is None, the scaling of attention weights is (N_dim * scaling_factor)**0.5

normalize: bool
: Whether to normalize the hidden vectors in attention weights calculation.

temperature: float
: If not None, the scaling of attention weights is temperature itself.

smooth_type_embedding: bool
: Whether to use smooth process in attention weights calculation.

concat_output_tebd: bool
: Whether to concat type embedding at the output of the descriptor.

stripped_type_embedding: bool, Optional
: (Deprecated, kept only for compatibility.) Whether to strip the type embedding into a separate embedding network. Setting this parameter to True is equivalent to setting tebd_input_mode to ‘strip’. Setting it to False is equivalent to setting tebd_input_mode to ‘concat’. The default value is None, which means the tebd_input_mode setting will be used instead.

spin
: (Only support None to keep consistent with other backend references.) (Not used in this version. Not-none option is not implemented.) The old implementation of deepspin.

References

[1]

property dim_out
Returns the output dimension of this descriptor.

__setitem__(key, value)

__getitem__(key)

get_dim_out()
Returns the output dimension of this descriptor.

get_dim_emb()
Returns the embedding (g2) dimension of this descriptor.

get_rcut()
Returns cutoff radius.

get_sel()
Returns cutoff radius.

mixed_types()
If true, the discriptor 1. assumes total number of atoms aligned across frames; 2. requires a neighbor list that does not distinguish different atomic types.

If false, the discriptor 1. assumes total number of atoms of each atom type aligned across frames; 2. requires a neighbor list that distinguishes different atomic types.

abstract share_params(base_class, shared_level, resume=False)
Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

get_ntypes() → int
Returns the number of element types.

abstract compute_input_stats(merged: List[dict], path: deepmd.utils.path.DPPath | None = None)
Update mean and stddev for descriptor elements.

cal_g(ss, embedding_idx)

cal_g_strip(ss, embedding_idx)

reinit_exclude(exclude_types: List[Tuple[int, int]] = [])

call(coord_ext, atype_ext, nlist, mapping: numpy.ndarray | None = None)
Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping from extended to lcoal region. not used by this descriptor.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. this descriptor returns None

h2
The rotationally equivariant pair-partical representation. this descriptor returns None

sw
The smooth switch function.

serialize() → dict

Serialize the descriptor to dict.

classmethod deserialize(data: dict) → DescrptDPA1

Deserialize from dict.

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

class deepmd.dpmodel.descriptor.dpa1.NeighborGatedAttention(layer_num: int, nnei: int, embed_dim: int, hidden_dim: int, dotr: bool = False, do_mask: bool = False, scaling_factor: float = 1.0, normalize: bool = True, temperature: float | None = None, trainable_ln: bool = True, ln_eps: float = 1e-05, smooth: bool = True, precision: str = DEFAULT_PRECISION)

Bases: *deepmd.dpmodel.NativeOP*

The unit operation of a native model.

call(input_G, nei_mask, input_r: numpy.ndarray | None = None, sw: numpy.ndarray | None = None)

Forward pass in NumPy implementation.

__getitem__(key)

__setitem__(key, value)

serialize()

Serialize the networks to a dict.

Returns

dict
The serialized networks.

classmethod deserialize(data: dict) → NeighborGatedAttention

Deserialize the networks from a dict.

Parameters

data
[dict] The dict to deserialize from.

class deepmd.dpmodel.descriptor.dpa1.NeighborGatedAttentionLayer(nnei: int, embed_dim: int, hidden_dim: int, dotr: bool = False, do_mask: bool = False, scaling_factor: float = 1.0, normalize: bool = True, temperature: float | None = None, trainable_ln: bool = True, ln_eps: float = 1e-05, smooth: bool = True, precision: str = DEFAULT_PRECISION)

Bases: *deepmd.dpmodel.NativeOP*

The unit operation of a native model.

call(x, nei_mask, input_r: numpy.ndarray | None = None, sw: numpy.ndarray | None = None)

Forward pass in NumPy implementation.

serialize() → dict

Serialize the networks to a dict.

Returns

dict
The serialized networks.

classmethod deserialize(data) → NeighborGatedAttentionLayer

Deserialize the networks from a dict.

Parameters

data
[dict] The dict to deserialize from.

class deepmd.dpmodel.descriptor.dpa1.GatedAttentionLayer(nnei: int, embed_dim: int, hidden_dim: int, dotr: bool = False, do_mask: bool = False, scaling_factor: float = 1.0, normalize: bool = True, temperature: float | None = None, bias: bool = True, smooth: bool = True, precision: str = DEFAULT_PRECISION)

Bases: *deepmd.dpmodel.NativeOP*

The unit operation of a native model.

call(query, nei_mask, input_r=None, sw=None, attnw_shift=20.0)

Forward pass in NumPy implementation.

serialize()

classmethod deserialize(data)

deepmd.dpmodel.descriptor.hybrid

Module Contents

Classes

DescrptHybrid	Concate a list of descriptors to form a new descriptor.

class deepmd.dpmodel.descriptor.hybrid.DescrptHybrid(list: List[deepmd.dpmodel.descriptor.base_descriptor.BaseDescrip | Dict[str, Any]])

Bases: *deepmd.dpmodel.descriptor.base_descriptor.BaseDescriptor*, *deepmd.dpmodel.common.NativeOP*

Concate a list of descriptors to form a new descriptor.

Parameters

list

[list][List[Union[BaseDescriptor, Dict[str, *Any*]]]] Build a descriptor from the concatenation of the list of descriptors. The descriptor can be either an object or a dictionary.

get_rcut() → float

Returns the cut-off radius.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_ntypes() → int

Returns the number of element types.

get_dim_out() → int

Returns the output dimension.

get_dim_emb() → int

Returns the output dimension.

mixed_types()

Returns if the descriptor requires a neighbor list that distinguish different atomic types or not.

abstract share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

compute_input_stats(merged: List[dict], path: deepmd.utils.path.DPPath | None = None)

Update mean and stddev for descriptor elements.

call(coord_ext, atype_ext, nlist, mapping: numpy.ndarray | None = None)

Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping, not required by this descriptor.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3.

g2
The rotationally invariant pair-partical representation.

h2
The rotationally equivariant pair-partical representation.

sw
The smooth switch function.

classmethod update_sel(global_jdata: dict, local_jdata: dict) → dict

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

serialize() → dict

Serialize the obj to dict.

classmethod deserialize(data: dict) → DescrptHybrid

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

BD
The deserialized descriptor

deepmd.dpmodel.descriptor.make_base_descriptor

Module Contents

Functions

make_base_descriptor(t_tensor[, fwd_method_name])	Make the base class for the descriptor.

deepmd.dpmodel.descriptor.make_base_descriptor.**make_base_descriptor**(t_tensor, fwd_method_name: str = 'forward')

Make the base class for the descriptor.

Parameters

t_tensor
The type of the tensor. used in the type hint.

fwd_method_name
Name of the forward method. For dpmodels, it should be "call". For torch models, it should be "forward".

deepmd.dpmodel.descriptor.se_e2_a

Module Contents

Classes

DescrptSeA	DeepPot-SE constructed from all information (both angular and radial) of

Attributes

__version__

deepmd.dpmodel.descriptor.se_e2_a.__version__ = 'unknown'

class deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, type_one_side: bool = True, exclude_types: List[List[int]] = [], env_protection: float = 0.0, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, spin: Any | None = None, seed: int | None = None)

Bases: *deepmd.dpmodel.NativeOP*, *deepmd.dpmodel.descriptor.base_descriptor.BaseDescriptor*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor $\mathcal{D}^i \in \mathcal{R}^{M_1 \times M_2}$ is given by [1]

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}^i_<$$

where $\mathcal{R}^i \in \mathbb{R}^{N \times 4}$ is the coordinate matrix, and each row of $\mathcal{R}^i$ can be constructed as follows

$$(\mathcal{R}^i)_j = [\begin{array}{c} s(r_{ji}) \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{array}]$$

where $\mathrm{R}_{ji} = \mathrm{R}_j - \mathrm{R}_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \|\mathrm{R}_{ji}\|$ is its norm. The switching function $s(r)$ is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r}\{(\frac{r-r_s}{r_c-r_s})^3(-6(\frac{r-r_s}{r_c-r_s})^2 + 15\frac{r-r_s}{r_c-r_s} - 10) + 1\}, & r_s \le r < r_c \\ 0, & r \ge r_c \end{cases}$$

Each row of the embedding matrix $\mathcal{G}^i \in \mathbb{R}^{N \times M_1}$ consists of outputs of a embedding network $\mathcal{N}$ of $s(r_{ji})$:

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}^i_< \in \mathbb{R}^{N \times M_2}$ takes first M_2 columns of $\mathcal{G}^i$. The equation of embedding network $\mathcal{N}$ can be found at *deepmd.tf.utils.network.embedding_net()*.

Parameters

rcut
: The cut-off radius r_c

rcut_smth
: From where the environment matrix should be smoothed r_s

sel
: [list[int]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

env_protection: float
: Protection parameter to prevent division by zero errors during environment matrix calculations.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

multi_task
: If the model has multi fitting nets to train.

spin
: The deepspin object.

References

[1]

property dim_out

Returns the output dimension of this descriptor.

__setitem__(key, value)

__getitem__(key)

get_dim_out()

Returns the output dimension of this descriptor.

get_dim_emb()

Returns the embedding (g2) dimension of this descriptor.

get_rcut()

Returns cutoff radius.

get_sel()

Returns cutoff radius.

mixed_types()

Returns if the descriptor requires a neighbor list that distinguish different atomic types or not.

abstract share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

get_ntypes() → int

Returns the number of element types.

abstract compute_input_stats(merged: List[dict], path: deepmd.utils.path.DPPath | None = None)

Update mean and stddev for descriptor elements.

cal_g(ss, embedding_idx)

reinit_exclude(exclude_types: List[Tuple[int, int]] = [])

call(coord_ext, atype_ext, nlist, mapping: numpy.ndarray | None = None)

Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping from extended to lcoal region. not used by this descriptor.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. this descriptor returns None

h2
The rotationally equivariant pair-partical representation. this descriptor returns None

sw
The smooth switch function.

serialize() → dict
Serialize the descriptor to dict.

classmethod deserialize(data: dict) → DescrptSeA
Deserialize from dict.

classmethod update_sel(global_jdata: dict, local_jdata: dict)
Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

deepmd.dpmodel.descriptor.se_r

Module Contents

Classes

DescrptSeR	DeepPot-SE_R constructed from only the radial imformation of atomic configurations.

Attributes

__version__

deepmd.dpmodel.descriptor.se_r.__version__ = 'unknown'

class deepmd.dpmodel.descriptor.se_r.DescrptSeR(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], resnet_dt: bool = False, trainable: bool = True, type_one_side: bool = True, exclude_types: List[List[int]] = [], env_protection: float = 0.0, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, spin: Any | None = None, seed: int | None = None)

Bases: *deepmd.dpmodel.NativeOP*, *deepmd.dpmodel.descriptor.base_descriptor.BaseDescriptor*

DeepPot-SE_R constructed from only the radial imformation of atomic configurations.

Parameters

rcut
: The cut-off radius r_c

rcut_smth
: From where the environment matrix should be smoothed r_s

sel
: [list[int]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

multi_task
: If the model has multi fitting nets to train.

spin
: The deepspin object.

References

[1]

property dim_out
: Returns the output dimension of this descriptor.

__setitem__(key, value)

__getitem__(key)

get_dim_out()
: Returns the output dimension of this descriptor.

abstract get_dim_emb()

Returns the embedding (g2) dimension of this descriptor.

get_rcut()

Returns cutoff radius.

get_sel()

Returns cutoff radius.

mixed_types()

Returns if the descriptor requires a neighbor list that distinguish different atomic types or not.

abstract share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

get_ntypes() → int

Returns the number of element types.

abstract compute_input_stats(merged: List[dict], path: deepmd.utils.path.DPPath | None = None)

Update mean and stddev for descriptor elements.

cal_g(ss, ll)

call(coord_ext, atype_ext, nlist, mapping: numpy.ndarray | None = None)

Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping from extended to lcoal region. not used by this descriptor.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. this descriptor returns None

h2
The rotationally equivariant pair-partical representation. this descriptor returns None

sw
The smooth switch function.

serialize() → dict
Serialize the descriptor to dict.

classmethod deserialize(data: dict) → DescrptSeR
Deserialize from dict.

classmethod update_sel(global_jdata: dict, local_jdata: dict)
Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

Package Contents

Classes

DescrptDPA1	Attention-based descriptor which is proposed in the pretrainable DPA-1[1] model.
DescrptHybrid	Concate a list of descriptors to form a new descriptor.
DescrptSeA	DeepPot-SE constructed from all information (both angular and radial) of
DescrptSeR	DeepPot-SE_R constructed from only the radial information of atomic configurations.

Functions

make_base_descriptor	

class deepmd.dpmodel.descriptor.DescrptDPA1(rcut: float, rcut_smth: float, sel: List[int] | int, ntypes: int, neuron: List[int] = [25, 50, 100], axis_neuron: int = 8, tebd_dim: int = 8, tebd_input_mode: str = 'concat', resnet_dt: bool = False, trainable: bool = True, type_one_side: bool = False, attn: int = 128, attn_layer: int = 2, attn_dotr: bool = True, attn_mask: bool = False, exclude_types: List[List[int]] = [], env_protection: float = 0.0, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, scaling_factor=1.0, normalize: bool = True, temperature: float | None = None, trainable_ln: bool = True, ln_eps: float | None = 1e-05, smooth_type_embedding: bool = True, concat_output_tebd: bool = True, spin: Any | None = None, stripped_type_embedding: bool | None = None, seed: int | None = None)

Bases: *deepmd.dpmodel.NativeOP*, *deepmd.dpmodel.descriptor.base_descriptor.BaseDescriptor*

Attention-based descriptor which is proposed in the pretrainable DPA-1[1] model.

This descriptor, $\mathcal{D}^i \in \mathbb{R}^{M \times M_<}$, is given by

$$\mathcal{D}^i = \frac{1}{N_c^2}(\hat{\mathcal{G}}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \hat{\mathcal{G}}^i_<,$$

where $\hat{\mathcal{G}}^i$ represents the embedding matrix:math:mathcal{G}^i after additional self-attention mechanism and $\mathcal{R}^i$ is defined by the full case in the se_e2_a descriptor. Note that we obtain $\mathcal{G}^i$ using the type embedding method by default in this descriptor.

To perform the self-attention mechanism, the queries $\mathcal{Q}^{i,l} \in \mathbb{R}^{N_c \times d_k}$, keys $\mathcal{K}^{i,l} \in \mathbb{R}^{N_c \times d_k}$, and values $\mathcal{V}^{i,l} \in \mathbb{R}^{N_c \times d_v}$ are first obtained:

$$\left(\mathcal{Q}^{i,l}\right)_j = Q_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$

$$\left(\mathcal{K}^{i,l}\right)_j = K_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$

$$\left(\mathcal{V}^{i,l}\right)_j = V_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$

where Q_l, K_l, V_l represent three trainable linear transformations that output the queries and keys of dimension d_k and values of dimension d_v, and l is the index of the attention layer. The input embedding matrix to the attention layers, denoted by $\mathcal{G}^{i,0}$, is chosen as the two-body embedding matrix.

Then the scaled dot-product attention method is adopted:

$$A(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{V}^{i,l}, \mathcal{R}^{i,l}) = \varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right)\mathcal{V}^{i,l},$$

where $\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right) \in \mathbb{R}^{N_c \times N_c}$ is attention weights. In the original attention method, one typically has $\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}\right) = \mathrm{softmax}\left(\frac{\mathcal{Q}^{i,l}(\mathcal{K}^{i,l})^T}{\sqrt{d_k}}\right)$, with $\sqrt{d_k}$ being the normalization temperature. This is slightly modified to incorporate the angular information:

$$\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right) = \mathrm{softmax}\left(\frac{\mathcal{Q}^{i,l}(\mathcal{K}^{i,l})^T}{\sqrt{d_k}}\right) \odot \hat{\mathcal{R}}^i(\hat{\mathcal{R}}^i)^T,$$

where $\hat{\mathcal{R}}^i \in \mathbb{R}^{N_c \times 3}$ denotes normalized relative coordinates,
$\hat{\mathcal{R}}^i_j = \frac{\boldsymbol{r}_{ij}}{\|\boldsymbol{r}_{ij}\|}$ and $\odot$ means element-wise multiplication.

Then layer normalization is added in a residual way to finally obtain the self-attention local embedding matrix
$\hat{\mathcal{G}}^i = \mathcal{G}^{i,L_a}$ after L_a attention layers:[^1]

$$\mathcal{G}^{i,l} = \mathcal{G}^{i,l-1} + \text{LayerNorm}(A(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{V}^{i,l}, \mathcal{R}^{i,l})).$$

Parameters

rcut: float
: The cut-off radius r_c

rcut_smth: float
: From where the environment matrix should be smoothed r_s

sel
: [`list`[`int`], `int`] list[int]: sel[i] specifies the maxmum number of type i atoms in the cut-off radius int: the total maxmum number of atoms in the cut-off radius

ntypes
: [`int`] Number of element types

neuron
: [`list`[`int`]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron: int
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

tebd_dim: int
: Dimension of the type embedding

tebd_input_mode: str
: The input mode of the type embedding. Supported modes are [“concat”, “strip”]. - “concat”: Concatenate the type embedding with the smoothed radial information as the union input for the embedding network. - “strip”: Use a separated embedding network for the type embedding and combine the output with the radial embedding network output.

resnet_dt: bool
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable: bool
: If the weights of this descriptors are trainable.

trainable_ln: bool
: Whether to use trainable shift and scale weights in layer normalization.

ln_eps: float, Optional
: The epsilon value for layer normalization.

type_one_side: bool
: If ‘False’, type embeddings of both neighbor and central atoms are considered. If ‘True’, only type embeddings of neighbor atoms are considered. Default is ‘False’.

attn: int
: Hidden dimension of the attention vectors

attn_layer: int
: Number of attention layers

attn_dotr: bool
: If dot the angular gate to the attention weights

attn_mask: bool
: (Only support False to keep consistent with other backend references.) (Not used in this version. True option is not implemented.) If mask the diagonal of attention weights

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

env_protection: float
: Protection parameter to prevent division by zero errors during environment matrix calculations.

set_davg_zero: bool
: Set the shift of embedding net input to zero.

activation_function: str
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision: str
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

scaling_factor: float
: The scaling factor of normalization in calculations of attention weights. If temperature is None, the scaling of attention weights is (N_dim * scaling_factor)**0.5

normalize: bool
: Whether to normalize the hidden vectors in attention weights calculation.

temperature: float
: If not None, the scaling of attention weights is temperature itself.

smooth_type_embedding: bool
: Whether to use smooth process in attention weights calculation.

concat_output_tebd: bool
: Whether to concat type embedding at the output of the descriptor.

stripped_type_embedding: bool, Optional
: (Deprecated, kept only for compatibility.) Whether to strip the type embedding into a separate embedding network. Setting this parameter to True is equivalent to setting tebd_input_mode to ‘strip’. Setting it to False is equivalent to setting tebd_input_mode to ‘concat’. The default value is None, which means the tebd_input_mode setting will be used instead.

spin
: (Only support None to keep consistent with other backend references.) (Not used in this version. Not-none option is not implemented.) The old implementation of deepspin.

References

[1]

property dim_out
Returns the output dimension of this descriptor.

__setitem__(key, value)

__getitem__(key)

get_dim_out()
Returns the output dimension of this descriptor.

get_dim_emb()
Returns the embedding (g2) dimension of this descriptor.

get_rcut()
Returns cutoff radius.

get_sel()
Returns cutoff radius.

mixed_types()
If true, the discriptor 1. assumes total number of atoms aligned across frames; 2. requires a neighbor list that does not distinguish different atomic types.

If false, the discriptor 1. assumes total number of atoms of each atom type aligned across frames; 2. requires a neighbor list that distinguishes different atomic types.

abstract share_params(base_class, shared_level, resume=False)
Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

get_ntypes() → int
Returns the number of element types.

abstract compute_input_stats(merged: List[dict], path: deepmd.utils.path.DPPath | None = None)
Update mean and stddev for descriptor elements.

cal_g(ss, embedding_idx)

cal_g_strip(ss, embedding_idx)

reinit_exclude(exclude_types: List[Tuple[int, int]] = [])

call(coord_ext, atype_ext, nlist, mapping: numpy.ndarray | None = None)
Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping from extended to lcoal region. not used by this descriptor.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. this descriptor returns None

h2
The rotationally equivariant pair-partical representation. this descriptor returns None

sw
The smooth switch function.

serialize() → dict
Serialize the descriptor to dict.

classmethod deserialize(data: dict) → DescrptDPA1
Deserialize from dict.

classmethod update_sel(global_jdata: dict, local_jdata: dict)
Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

class deepmd.dpmodel.descriptor.DescrptHybrid(list: List[deepmd.dpmodel.descriptor.base_descriptor.BaseDescriptor | Dict[str, Any]])

Bases: *deepmd.dpmodel.descriptor.base_descriptor.BaseDescriptor*, *deepmd.dpmodel.common.NativeOP*

Concate a list of descriptors to form a new descriptor.

Parameters

list
[list][List[Union[BaseDescriptor, Dict[str, *Any*]]]] Build a descriptor from the concatenation of the list of descriptors. The descriptor can be either an object or a dictionary.

get_rcut() → float
Returns the cut-off radius.

get_sel() → List[int]
Returns the number of selected atoms for each type.

get_ntypes() → int

Returns the number of element types.

get_dim_out() → int

Returns the output dimension.

get_dim_emb() → int

Returns the output dimension.

mixed_types()

Returns if the descriptor requires a neighbor list that distinguish different atomic types or not.

abstract share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

compute_input_stats(merged: List[dict], path: deepmd.utils.path.DPPath | None = None)

Update mean and stddev for descriptor elements.

call(coord_ext, atype_ext, nlist, mapping: numpy.ndarray | None = None)

Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping, not required by this descriptor.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3.

g2
The rotationally invariant pair-partical representation.

h2
The rotationally equivariant pair-partical representation.

sw
The smooth switch function.

classmethod update_sel(global_jdata: dict, local_jdata: dict) → dict

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

serialize() → dict

Serialize the obj to dict.

classmethod deserialize(data: dict) → DescrptHybrid

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

BD
The deserialized descriptor

deepmd.dpmodel.descriptor.make_base_descriptor(t_tensor, fwd_method_name: str = 'forward')

Make the base class for the descriptor.

Parameters

t_tensor
The type of the tensor. used in the type hint.

fwd_method_name
Name of the forward method. For dpmodels, it should be "call". For torch models, it should be "forward".

class deepmd.dpmodel.descriptor.DescrptSeA(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, type_one_side: bool = True, exclude_types: List[List[int]] = [], env_protection: float = 0.0, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, spin: Any | None = None, seed: int | None = None)

Bases: *deepmd.dpmodel.NativeOP*, *deepmd.dpmodel.descriptor.base_descriptor.BaseDescriptor*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor $\mathcal{D}^i \in \mathcal{R}^{M_1 \times M_2}$ is given by [1]

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}^i_<$$

where $\mathcal{R}^i \in \mathbb{R}^{N \times 4}$ is the coordinate matrix, and each row of $\mathcal{R}^i$ can be constructed as follows

$$(\mathcal{R}^i)_j = [\ \frac{s(r_{ji})}{r_{ji}} \quad \frac{s(r_{ji})x_{ji}}{r_{ji}} \quad \frac{s(r_{ji})y_{ji}}{r_{ji}} \quad \frac{s(r_{ji})z_{ji}}{r_{ji}}\ ]$$

where $\mathrm{R}_{ji} = \mathrm{R}_j - \mathrm{R}_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \|\mathrm{R}_{ji}\|$ is its norm. The switching

function $s(r)$ is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r}\{(\frac{r-r_s}{r_c-r_s})^3(-6(\frac{r-r_s}{r_c-r_s})^2 + 15\frac{r-r_s}{r_c-r_s} - 10) + 1\}, & r_s \leq r < r_c \\ 0, & r \geq r_c \end{cases}$$

Each row of the embedding matrix $\mathcal{G}^i \in \mathbb{R}^{N \times M_1}$ consists of outputs of a embedding network $\mathcal{N}$ of $s(r_{ji})$:

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}^i_< \in \mathbb{R}^{N \times M_2}$ takes first M_2 columns of $\mathcal{G}^i$. The equation of embedding network $\mathcal{N}$ can be found at *deepmd.tf.utils.network.embedding_net()*.

Parameters

rcut
: The cut-off radius r_c

rcut_smth
: From where the environment matrix should be smoothed r_s

sel
: [list[int]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

env_protection: float
: Protection parameter to prevent division by zero errors during environment matrix calculations.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

multi_task
: If the model has multi fitting nets to train.

spin
: The deepspin object.

References

[1]

property dim_out
: Returns the output dimension of this descriptor.

__setitem__(key, value)

__getitem__(key)

get_dim_out()
: Returns the output dimension of this descriptor.

get_dim_emb()
: Returns the embedding (g2) dimension of this descriptor.

get_rcut()
: Returns cutoff radius.

get_sel()
: Returns cutoff radius.

mixed_types()
: Returns if the descriptor requires a neighbor list that distinguish different atomic types or not.

abstract share_params(base_class, shared_level, resume=False)
: Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

get_ntypes() → int
: Returns the number of element types.

abstract compute_input_stats(merged: List[dict], path: deepmd.utils.path.DPPath | None = None)
: Update mean and stddev for descriptor elements.

cal_g(ss, embedding_idx)

reinit_exclude(exclude_types: List[Tuple[int, int]] = [])

call(coord_ext, atype_ext, nlist, mapping: numpy.ndarray | None = None)
: Compute the descriptor.

  Parameters

  coord_ext
  : The extended coordinates of atoms. shape: nf x (nallx3)

  atype_ext
  : The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping from extended to lcoal region. not used by this descriptor.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. this descriptor returns None

h2
The rotationally equivariant pair-partical representation. this descriptor returns None

sw
The smooth switch function.

serialize() → dict

Serialize the descriptor to dict.

classmethod deserialize(data: dict) → DescrptSeA

Deserialize from dict.

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

class deepmd.dpmodel.descriptor.DescrptSeR(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], resnet_dt: bool = False, trainable: bool = True, type_one_side: bool = True, exclude_types: List[List[int]] = [], env_protection: float = 0.0, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, spin: Any | None = None, seed: int | None = None)

Bases: *deepmd.dpmodel.NativeOP*, *deepmd.dpmodel.descriptor.base_descriptor.BaseDescriptor*

DeepPot-SE_R constructed from only the radial imformation of atomic configurations.

Parameters

rcut
The cut-off radius r_c

rcut_smth
From where the environment matrix should be smoothed r_s

sel
: [list[int]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

multi_task
: If the model has multi fitting nets to train.

spin
: The deepspin object.

References

[1]

property dim_out
: Returns the output dimension of this descriptor.

__setitem__(key, value)

__getitem__(key)

get_dim_out()
: Returns the output dimension of this descriptor.

abstract get_dim_emb()
: Returns the embedding (g2) dimension of this descriptor.

get_rcut()
: Returns cutoff radius.

get_sel()
: Returns cutoff radius.

mixed_types()

Returns if the descriptor requires a neighbor list that distinguish different atomic types or not.

abstract share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

get_ntypes() → int

Returns the number of element types.

abstract compute_input_stats(merged: List[dict], path: deepmd.utils.path.DPPath | None = None)

Update mean and stddev for descriptor elements.

cal_g(ss, ll)

call(coord_ext, atype_ext, nlist, mapping: numpy.ndarray | None = None)

Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping from extended to lcoal region. not used by this descriptor.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. this descriptor returns None

h2
The rotationally equivariant pair-partical representation. this descriptor returns None

sw
The smooth switch function.

serialize() → dict

Serialize the descriptor to dict.

classmethod deserialize(data: dict) → DescrptSeR

Deserialize from dict.

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

deepmd.dpmodel.fitting

Submodules

deepmd.dpmodel.fitting.base_fitting

Module Contents

deepmd.dpmodel.fitting.base_fitting.**BaseFitting**

deepmd.dpmodel.fitting.dipole_fitting

Module Contents

Classes

DipoleFitting	Fitting rotationally equivariant diploe of the system.

class deepmd.dpmodel.fitting.dipole_fitting.**DipoleFitting**(ntypes: int, dim_descrpt: int, embedding_width: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, rcond: float | None = None, tot_ener_zero: bool = False, trainable: List[bool] | None = None, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, layer_name: List[str | None] | None = None, use_aparam_as_mask: bool = False, spin: Any = None, mixed_types: bool = False, exclude_types: List[int] = [], r_differentiable: bool = True, c_differentiable: bool = True, old_impl=False, seed: int | None = None)

Bases: *deepmd.dpmodel.fitting.general_fitting.GeneralFitting*

Fitting rotationally equivariant diploe of the system.

Parameters

ntypes
: The number of atom types.

dim_descrpt
: The dimension of the input descriptor.

embedding_width
: [int] The dimension of rotation matrix, m1.

neuron
: Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

rcond
: The condition number for the regression of atomic energy.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

activation_function
: The activation function $\boldsymbol{\phi}$ in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

layer_name
: [list[Optional[str]], optional] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
: If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
: If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

exclude_types
: Atomic contributions of the excluded atom types are set zero.

r_differentiable
: If the variable is differentiated with respect to coordinates of atoms. Only reduciable variable are differentiable.

c_differentiable
: If the variable is differentiated with respect to the cell tensor (pbc case). Only reduciable variable are differentiable.

_net_out_dim()

Set the FittingNet output dim.

serialize() → dict

Serialize the fitting to dict.

classmethod deserialize(data: dict) → deepmd.dpmodel.fitting.general_fitting.GeneralFitting

Deserialize the fitting.

Parameters

data
: [dict] The serialized data

Returns

BF
: The deserialized fitting

output_def()

Returns the output def of the fitting net.

call(descriptor: numpy.ndarray, atype: numpy.ndarray, gr: numpy.ndarray | None = None, g2: numpy.ndarray | None = None, h2: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]

Calculate the fitting.

Parameters

descriptor
: input descriptor. shape: nf x nloc x nd

atype
: the atom type. shape: nf x nloc

gr
: The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
: The rotationally invariant pair-partical representation. shape: nf x nloc x nnei x ng

h2
: The rotationally equivariant pair-partical representation. shape: nf x nloc x nnei x 3

fparam
: The frame parameter. shape: nf x nfp. nfp being numb_fparam

aparam
: The atomic parameter. shape: nf x nloc x nap. nap being numb_aparam

deepmd.dpmodel.fitting.dos_fitting

Module Contents

Classes

DOSFittingNet	Fitting the energy (or a rotationally invariant porperty of dim_out) of the system. The force and the virial can also be trained.

class deepmd.dpmodel.fitting.dos_fitting.**DOSFittingNet**(ntypes: int, dim_descrpt: int, numb_dos: int = 300, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, bias_dos: numpy.ndarray | None = None, rcond: float | None = None, trainable: bool | List[bool] = True, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, mixed_types: bool = False, exclude_types: List[int] = [], seed: int | None = None)

Bases: *deepmd.dpmodel.fitting.invar_fitting.InvarFitting*

Fitting the energy (or a rotationally invariant porperty of dim_out) of the system. The force and the virial can also be trained.

Lets take the energy fitting task as an example. The potential energy E is a fitting network function of the descriptor $\mathcal{D}$:

$$E(\mathcal{D}) = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \cdots \circ \mathcal{L}^{(1)} \circ \mathcal{L}^{(0)}$$

The first n hidden layers $\mathcal{L}^{(0)}, \cdots, \mathcal{L}^{(n-1)}$ are given by

$$\mathrm{y} = \mathcal{L}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \boldsymbol{\phi}(\mathrm{x}^T \mathrm{w} + \mathrm{b})$$

where $\mathrm{x} \in \mathbb{R}^{N_1}$ is the input vector and $\mathrm{y} \in \mathbb{R}^{N_2}$ is the output vector. $\mathrm{w} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathrm{b} \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable[i] is True. $\boldsymbol{\phi}$ is the activation function.

The output layer $\mathcal{L}^{(n)}$ is given by

$$\mathrm{y} = \mathcal{L}^{(n)}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \mathrm{x}^T \mathrm{w} + \mathrm{b}$$

where $\mathrm{x} \in \mathbb{R}^{N_{n-1}}$ is the input vector and $\mathrm{y} \in \mathbb{R}$ is the output scalar. $\mathrm{w} \in \mathbb{R}^{N_{n-1}}$ and $\mathrm{b} \in \mathbb{R}$ are weights and bias, respectively, both of which are trainable if trainable[n] is True.

Parameters

var_name
: The name of the output variable.

ntypes
: The number of atom types.

dim_descrpt
: The dimension of the input descriptor.

dim_out
: The dimension of the output fit property.

neuron
: Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

rcond
: The condition number for the regression of atomic energy.

bias_atom
: Bias for each element.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

atom_ener
: Specifying atomic energy contribution in vacuum. The set_davg_zero key in the descrptor should be set.

activation_function
: The activation function ϕ in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

layer_name
: [`list[Optional[str]]`, `optional`] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
: If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
: If false, different atomic types uses different fitting net, otherwise different atom types share the same fitting net.

exclude_types: List[int]
: Atomic contributions of the excluded atom types are set zero.

classmethod deserialize(data: dict) → deepmd.dpmodel.fitting.general_fitting.GeneralFitting

Deserialize the fitting.

Parameters

data
[dict] The serialized data

Returns

BF
The deserialized fitting

serialize() → dict

Serialize the fitting to dict.

deepmd.dpmodel.fitting.ener_fitting

Module Contents

Classes

EnergyFittingNet	Fitting the energy (or a rotationally invariant porperty of dim_out) of the system. The force and the virial can also be trained.

class deepmd.dpmodel.fitting.ener_fitting.**EnergyFittingNet**(ntypes: int, dim_descrpt: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, rcond: float | None = None, tot_ener_zero: bool = False, trainable: List[bool] | None = None, atom_ener: List[float] | None = None, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, layer_name: List[str | None] | None = None, use_aparam_as_mask: bool = False, spin: Any = None, mixed_types: bool = False, exclude_types: List[int] = [], seed: int | None = None)

Bases: *deepmd.dpmodel.fitting.invar_fitting.InvarFitting*

Fitting the energy (or a rotationally invariant porperty of dim_out) of the system. The force and the virial can also be trained.

Lets take the energy fitting task as an example. The potential energy E is a fitting network function of the descriptor $\mathcal{D}$:

$$E(\mathcal{D}) = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \cdots \circ \mathcal{L}^{(1)} \circ \mathcal{L}^{(0)}$$

The first n hidden layers $\mathcal{L}^{(0)}, \cdots, \mathcal{L}^{(n-1)}$ are given by

$$\mathrm{y} = \mathcal{L}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \boldsymbol{\phi}(\mathrm{x}^T \mathrm{w} + \mathrm{b})$$

where $\mathrm{x} \in \mathbb{R}^{N_1}$ is the input vector and $\mathrm{y} \in \mathbb{R}^{N_2}$ is the output vector. $\mathrm{w} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathrm{b} \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable[i] is True. $\boldsymbol{\phi}$ is the activation function.

The output layer $\mathcal{L}^{(n)}$ is given by

$$\mathrm{y} = \mathcal{L}^{(n)}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \mathrm{x}^T \mathrm{w} + \mathrm{b}$$

where $\mathrm{x} \in \mathbb{R}^{N_{n-1}}$ is the input vector and $\mathrm{y} \in \mathbb{R}$ is the output scalar. $\mathrm{w} \in \mathbb{R}^{N_{n-1}}$ and $\mathrm{b} \in \mathbb{R}$ are weights and bias, respectively, both of which are trainable if trainable[n] is True.

Parameters

var_name
: The name of the output variable.

ntypes
: The number of atom types.

dim_descrpt
: The dimension of the input descriptor.

dim_out
: The dimension of the output fit property.

neuron
: Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

rcond
: The condition number for the regression of atomic energy.

bias_atom
: Bias for each element.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

atom_ener
: Specifying atomic energy contribution in vacuum. The set_davg_zero key in the descrptor should be set.

activation_function
: The activation function $\boldsymbol{\phi}$ in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

layer_name
: [list[Optional[str]], optional] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
: If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
: If false, different atomic types uses different fitting net, otherwise different atom types share the same fitting net.

exclude_types: List[int]
: Atomic contributions of the excluded atom types are set zero.

classmethod deserialize(data: dict) → deepmd.dpmodel.fitting.general_fitting.GeneralFitting

Deserialize the fitting.

Parameters

data
: [dict] The serialized data

Returns

BF
: The deserialized fitting

serialize() → dict

Serialize the fitting to dict.

deepmd.dpmodel.fitting.general_fitting

Module Contents

Classes

GeneralFitting	General fitting class.

class deepmd.dpmodel.fitting.general_fitting.**GeneralFitting**(var_name: str, ntypes: int, dim_descrpt: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, bias_atom_e: numpy.ndarray | None = None, rcond: float | None = None, tot_ener_zero: bool = False, trainable: List[bool] | None = None, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, layer_name: List[str | None] | None = None, use_aparam_as_mask: bool = False, spin: Any = None, mixed_types: bool = True, exclude_types: List[int] = [], remove_vaccum_contribution: List[bool] | None = None)

Bases: *deepmd.dpmodel.NativeOP*, *deepmd.dpmodel.fitting.base_fitting.BaseFitting*

General fitting class.

Parameters

var_name
: The name of the output variable.

ntypes
: The number of atom types.

dim_descrpt
: The dimension of the input descriptor.

neuron
: Number of neurons N in each hidden layer of the fitting net

bias_atom_e
: Average enery per atom for each element.

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

rcond
: The condition number for the regression of atomic energy.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

activation_function
: The activation function ϕ in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

layer_name
: [list[Optional[str]], optional] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
: If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
: If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

exclude_types: List[int]
: Atomic contributions of the excluded atom types are set zero.

remove_vaccum_contribution: List[bool], optional
: Remove vaccum contribution before the bias is added. The list assigned each type. For mixed_types provide [True], otherwise it should be a list of the same length as ntypes signaling if or not removing the vaccum contribution for the atom types in the list.

abstract _net_out_dim()

Set the FittingNet output dim.

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

__setitem__(key, value)

__getitem__(key)

reinit_exclude(exclude_types: List[int] = [])

serialize() → dict

Serialize the fitting to dict.

classmethod deserialize(data: dict) → GeneralFitting

Deserialize the fitting.

Parameters

data
[dict] The serialized data

Returns

BF
The deserialized fitting

_call_common(descriptor: numpy.ndarray, atype: numpy.ndarray, gr: numpy.ndarray | None = None, g2: numpy.ndarray | None = None, h2: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]

Calculate the fitting.

Parameters

descriptor
input descriptor. shape: nf x nloc x nd

atype
the atom type. shape: nf x nloc

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. shape: nf x nloc x nnei x ng

h2
The rotationally equivariant pair-partical representation. shape: nf x nloc x nnei x 3

fparam
The frame parameter. shape: nf x nfp. nfp being numb_fparam

aparam
The atomic parameter. shape: nf x nloc x nap. nap being numb_aparam

deepmd.dpmodel.fitting.invar_fitting

Module Contents

Classes

InvarFitting	Fitting the energy (or a rotationally invariant porperty of dim_out) of the system. The force and the virial can also be trained.

class deepmd.dpmodel.fitting.invar_fitting.**InvarFitting**(var_name: str, ntypes: int, dim_descrpt: int, dim_out: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, bias_atom: numpy.ndarray | None = None, rcond: float | None = None, tot_ener_zero: bool = False, trainable: List[bool] | None = None, atom_ener: List[float] | None = None, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, layer_name: List[str | None] | None = None, use_aparam_as_mask: bool = False, spin: Any = None, mixed_types: bool = True, exclude_types: List[int] = [])

Bases: *deepmd.dpmodel.fitting.general_fitting.GeneralFitting*

Fitting the energy (or a rotationally invariant porperty of dim_out) of the system. The force and the virial can also be trained.

Lets take the energy fitting task as an example. The potential energy E is a fitting network function of the descriptor $\mathcal{D}$:

$$E(\mathcal{D}) = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \cdots \circ \mathcal{L}^{(1)} \circ \mathcal{L}^{(0)}$$

The first n hidden layers $\mathcal{L}^{(0)}, \cdots, \mathcal{L}^{(n-1)}$ are given by

$$\mathrm{y} = \mathcal{L}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \boldsymbol{\phi}(\mathrm{x}^T \mathrm{w} + \mathrm{b})$$

where $\mathrm{x} \in \mathbb{R}^{N_1}$ is the input vector and $\mathrm{y} \in \mathbb{R}^{N_2}$ is the output vector. $\mathrm{w} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathrm{b} \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable[i] is True. $\boldsymbol{\phi}$ is the activation function.

The output layer $\mathcal{L}^{(n)}$ is given by

$$\mathrm{y} = \mathcal{L}^{(n)}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \mathrm{x}^T \mathrm{w} + \mathrm{b}$$

where $\mathrm{x} \in \mathbb{R}^{N_{n-1}}$ is the input vector and $\mathrm{y} \in \mathbb{R}$ is the output scalar. $\mathrm{w} \in \mathbb{R}^{N_{n-1}}$ and $\mathrm{b} \in \mathbb{R}$ are weights and bias, respectively, both of which are trainable if trainable[n] is True.

Parameters

var_name
: The name of the output variable.

ntypes
: The number of atom types.

dim_descrpt
: The dimension of the input descriptor.

dim_out
: The dimension of the output fit property.

neuron
: Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

rcond
: The condition number for the regression of atomic energy.

bias_atom
: Bias for each element.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

atom_ener
: Specifying atomic energy contribution in vacuum. The set_davg_zero key in the descrptor should be set.

activation_function
: The activation function ϕ in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

layer_name
: [`list`[`Optional`[`str`]], `optional`] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
: If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
: If false, different atomic types uses different fitting net, otherwise different atom types share the same fitting net.

exclude_types: List[int]
: Atomic contributions of the excluded atom types are set zero.

serialize() → dict

Serialize the fitting to dict.

classmethod deserialize(data: dict) → deepmd.dpmodel.fitting.general_fitting.GeneralFitting

Deserialize the fitting.

Parameters

data
: [`dict`] The serialized data

Returns

BF
The deserialized fitting

_net_out_dim()
Set the FittingNet output dim.

abstract compute_output_stats(merged)
Update the output bias for fitting net.

output_def()
Returns the output def of the fitting net.

call(descriptor: numpy.ndarray, atype: numpy.ndarray, gr: numpy.ndarray | None = None, g2: numpy.ndarray | None = None, h2: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]
Calculate the fitting.

Parameters

descriptor
input descriptor. shape: nf x nloc x nd

atype
the atom type. shape: nf x nloc

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. shape: nf x nloc x nnei x ng

h2
The rotationally equivariant pair-partical representation. shape: nf x nloc x nnei x 3

fparam
The frame parameter. shape: nf x nfp. nfp being numb_fparam

aparam
The atomic parameter. shape: nf x nloc x nap. nap being numb_aparam

deepmd.dpmodel.fitting.make_base_fitting

Module Contents

Functions

make_base_fitting(t_tensor[, fwd_method_name])	Make the base class for the fitting.

deepmd.dpmodel.fitting.make_base_fitting.**make_base_fitting**(t_tensor, fwd_method_name: str = 'forward')
Make the base class for the fitting.

Parameters

t_tensor
: The type of the tensor. used in the type hint.

fwd_method_name
: Name of the forward method. For dpmodels, it should be “call”. For torch models, it should be “forward”.

deepmd.dpmodel.fitting.polarizability_fitting

Module Contents

Classes

PolarFitting	Fitting rotationally equivariant polarizability of the system.

class deepmd.dpmodel.fitting.polarizability_fitting.**PolarFitting**(ntypes: int, dim_descrpt: int, embedding_width: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, rcond: float | None = None, tot_ener_zero: bool = False, trainable: List[bool] | None = None, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, layer_name: List[str | None] | None = None, use_aparam_as_mask: bool = False, spin: Any = None, mixed_types: bool = False, exclude_types: List[int] = [], old_impl: bool = False, fit_diag: bool = True, scale: List[float] | None = None, shift_diag: bool = True, seed: int | None = None)

Bases: *deepmd.dpmodel.fitting.general_fitting.GeneralFitting*

Fitting rotationally equivariant polarizability of the system.

Parameters

ntypes
: The number of atom types.

dim_descrpt
: The dimension of the input descriptor.

embedding_width
: [int] The dimension of rotation matrix, m1.

neuron
: Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

rcond
: The condition number for the regression of atomic energy.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

activation_function
: The activation function $\boldsymbol{\phi}$ in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

layer_name
: [list[Optional[str]], optional] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
: If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
: If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

fit_diag
: [bool] Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.

scale
: [List[float]] The output of the fitting net (polarizability matrix) for type i atom will be scaled by scale[i]

shift_diag
: [bool] Whether to shift the diagonal part of the polarizability matrix. The shift operation is carried out after scale.

`_net_out_dim()`
Set the FittingNet output dim.

`__setitem__`(key, value)

`__getitem__`(key)

`serialize()` → dict
Serialize the fitting to dict.

`classmethod deserialize`(data: dict) → deepmd.dpmodel.fitting.general_fitting.GeneralFitting
Deserialize the fitting.

Parameters

data
[`dict`] The serialized data

Returns

BF
The deserialized fitting

`output_def()`
Returns the output def of the fitting net.

`call`(descriptor: numpy.ndarray, atype: numpy.ndarray, gr: numpy.ndarray | None = None, g2: numpy.ndarray | None = None, h2: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]
Calculate the fitting.

Parameters

descriptor
input descriptor. shape: nf x nloc x nd

atype
the atom type. shape: nf x nloc

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. shape: nf x nloc x nnei x ng

h2
The rotationally equivariant pair-partical representation. shape: nf x nloc x nnei x 3

fparam
The frame parameter. shape: nf x nfp. nfp being numb_fparam

aparam
The atomic parameter. shape: nf x nloc x nap. nap being numb_aparam

Package Contents

Classes

DipoleFitting	Fitting rotationally equivariant diploe of the system.
DOSFittingNet	Fitting the energy (or a rotationally invariant porperty of dim_out) of the system. The force and the virial can also be trained.
EnergyFittingNet	Fitting the energy (or a rotationally invariant porperty of dim_out) of the system. The force and the virial can also be trained.
InvarFitting	Fitting the energy (or a rotationally invariant porperty of dim_out) of the system. The force and the virial can also be trained.
PolarFitting	Fitting rotationally equivariant polarizability of the system.

Functions

make_base_fitting(t_tensor[, fwd_method_name])	Make the base class for the fitting.

class deepmd.dpmodel.fitting.**DipoleFitting**(ntypes: int, dim_descrpt: int, embedding_width: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, rcond: float | None = None, tot_ener_zero: bool = False, trainable: List[bool] | None = None, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, layer_name: List[str | None] | None = None, use_aparam_as_mask: bool = False, spin: Any = None, mixed_types: bool = False, exclude_types: List[int] = [], r_differentiable: bool = True, c_differentiable: bool = True, old_impl=False, seed: int | None = None)

Bases: *deepmd.dpmodel.fitting.general_fitting.GeneralFitting*

Fitting rotationally equivariant diploe of the system.

Parameters

ntypes
: The number of atom types.

dim_descrpt
: The dimension of the input descriptor.

embedding_width
: [int] The dimension of rotation matrix, m1.

neuron
: Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

rcond
: The condition number for the regression of atomic energy.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

activation_function
: The activation function $\boldsymbol{\phi}$ in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

layer_name
: [`list`[`Optional`[`str`]], `optional`] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
: If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
: If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

exclude_types
: Atomic contributions of the excluded atom types are set zero.

r_differentiable
: If the variable is differentiated with respect to coordinates of atoms. Only reduciable variable are differentiable.

c_differentiable
: If the variable is differentiated with respect to the cell tensor (pbc case). Only reduciable variable are differentiable.

_net_out_dim()

Set the FittingNet output dim.

serialize() → dict

Serialize the fitting to dict.

classmethod deserialize(data: dict) → deepmd.dpmodel.fitting.general_fitting.GeneralFitting

Deserialize the fitting.

Parameters

data
[dict] The serialized data

Returns

BF
The deserialized fitting

output_def()

Returns the output def of the fitting net.

call(descriptor: numpy.ndarray, atype: numpy.ndarray, gr: numpy.ndarray | None = None, g2: numpy.ndarray | None = None, h2: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]

Calculate the fitting.

Parameters

descriptor
input descriptor. shape: nf x nloc x nd

atype
the atom type. shape: nf x nloc

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. shape: nf x nloc x nnei x ng

h2
The rotationally equivariant pair-partical representation. shape: nf x nloc x nnei x 3

fparam
The frame parameter. shape: nf x nfp. nfp being numb_fparam

aparam
The atomic parameter. shape: nf x nloc x nap. nap being numb_aparam

class deepmd.dpmodel.fitting.DOSFittingNet(ntypes: int, dim_descrpt: int, numb_dos: int = 300, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, bias_dos: numpy.ndarray | None = None, rcond: float | None = None, trainable: bool | List[bool] = True, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, mixed_types: bool = False, exclude_types: List[int] = [], seed: int | None = None)

Bases: *deepmd.dpmodel.fitting.invar_fitting.InvarFitting*

Fitting the energy (or a rotationally invariant porperty of dim_out) of the system. The force and the virial can also be trained.

Lets take the energy fitting task as an example. The potential energy E is a fitting network function of the descriptor $\mathcal{D}$:

$$E(\mathcal{D}) = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \cdots \circ \mathcal{L}^{(1)} \circ \mathcal{L}^{(0)}$$

The first n hidden layers $\mathcal{L}^{(0)}, \cdots, \mathcal{L}^{(n-1)}$ are given by

$$\mathrm{y} = \mathcal{L}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \boldsymbol{\phi}(\mathrm{x}^T \mathrm{w} + \mathrm{b})$$

where $\mathrm{x} \in \mathbb{R}^{N_1}$ is the input vector and $\mathrm{y} \in \mathbb{R}^{N_2}$ is the output vector. $\mathrm{w} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathrm{b} \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable[i] is True. $\boldsymbol{\phi}$ is the activation function.

The output layer $\mathcal{L}^{(n)}$ is given by

$$\mathrm{y} = \mathcal{L}^{(n)}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \mathrm{x}^T \mathrm{w} + \mathrm{b}$$

where $\mathrm{x} \in \mathbb{R}^{N_{n-1}}$ is the input vector and $\mathrm{y} \in \mathbb{R}$ is the output scalar. $\mathrm{w} \in \mathbb{R}^{N_{n-1}}$ and $\mathrm{b} \in \mathbb{R}$ are weights and bias, respectively, both of which are trainable if trainable[n] is True.

Parameters

var_name
: The name of the output variable.

ntypes
: The number of atom types.

dim_descrpt
: The dimension of the input descriptor.

dim_out
: The dimension of the output fit property.

neuron
: Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

rcond
: The condition number for the regression of atomic energy.

bias_atom
: Bias for each element.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

atom_ener
: Specifying atomic energy contribution in vacuum. The set_davg_zero key in the descrptor should be set.

activation_function
: The activation function $\boldsymbol{\phi}$ in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

layer_name
: [list[Optional[str]], optional] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
: If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
: If false, different atomic types uses different fitting net, otherwise different atom types share the same fitting net.

exclude_types: List[int]
: Atomic contributions of the excluded atom types are set zero.

classmethod deserialize(data: dict) → deepmd.dpmodel.fitting.general_fitting.GeneralFitting

Deserialize the fitting.

Parameters

data
: [dict] The serialized data

Returns

BF
: The deserialized fitting

serialize() → dict

Serialize the fitting to dict.

class deepmd.dpmodel.fitting.EnergyFittingNet(ntypes: int, dim_descrpt: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, rcond: float | None = None, tot_ener_zero: bool = False, trainable: List[bool] | None = None, atom_ener: List[float] | None = None, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, layer_name: List[str | None] | None = None, use_aparam_as_mask: bool = False, spin: Any = None, mixed_types: bool = False, exclude_types: List[int] = [], seed: int | None = None)

Bases: *deepmd.dpmodel.fitting.invar_fitting.InvarFitting*

Fitting the energy (or a rotationally invariant porperty of dim_out) of the system. The force and the virial can also be trained.

Lets take the energy fitting task as an example. The potential energy E is a fitting network function of the descriptor $\mathcal{D}$:

$$E(\mathcal{D}) = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \cdots \circ \mathcal{L}^{(1)} \circ \mathcal{L}^{(0)}$$

The first n hidden layers $\mathcal{L}^{(0)}, \cdots, \mathcal{L}^{(n-1)}$ are given by

$$\mathrm{y} = \mathcal{L}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \boldsymbol{\phi}(\mathrm{x}^T \mathrm{w} + \mathrm{b})$$

where $\mathrm{x} \in \mathbb{R}^{N_1}$ is the input vector and $\mathrm{y} \in \mathbb{R}^{N_2}$ is the output vector. $\mathrm{w} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathrm{b} \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable[i] is True. $\boldsymbol{\phi}$ is the activation function.

The output layer $\mathcal{L}^{(n)}$ is given by

$$\mathrm{y} = \mathcal{L}^{(n)}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \mathrm{x}^T \mathrm{w} + \mathrm{b}$$

where $\mathrm{x} \in \mathbb{R}^{N_{n-1}}$ is the input vector and $\mathrm{y} \in \mathbb{R}$ is the output scalar. $\mathrm{w} \in \mathbb{R}^{N_{n-1}}$ and $\mathrm{b} \in \mathbb{R}$ are weights and bias, respectively, both of which are trainable if trainable[n] is True.

Parameters

var_name
: The name of the output variable.

ntypes
: The number of atom types.

dim_descrpt
: The dimension of the input descriptor.

dim_out
: The dimension of the output fit property.

neuron
: Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

rcond
: The condition number for the regression of atomic energy.

bias_atom
: Bias for each element.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

atom_ener
: Specifying atomic energy contribution in vacuum. The set_davg_zero key in the descrptor should be set.

activation_function
: The activation function $\boldsymbol{\phi}$ in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

layer_name
: [list[Optional[str]], optional] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
: If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
: If false, different atomic types uses different fitting net, otherwise different atom types share the same fitting net.

exclude_types: List[int]
: Atomic contributions of the excluded atom types are set zero.

classmethod deserialize(data: dict) → deepmd.dpmodel.fitting.general_fitting.GeneralFitting

Deserialize the fitting.

Parameters

data
: [dict] The serialized data

Returns

BF
: The deserialized fitting

serialize() → dict

Serialize the fitting to dict.

class deepmd.dpmodel.fitting.InvarFitting(var_name: str, ntypes: int, dim_descrpt: int, dim_out: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, bias_atom: numpy.ndarray | None = None, rcond: float | None = None, tot_ener_zero: bool = False, trainable: List[bool] | None = None, atom_ener: List[float] | None = None, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, layer_name: List[str | None] | None = None, use_aparam_as_mask: bool = False, spin: Any = None, mixed_types: bool = True, exclude_types: List[int] = [])

Bases: *deepmd.dpmodel.fitting.general_fitting.GeneralFitting*

Fitting the energy (or a rotationally invariant porperty of dim_out) of the system. The force and the virial can also be trained.

Lets take the energy fitting task as an example. The potential energy E is a fitting network function of the descriptor $\mathcal{D}$:

$$E(\mathcal{D}) = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \cdots \circ \mathcal{L}^{(1)} \circ \mathcal{L}^{(0)}$$

The first n hidden layers $\mathcal{L}^{(0)}, \cdots, \mathcal{L}^{(n-1)}$ are given by

$$\mathrm{y} = \mathcal{L}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \boldsymbol{\phi}(\mathrm{x}^T \mathrm{w} + \mathrm{b})$$

where $\mathrm{x} \in \mathbb{R}^{N_1}$ is the input vector and $\mathrm{y} \in \mathbb{R}^{N_2}$ is the output vector. $\mathrm{w} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathrm{b} \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable[i] is True. $\boldsymbol{\phi}$ is the activation function.

The output layer $\mathcal{L}^{(n)}$ is given by

$$\mathrm{y} = \mathcal{L}^{(n)}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \mathrm{x}^T \mathrm{w} + \mathrm{b}$$

where $\mathrm{x} \in \mathbb{R}^{N_{n-1}}$ is the input vector and $\mathrm{y} \in \mathbb{R}$ is the output scalar. $\mathrm{w} \in \mathbb{R}^{N_{n-1}}$ and $\mathrm{b} \in \mathbb{R}$ are weights and bias, respectively, both of which are trainable if trainable[n] is True.

Parameters

var_name
: The name of the output variable.

ntypes
: The number of atom types.

dim_descrpt
: The dimension of the input descriptor.

dim_out
: The dimension of the output fit property.

neuron
: Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

rcond
: The condition number for the regression of atomic energy.

bias_atom
: Bias for each element.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

atom_ener
: Specifying atomic energy contribution in vacuum. The set_davg_zero key in the descrptor should be set.

activation_function
: The activation function $\boldsymbol{\phi}$ in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

layer_name
[list[Optional[str]], optional] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
If false, different atomic types uses different fitting net, otherwise different atom types share the same fitting net.

exclude_types: List[int]
Atomic contributions of the excluded atom types are set zero.

serialize() → dict

Serialize the fitting to dict.

classmethod deserialize(data: dict) → deepmd.dpmodel.fitting.general_fitting.GeneralFitting

Deserialize the fitting.

Parameters

data
[dict] The serialized data

Returns

BF
The deserialized fitting

_net_out_dim()

Set the FittingNet output dim.

abstract compute_output_stats(merged)

Update the output bias for fitting net.

output_def()

Returns the output def of the fitting net.

call(descriptor: numpy.ndarray, atype: numpy.ndarray, gr: numpy.ndarray | None = None, g2: numpy.ndarray | None = None, h2: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]

Calculate the fitting.

Parameters

descriptor
input descriptor. shape: nf x nloc x nd

atype
the atom type. shape: nf x nloc

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. shape: nf x nloc x nnei x ng

h2
The rotationally equivariant pair-partical representation. shape: nf x nloc x nnei x 3

fparam
The frame parameter. shape: nf x nfp. nfp being numb_fparam

aparam
The atomic parameter. shape: nf x nloc x nap. nap being numb_aparam

deepmd.dpmodel.fitting.**make_base_fitting**(t_tensor, fwd_method_name: str = 'forward')

Make the base class for the fitting.

Parameters

t_tensor
The type of the tensor. used in the type hint.

fwd_method_name
Name of the forward method. For dpmodels, it should be "call". For torch models, it should be "forward".

class deepmd.dpmodel.fitting.**PolarFitting**(ntypes: int, dim_descrpt: int, embedding_width: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, rcond: float | None = None, tot_ener_zero: bool = False, trainable: List[bool] | None = None, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, layer_name: List[str | None] | None = None, use_aparam_as_mask: bool = False, spin: Any = None, mixed_types: bool = False, exclude_types: List[int] = [], old_impl: bool = False, fit_diag: bool = True, scale: List[float] | None = None, shift_diag: bool = True, seed: int | None = None)

Bases: *deepmd.dpmodel.fitting.general_fitting.GeneralFitting*

Fitting rotationally equivariant polarizability of the system.

Parameters

ntypes
The number of atom types.

dim_descrpt
The dimension of the input descriptor.

embedding_width
[int] The dimension of rotation matrix, m1.

neuron
Number of neurons N in each hidden layer of the fitting net

resnet_dt
Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
Number of frame parameter

numb_aparam
Number of atomic parameter

rcond
: The condition number for the regression of atomic energy.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

activation_function
: The activation function ϕ in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

layer_name
: [`list`[`Optional`[`str`]], `optional`] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
: If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
: If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

fit_diag
: [bool] Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.

scale
: [`List`[`float`]] The output of the fitting net (polarizability matrix) for type i atom will be scaled by scale[i]

shift_diag
: [bool] Whether to shift the diagonal part of the polarizability matrix. The shift operation is carried out after scale.

_net_out_dim()
: Set the FittingNet output dim.

__setitem__(key, value)

__getitem__(key)

serialize() → dict
: Serialize the fitting to dict.

classmethod deserialize(data: dict) → deepmd.dpmodel.fitting.general_fitting.GeneralFitting
: Deserialize the fitting.

  Parameters

data
[dict] The serialized data

Returns

BF
The deserialized fitting

output_def()
Returns the output def of the fitting net.

call(descriptor: numpy.ndarray, atype: numpy.ndarray, gr: numpy.ndarray | None = None, g2: numpy.ndarray | None = None, h2: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None) → Dict[str, numpy.ndarray]
Calculate the fitting.

Parameters

descriptor
input descriptor. shape: nf x nloc x nd

atype
the atom type. shape: nf x nloc

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. shape: nf x nloc x nnei x ng

h2
The rotationally equivariant pair-partical representation. shape: nf x nloc x nnei x 3

fparam
The frame parameter. shape: nf x nfp. nfp being numb_fparam

aparam
The atomic parameter. shape: nf x nloc x nap. nap being numb_aparam

deepmd.dpmodel.infer

Submodules

deepmd.dpmodel.infer.deep_eval

Module Contents

Classes

DeepEval	NumPy backend implementaion of DeepEval.

class deepmd.dpmodel.infer.deep_eval.DeepEval(model_file: str, output_def: deepmd.dpmodel.output_def.ModelOutputDef, *args: List[Any], auto_batch_size: bool | int | deepmd.dpmodel.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_eval.DeepEvalBackend*

NumPy backend implementaion of DeepEval.

Parameters

model_file
[Path] The name of the frozen model file.

output_def
[ModelOutputDef] The output definition of the model.

*args
[list] Positional arguments.

auto_batch_size
[bool or int or AutomaticBatchSize, default: False] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

property model_type: Type[*deepmd.infer.deep_eval.DeepEval*]

The the evaluator of the model type.

get_rcut() → float

Get the cutoff radius of this model.

get_ntypes() → int

Get the number of atom types of this model.

get_type_map() → List[str]

Get the type map (element name of the atom types) of this model.

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this DP.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this DP.

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

get_numb_dos() → int

Get the number of DOS.

`get_has_efield()`

Check if the model has efield.

`get_ntypes_spin()`

Get the number of spin atom types of this model.

`eval`(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, **kwargs: Dict[str, Any]) → Dict[str, numpy.ndarray]

Evaluate the energy, force and virial by using this DP.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

atomic
Calculate the atomic energy and virial

fparam
The frame parameter. The array can be of size : - nframes x dim_fparam. - dim_fparam. Then all frames are assumed to be provided with the same fparam.

aparam
The atomic parameter The array can be of size : - nframes x natoms x dim_aparam. - natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. - dim_aparam. Then all frames and atoms are provided with the same aparam.

**kwargs
Other parameters

Returns

output_dict
[dict] The output of the evaluation. The keys are the names of the output variables, and the values are the corresponding output arrays.

`_get_request_defs`(atomic: bool) → List[deepmd.dpmodel.output_def.OutputVariableDef]

Get the requested output definitions.

When atomic is True, all output_def are requested. When atomic is False, only energy (tensor), force, and virial are requested.

Parameters

atomic
[bool] Whether to request the atomic output.

Returns

list[**OutputVariableDef**]
The requested output definitions.

_eval_func(inner_func: Callable, numb_test: int, natoms: int) → Callable

Wrapper method with auto batch size.

Parameters

inner_func
[Callable] the method to be wrapped

numb_test
[int] number of tests

natoms
[int] number of atoms

Returns

Callable
the wrapper

_get_natoms_and_nframes(coords: numpy.ndarray, atom_types: numpy.ndarray, mixed_type: bool = False) → Tuple[int, int]

_eval_model(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: numpy.ndarray, request_defs: List[deepmd.dpmodel.output_def.OutputVariableDef])

_get_output_shape(odef, nframes, natoms)

deepmd.dpmodel.model

The model that takes the coordinates, cell and atom types as input and predicts some property. The models are automatically generated from atomic models by the deepmd.dpmodel.make_model method.

The make_model method does the reduction, auto-differentiation (dummy for dpmodels) and communication of the atomic properties according to output variable definition deepmd.dpmodel.OutputVariableDef.

All models should be inherited from *deepmd.dpmodel.model.base_model.BaseModel*. Models generated by make_model have already done it.

Submodules

deepmd.dpmodel.model.base_model

Module Contents

Classes

BaseModel	Base class for final exported model that will be directly used for inference.

Functions

make_base_model(→ Type[object])

deepmd.dpmodel.model.base_model.**make_base_model**() → Type[object]

class deepmd.dpmodel.model.base_model.**BaseModel**

Bases: *make_base_model*()

Base class for final exported model that will be directly used for inference.

The class defines some abstractmethods that will be directly called by the inference interface. If the final model class inherbits some of those methods from other classes, BaseModel should be inherited as the last class to ensure the correct method resolution order.

This class is for the DPModel backend.

See also:

deepmd.dpmodel.model.base_model.BaseBaseModel
Backend-independent BaseModel class.

get_model_def_script() → str
Get the model definition script.

deepmd.dpmodel.model.dp_model

Module Contents

Classes

DPModelCommon

class deepmd.dpmodel.model.dp_model.**DPModelCommon**

classmethod update_sel(global_jdata: dict, local_jdata: dict)
Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

deepmd.dpmodel.model.ener_model

Module Contents

Classes

EnergyModel	The unit operation of a native model.

Attributes

*DPEnergyModel_*

deepmd.dpmodel.model.ener_model.**DPEnergyModel_**

class deepmd.dpmodel.model.ener_model.**EnergyModel**(*args, **kwargs)

Bases: *deepmd.dpmodel.model.dp_model.DPModelCommon*, *DPEnergyModel_*

The unit operation of a native model.

deepmd.dpmodel.model.make_model

Module Contents

Functions

make_model(T_AtomicModel)	Make a model as a derived class of an atomic model.

deepmd.dpmodel.model.make_model.**make_model**(T_AtomicModel: Type[deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicM

Make a model as a derived class of an atomic model.

The model provide two interfaces.

1. the call_lower, that takes extended coordinates, atyps and neighbor list, and outputs the atomic and property and derivatives (if required) on the extended region.

2. the call, that takes coordinates, atypes and cell and predicts the atomic and reduced property, and derivatives (if required) on the local region.

Parameters

T_AtomicModel
The atomic model.

Returns

CM
The model.

deepmd.dpmodel.model.model

Module Contents

Functions

get_standard_model(...)	Get a EnergyModel from a dictionary.
get_spin_model(→ deepmd.dpmodel.model.spin_m	Get a spin model from a dictionary.
get_model(data)	Get a model from a dictionary.

deepmd.dpmodel.model.model.**get_standard_model**(data: dict) → deepmd.dpmodel.model.ener_model.EnergyModel

Get a EnergyModel from a dictionary.

Parameters

data
[dict] The data to construct the model.

deepmd.dpmodel.model.model.**get_spin_model**(data: dict) → deepmd.dpmodel.model.spin_model.SpinModel

Get a spin model from a dictionary.

Parameters

data
[dict] The data to construct the model.

deepmd.dpmodel.model.model.**get_model**(data: dict)

Get a model from a dictionary.

Parameters

data
[dict] The data to construct the model.

deepmd.dpmodel.model.spin_model

Module Contents

Classes

SpinModel	A spin model wrapper, with spin input preprocess and output split.

class deepmd.dpmodel.model.spin_model.**SpinModel**(backbone_model, spin: deepmd.utils.spin.Spin)

A spin model wrapper, with spin input preprocess and output split.

process_spin_input(coord, atype, spin)

Generate virtual coordinates and types, concat into the input.

process_spin_input_lower(extended_coord: numpy.ndarray, extended_atype: numpy.ndarray, extended_spin: numpy.ndarray, nlist: numpy.ndarray, mapping: numpy.ndarray | None = None)

Add extended_spin into extended_coord to generate virtual atoms, and extend nlist and mapping. Note that the final extended_coord_updated with shape [nframes, nall + nall, 3] has the following order: - [:, :nloc]: original nloc real atoms. - [:, nloc: nloc + nloc]: virtual atoms corresponding to nloc real atoms. - [:, nloc + nloc: nloc + nall]: ghost real atoms. - [:, nloc + nall: nall + nall]: virtual atoms corresponding to ghost real atoms.

process_spin_output(atype, out_tensor, add_mag: bool = True, virtual_scale: bool = True)

Split the output both real and virtual atoms, and scale the latter.

process_spin_output_lower(extended_atype, extended_out_tensor, nloc: int, add_mag: bool = True, virtual_scale: bool = True)

Split the extended output of both real and virtual atoms with switch, and scale the latter.

static extend_nlist(extended_atype, nlist)

static concat_switch_virtual(extended_tensor, extended_tensor_virtual, nloc: int)

get_type_map() → List[str]

Get the type map.

get_rcut()

Get the cut-off radius.

get_dim_fparam()

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam()

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model. Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim). If False, the shape is (nframes, nloc, ndim).

model_output_type() → List[str]

Get the output type for the model.

get_model_def_script() → str

Get the model definition script.

get_nnei() → int

Returns the total number of selected neighboring atoms in the cut-off radius.

get_nsel() → int

Returns the total number of selected neighboring atoms in the cut-off radius.

static has_spin() → bool

Returns whether it has spin input and output.

__getattr__(name)

Get attribute from the wrapped model.

serialize() → dict

classmethod deserialize(data) → SpinModel

call(coord, atype, spin, box: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, do_atomic_virial: bool = False) → Dict[str, numpy.ndarray]

Return model prediction.

Parameters

coord
The coordinates of the atoms. shape: nf x (nloc x 3)

atype
The type of atoms. shape: nf x nloc

spin
The spins of the atoms. shape: nf x (nloc x 3)

box
The simulation box. shape: nf x 9

fparam
frame parameter. nf x ndf

aparam
atomic parameter. nf x nloc x nda

do_atomic_virial
If calculate the atomic virial.

Returns

ret_dict
The result dict of type Dict[str,np.ndarray]. The keys are defined by the ModelOutputDef.

call_lower(extended_coord: numpy.ndarray, extended_atype: numpy.ndarray, extended_spin: numpy.ndarray, nlist: numpy.ndarray, mapping: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, do_atomic_virial: bool = False)

Return model prediction. Lower interface that takes extended atomic coordinates, types and spins, nlist, and mapping as input, and returns the predictions on the extended region. The predictions are not reduced.

Parameters

extended_coord
coordinates in extended region. nf x (nall x 3).

extended_atype
atomic type in extended region. nf x nall.

extended_spin
spins in extended region. nf x (nall x 3).

nlist
neighbor list. nf x nloc x nsel.

mapping
maps the extended indices to local indices. nf x nall.

fparam
: frame parameter. nf x ndf

aparam
: atomic parameter. nf x nloc x nda

do_atomic_virial
: whether calculate atomic virial

Returns

result_dict
: the result dict, defined by the FittingOutputDef.

deepmd.dpmodel.model.transform_output

Module Contents

Functions

fit_output_to_model_output(→ numpy.ndarray])	Dict[str,	Transform the output of the fitting network to
communicate_extended_output(→ numpy.ndarray])	Dict[str,	Transform the output of the model network defined on

deepmd.dpmodel.model.transform_output.**fit_output_to_model_output**(fit_ret: Dict[str, numpy.ndarray], fit_output_def: deepmd.dpmodel.output_def.FittingOutputDe coord_ext: numpy.ndarray, do_atomic_virial: bool = False) → Dict[str, numpy.ndarray]

Transform the output of the fitting network to the model output.

deepmd.dpmodel.model.transform_output.**communicate_extended_output**(model_ret: Dict[str, numpy.ndarray], model_output_def: deepmd.dpmodel.output_def.ModelOutputD mapping: numpy.ndarray, do_atomic_virial: bool = False) → Dict[str, numpy.ndarray]

Transform the output of the model network defined on local and ghost (extended) atoms to local atoms.

Package Contents

Classes

DPModelCommon	
SpinModel	A spin model wrapper, with spin input preprocess and output split.

Functions

make_model(T_AtomicModel)	Make a model as a derived class of an atomic model.

class deepmd.dpmodel.model.**DPModelCommon**

classmethod **update_sel**(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

deepmd.dpmodel.model.**make_model**(T_AtomicModel: Type[deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicModel])

Make a model as a derived class of an atomic model.

The model provide two interfaces.

1. the call_lower, that takes extended coordinates, atyps and neighbor list, and outputs the atomic and property and derivatives (if required) on the extended region.

2. the call, that takes coordinates, atypes and cell and predicts the atomic and reduced property, and derivatives (if required) on the local region.

Parameters

T_AtomicModel
The atomic model.

Returns

CM
The model.

class deepmd.dpmodel.model.**SpinModel**(backbone_model, spin: deepmd.utils.spin.Spin)

A spin model wrapper, with spin input preprocess and output split.

process_spin_input(coord, atype, spin)

Generate virtual coordinates and types, concat into the input.

process_spin_input_lower(extended_coord: numpy.ndarray, extended_atype: numpy.ndarray, extended_spin: numpy.ndarray, nlist: numpy.ndarray, mapping: numpy.ndarray | None = None)

Add extended_spin into extended_coord to generate virtual atoms, and extend nlist and mapping. Note that the final extended_coord_updated with shape [nframes, nall + nall, 3] has the following order: - [:, :nloc]: original nloc real atoms. - [:, nloc: nloc + nloc]: virtual atoms corresponding to nloc real atoms. - [:, nloc + nloc: nloc + nall]: ghost real atoms. - [:, nloc + nall: nall + nall]: virtual atoms corresponding to ghost real atoms.

process_spin_output(atype, out_tensor, add_mag: bool = True, virtual_scale: bool = True)

Split the output both real and virtual atoms, and scale the latter.

process_spin_output_lower(extended_atype, extended_out_tensor, nloc: int, add_mag: bool = True, virtual_scale: bool = True)

Split the extended output of both real and virtual atoms with switch, and scale the latter.

static extend_nlist(extended_atype, nlist)

static concat_switch_virtual(extended_tensor, extended_tensor_virtual, nloc: int)

get_type_map() → List[str]

Get the type map.

get_rcut()

Get the cut-off radius.

get_dim_fparam()

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam()

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model. Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim). If False, the shape is (nframes, nloc, ndim).

model_output_type() → List[str]

Get the output type for the model.

get_model_def_script() → str

Get the model definition script.

get_nnei() → int

Returns the total number of selected neighboring atoms in the cut-off radius.

get_nsel() → int

Returns the total number of selected neighboring atoms in the cut-off radius.

static has_spin() → bool

Returns whether it has spin input and output.

__getattr__(name)

Get attribute from the wrapped model.

serialize() → dict

classmethod deserialize(data) → SpinModel

call(coord, atype, spin, box: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, do_atomic_virial: bool = False) → Dict[str, numpy.ndarray]

Return model prediction.

Parameters

coord
The coordinates of the atoms. shape: nf x (nloc x 3)

atype
The type of atoms. shape: nf x nloc

spin
The spins of the atoms. shape: nf x (nloc x 3)

box
The simulation box. shape: nf x 9

fparam
frame parameter. nf x ndf

aparam
atomic parameter. nf x nloc x nda

do_atomic_virial
If calculate the atomic virial.

Returns

ret_dict
The result dict of type Dict[str,np.ndarray]. The keys are defined by the ModelOutputDef.

call_lower(extended_coord: numpy.ndarray, extended_atype: numpy.ndarray, extended_spin: numpy.ndarray, nlist: numpy.ndarray, mapping: numpy.ndarray | None = None, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, do_atomic_virial: bool = False)

Return model prediction. Lower interface that takes extended atomic coordinates, types and spins, nlist, and mapping as input, and returns the predictions on the extended region. The predictions are not reduced.

Parameters

extended_coord
coordinates in extended region. nf x (nall x 3).

extended_atype
atomic type in extended region. nf x nall.

extended_spin
spins in extended region. nf x (nall x 3).

nlist
neighbor list. nf x nloc x nsel.

mapping
maps the extended indices to local indices. nf x nall.

fparam
frame parameter. nf x ndf

aparam
atomic parameter. nf x nloc x nda

do_atomic_virial
whether calculate atomic virial

Returns

result_dict
the result dict, defined by the FittingOutputDef.

deepmd.dpmodel.utils

Submodules

deepmd.dpmodel.utils.batch_size

Module Contents

Classes

AutoBatchSize	Automatic batch size for NumPy.

class deepmd.dpmodel.utils.batch_size.**AutoBatchSize**(initial_batch_size: int = 1024, factor: float = 2.0)

Bases: *deepmd.utils.batch_size.AutoBatchSize*

Automatic batch size for NumPy.

is_gpu_available() → bool
Check if GPU is available.

Returns

bool
True if GPU is available

is_oom_error(e: Exception) → bool
Check if the exception is an OOM error.

Parameters

e
[Exception] Exception

deepmd.dpmodel.utils.env_mat

Module Contents

Classes

EnvMat	The unit operation of a native model.

Functions

compute_smooth_weight(distance, rmin, rmax)	Compute smooth weight for descriptor elements.
_make_env_mat(nlist, coord, rcut, ruct_smth[, ...])	Make smooth environment matrix.

deepmd.dpmodel.utils.env_mat.**compute_smooth_weight**(distance: numpy.ndarray, rmin: float, rmax: float)

Compute smooth weight for descriptor elements.

deepmd.dpmodel.utils.env_mat.**_make_env_mat**(nlist, coord, rcut: float, ruct_smth: float, radial_only: bool = False, protection: float = 0.0)

Make smooth environment matrix.

class deepmd.dpmodel.utils.env_mat.**EnvMat**(rcut, rcut_smth, protection: float = 0.0)

Bases: *deepmd.dpmodel.NativeOP*

The unit operation of a native model.

call(coord_ext: numpy.ndarray, atype_ext: numpy.ndarray, nlist: numpy.ndarray, davg: numpy.ndarray | None = None, dstd: numpy.ndarray | None = None, radial_only: bool = False) → numpy.ndarray | numpy.ndarray

Compute the environment matrix.

Parameters

nlist
The neighbor list. shape: nf x nloc x nnei

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

davg
The data avg. shape: nt x nnei x (4 or 1)

dstd
The inverse of data std. shape: nt x nnei x (4 or 1)

radial_only
Whether to only compute radial part of the environment matrix. If True, the output will be of shape nf x nloc x nnei x 1. Otherwise, the output will be of shape nf x nloc x nnei x 4. Default: False.

Returns

env_mat
The environment matrix. shape: nf x nloc x nnei x (4 or 1)

switch
The value of switch function. shape: nf x nloc x nnei

_call(nlist, coord_ext, radial_only)

serialize() → dict

classmethod deserialize(data: dict) → EnvMat

deepmd.dpmodel.utils.exclude_mask

Module Contents

Classes

AtomExcludeMask	Computes the type exclusion mask for atoms.
PairExcludeMask	Computes the type exclusion mask for atom pairs.

class deepmd.dpmodel.utils.exclude_mask.AtomExcludeMask(ntypes: int, exclude_types: List[int] = [])

Computes the type exclusion mask for atoms.

get_exclude_types()

get_type_mask()

build_type_exclude_mask(atype: numpy.ndarray)

Compute type exclusion mask for atoms.

Parameters

atype
The extended aotm types. shape: nf x natom

Returns

mask
The type exclusion mask for atoms. shape: nf x natom Element [ff,ii] being 0 if type(ii) is excluded, otherwise being 1.

class deepmd.dpmodel.utils.exclude_mask.PairExcludeMask(ntypes: int, exclude_types: List[Tuple[int, int]] = [])

Computes the type exclusion mask for atom pairs.

get_exclude_types()

build_type_exclude_mask(nlist: numpy.ndarray, atype_ext: numpy.ndarray)

Compute type exclusion mask for atom pairs.

Parameters

nlist
The neighbor list. shape: nf x nloc x nnei

atype_ext
The extended aotm types. shape: nf x nall

Returns

mask
The type exclusion mask for pair atoms of shape: nf x nloc x nnei. Element [ff,ii,jj] being 0 if type(ii), type(nlist[ff,ii,jj]) is excluded, otherwise being 1.

__contains__(item)

deepmd.dpmodel.utils.neighbor_stat

Module Contents

Classes

NeighborStatOP	Class for getting neighbor statics data information.
NeighborStat	Neighbor statistics using pure NumPy.

class deepmd.dpmodel.utils.neighbor_stat.NeighborStatOP(ntypes: int, rcut: float, mixed_types: bool)

Bases: *deepmd.dpmodel.common.NativeOP*

Class for getting neighbor statics data information.

Parameters

ntypes
The num of atom types

rcut
The cut-off radius

mixed_types
[bool, **optional**] If True, treat all types as a single type.

call(coord: numpy.ndarray, atype: numpy.ndarray, cell: numpy.ndarray | None) → Tuple[float, numpy.ndarray]

Calculate the neareest neighbor distance between atoms, maximum nbor size of atoms and the output data range of the environment matrix.

Parameters

coord
The coordinates of atoms.

atype
The atom types.

cell
The cell.

Returns

float
The minimal squared distance between two atoms

np.ndarray
The maximal number of neighbors

class deepmd.dpmodel.utils.neighbor_stat.NeighborStat(ntypes: int, rcut: float, mixed_type: bool = False)

Bases: *deepmd.utils.neighbor_stat.NeighborStat*

Neighbor statistics using pure NumPy.

Parameters

ntypes
[int] The num of atom types

rcut
[float] The cut-off radius

mixed_type
[bool, optional, default=False] Treat all types as a single type.

iterator(data: deepmd.utils.data_system.DeepmdDataSystem) → Iterator[Tuple[numpy.ndarray, float, str]]

Abstract method for producing data.

Yields

np.ndarray
The maximal number of neighbors

float
The squared minimal distance between two atoms

str
The directory of the data system

deepmd.dpmodel.utils.network

Native DP model format for multiple backends.

See issue #2982 for more information.

Module Contents

Classes

NativeLayer	Native representation of a layer.
LayerNorm	Implementation of Layer Normalization layer.
NetworkCollection	A collection of networks for multiple elements.

Functions

get_activation_fn(→ Callable[[numpy.ndarray], ...)	
make_multilayer_network(T_NetworkLayer, ModuleBase)	
make_embedding_network(T_Network, T_NetworkLayer)	
make_fitting_network(T_EmbeddingNet, T_Network, ...)	

Attributes

__version__	
NativeNet	
EmbeddingNet	
FittingNet	

deepmd.dpmodel.utils.network.__version__ = 'unknown'

class deepmd.dpmodel.utils.network.NativeLayer(num_in, num_out, bias: bool = True, use_timestep: bool = False, activation_function: str | None = None, resnet: bool = False, precision: str = DEFAULT_PRECISION)

Bases: *deepmd.dpmodel.NativeOP*

Native representation of a layer.

Parameters

w
: [np.ndarray, optional] The weights of the layer.

b
: [np.ndarray, optional] The biases of the layer.

idt
: [np.ndarray, optional] The identity matrix of the layer.

activation_function
: [str, optional] The activation function of the layer.

resnet
: [bool, optional] Whether the layer is a residual layer.

serialize() → dict

Serialize the layer to a dict.

Returns

dict
The serialized layer.

classmethod deserialize(data: dict) → NativeLayer
Deserialize the layer from a dict.

Parameters

data
[dict] The dict to deserialize from.

check_shape_consistency()

check_type_consistency()

__setitem__(key, value)

__getitem__(key)

dim_in() → int

dim_out() → int

call(x: numpy.ndarray) → numpy.ndarray
Forward pass.

Parameters

x
[np.ndarray] The input.

Returns

np.ndarray
The output.

deepmd.dpmodel.utils.network.**get_activation_fn**(activation_function: str) → Callable[[numpy.ndarray], numpy.ndarray]

class deepmd.dpmodel.utils.network.**LayerNorm**(num_in: int, eps: float = 1e-05, uni_init: bool = True, trainable: bool = True, precision: str = DEFAULT_PRECISION)

Bases: *NativeLayer*

Implementation of Layer Normalization layer.

Parameters

num_in
[int] The input dimension of the layer.

eps
[float, optional] A small value added to prevent division by zero in calculations.

uni_init
[bool, optional] If initialize the weights to be zeros and ones.

serialize() → dict
Serialize the layer to a dict.

Returns

dict
The serialized layer.

classmethod deserialize(data: dict) → LayerNorm
Deserialize the layer from a dict.

Parameters

data
[dict] The dict to deserialize from.

_check_shape_consistency()

__setitem__(key, value)

__getitem__(key)

dim_out() → int

call(x: numpy.ndarray) → numpy.ndarray
Forward pass.

Parameters

x
[np.ndarray] The input.

Returns

np.ndarray
The output.

static layer_norm_numpy(x, shape, weight=None, bias=None, eps=1e-05)

deepmd.dpmodel.utils.network.**make_multilayer_network**(T_NetworkLayer, ModuleBase)

deepmd.dpmodel.utils.network.**NativeNet**

deepmd.dpmodel.utils.network.**make_embedding_network**(T_Network, T_NetworkLayer)

deepmd.dpmodel.utils.network.**EmbeddingNet**

deepmd.dpmodel.utils.network.**make_fitting_network**(T_EmbeddingNet, T_Network, T_NetworkLayer)

deepmd.dpmodel.utils.network.**FittingNet**

class deepmd.dpmodel.utils.network.**NetworkCollection**(ndim: int, ntypes: int, network_type: str = 'network', networks: List[NativeNet | dict] = [])

A collection of networks for multiple elements.

The number of dimesions for types might be 0, 1, or 2. - 0: embedding or fitting with type embedding, in () - 1: embedding with type_one_side, or fitting, in (type_i) - 2: embedding without type_one_side, in (type_i, type_j)

Parameters

ndim
[int] The number of dimensions.

network_type
[str, optional] The type of the network.

networks
[dict, optional] The networks to initialize with.

NETWORK_TYPE_MAP: ClassVar[Dict[str, type]]

check_completeness()
Check whether the collection is complete.

Raises

RuntimeError
If the collection is incomplete.

_convert_key(key)

__getitem__(key)

__setitem__(key, value)

serialize() → dict
Serialize the networks to a dict.

Returns

dict
The serialized networks.

classmethod deserialize(data: dict) → NetworkCollection
Deserialize the networks from a dict.

Parameters

data
[dict] The dict to deserialize from.

deepmd.dpmodel.utils.nlist

Module Contents

Functions

build_neighbor_list(→ numpy.ndarray)	Build neightbor list for a single frame. keeps nsel neighbors.
nlist_distinguish_types(nlist, atype, sel)	Given a nlist that does not distinguish atom types, return a nlist that
get_multiple_nlist_key(→ str)	
build_multiple_neighbor_list(→ Dict[str, numpy.ndarray])	Input one neighbor list, and produce multiple neighbor lists with
extend_coord_with_ghosts(coord, atype, cell, rcut)	Extend the coordinates of the atoms by appending peridoc images.

deepmd.dpmodel.utils.nlist.build_neighbor_list(coord: numpy.ndarray, atype: numpy.ndarray, nloc: int, rcut: float, sel: int | List[int], distinguish_types: bool = True) → numpy.ndarray

Build neightbor list for a single frame. keeps nsel neighbors.

Parameters

coord
: [np.ndarray] exptended coordinates of shape [batch_size, nall x 3]

atype
: [np.ndarray] extended atomic types of shape [batch_size, nall] type < 0 the atom is treat as virtual atoms.

nloc
: [int] number of local atoms.

rcut
: [float] cut-off radius

sel
: [int or List[int]] maximal number of neighbors (of each type). if distinguish_types==True, nsel should be list and the length of nsel should be equal to number of types.

distinguish_types
: [bool] distinguish different types.

Returns

neighbor_list
: [np.ndarray] Neighbor list of shape [batch_size, nloc, nsel], the neighbors are stored in an ascending order. If the number of neighbors is less than nsel, the positions are masked with -1. The neighbor list of an atom looks like |------ nsel ------| xx xx xx xx -1 -1 -1 if distinguish_types==True and we have two types |---- nsel[0] -----| |---- nsel[1] -----| xx xx xx xx -1 -1 -1 xx xx xx -1 -1 -1 -1 For virtual atoms all neighboring positions are filled with -1.

deepmd.dpmodel.utils.nlist.nlist_distinguish_types(nlist: numpy.ndarray, atype: numpy.ndarray, sel: List[int])

Given a nlist that does not distinguish atom types, return a nlist that distinguish atom types.

deepmd.dpmodel.utils.nlist.get_multiple_nlist_key(rcut: float, nsel: int) → str

deepmd.dpmodel.utils.nlist.build_multiple_neighbor_list(coord: numpy.ndarray, nlist: numpy.ndarray, rcuts: List[float], nsels: List[int]) → Dict[str, numpy.ndarray]

Input one neighbor list, and produce multiple neighbor lists with different cutoff radius and numbers of selection out of it. The required rcuts and nsels should be smaller or equal to the input nlist.

Parameters

coord
: [np.ndarray] exptended coordinates of shape [batch_size, nall x 3]

nlist
: [np.ndarray] Neighbor list of shape [batch_size, nloc, nsel], the neighbors should be stored in an ascending order.

rcuts
: [List[float]] list of cut-off radius in ascending order.

nsels
: [List[int]] maximal number of neighbors in ascending order.

Returns

nlist_dict
[Dict[str, np.ndarray]] A dict of nlists, key given by get_multiple_nlist_key(rc, nsel) value being the corresponding nlist.

deepmd.dpmodel.utils.nlist.**extend_coord_with_ghosts**(coord: numpy.ndarray, atype: numpy.ndarray, cell: numpy.ndarray | None, rcut: float)

Extend the coordinates of the atoms by appending peridoc images. The number of images is large enough to ensure all the neighbors within rcut are appended.

Parameters

coord
[np.ndarray] original coordinates of shape [-1, nloc*3].

atype
[np.ndarray] atom type of shape [-1, nloc].

cell
[np.ndarray] simulation cell tensor of shape [-1, 9].

rcut
[float] the cutoff radius

Returns

extended_coord: np.ndarray
extended coordinates of shape [-1, nall*3].

extended_atype: np.ndarray
extended atom type of shape [-1, nall].

index_mapping: np.ndarray
maping extended index to the local index

deepmd.dpmodel.utils.region

Module Contents

Functions

phys2inter(→ numpy.ndarray)	Convert physical coordinates to internal(direct) coordinates.
inter2phys(→ numpy.ndarray)	Convert internal(direct) coordinates to physical coordinates.
normalize_coord(→ numpy.ndarray)	Apply PBC according to the atomic coordinates.
to_face_distance(→ numpy.ndarray)	Compute the to-face-distance of the simulation cell.
b_to_face_distance(cell)	

deepmd.dpmodel.utils.region.**phys2inter**(coord: numpy.ndarray, cell: numpy.ndarray) → numpy.ndarray

Convert physical coordinates to internal(direct) coordinates.

Parameters

coord
[np.ndarray] physical coordinates of shape [*, na, 3].

cell
[np.ndarray] simulation cell tensor of shape [*, 3, 3].

Returns

inter_coord: np.ndarray
the internal coordinates

deepmd.dpmodel.utils.region.**inter2phys**(coord: numpy.ndarray, cell: numpy.ndarray) → numpy.ndarray

Convert internal(direct) coordinates to physical coordinates.

Parameters

coord
[np.ndarray] internal coordinates of shape [*, na, 3].

cell
[np.ndarray] simulation cell tensor of shape [*, 3, 3].

Returns

phys_coord: np.ndarray
the physical coordinates

deepmd.dpmodel.utils.region.**normalize_coord**(coord: numpy.ndarray, cell: numpy.ndarray) → numpy.ndarray

Apply PBC according to the atomic coordinates.

Parameters

coord
[np.ndarray] orignal coordinates of shape [*, na, 3].

cell
[np.ndarray] simulation cell shape [*, 3, 3].

Returns

wrapped_coord: np.ndarray
wrapped coordinates of shape [*, na, 3].

deepmd.dpmodel.utils.region.**to_face_distance**(cell: numpy.ndarray) → numpy.ndarray

Compute the to-face-distance of the simulation cell.

Parameters

cell
[np.ndarray] simulation cell tensor of shape [*, 3, 3].

Returns

dist: np.ndarray
the to face distances of shape [*, 3]

deepmd.dpmodel.utils.region.**b_to_face_distance**(cell)

deepmd.dpmodel.utils.serialization

Module Contents

Classes

Counter	A callable counter.

Functions

traverse_model_dict(model_obj, callback[, is_variable])	Traverse a model dict and call callback on each variable.
save_dp_model(→ None)	Save a DP model to a file in the native format.
load_dp_model(→ dict)	Load a DP model from a file in the native format.

Attributes

__version__

deepmd.dpmodel.utils.serialization.__version__ = 'unknown'

deepmd.dpmodel.utils.serialization.traverse_model_dict(model_obj, callback: Callable, is_variable: bool = False)

Traverse a model dict and call callback on each variable.

Parameters

model_obj
[object] The model object to traverse.

callback
[callable()] The callback function to call on each variable.

is_variable
[bool, optional] Whether the current node is a variable.

Returns

object
The model object after traversing.

class deepmd.dpmodel.utils.serialization.Counter

A callable counter.

Examples

```
>>> counter = Counter()
>>> counter()
0
>>> counter()
1
```

`__call__`()

deepmd.dpmodel.utils.serialization.**save_dp_model**(filename: str, model_dict: dict) → None

Save a DP model to a file in the native format.

Parameters

filename
: [str] The filename to save to.

model_dict
: [dict] The model dict to save.

deepmd.dpmodel.utils.serialization.**load_dp_model**(filename: str) → dict

Load a DP model from a file in the native format.

Parameters

filename
: [str] The filename to load from.

Returns

dict
: The loaded model dict, including meta information.

deepmd.dpmodel.utils.type_embed

Module Contents

Classes

TypeEmbedNet	Type embedding network.

class deepmd.dpmodel.utils.type_embed.**TypeEmbedNet**(*, ntypes: int, neuron: List[int], resnet_dt: bool = False, activation_function: str = 'tanh', precision: str = 'default', trainable: bool = True, seed: int | None = None, padding: bool = False)

Bases: *deepmd.dpmodel.common.NativeOP*

Type embedding network.

Parameters

ntypes
: [int] Number of atom types

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

activation_function
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

padding
: Concat the zero padding to the output, as the default embedding of empty type.

call() → numpy.ndarray

Compute the type embedding network.

classmethod deserialize(data: dict)

Deserialize the model.

Parameters

data
: [dict] The serialized data

Returns

Model
: The deserialized model

serialize() → dict

Serialize the model.

Returns

dict
: The serialized data

deepmd.dpmodel.utils.update_sel

Module Contents

Classes

UpdateSel	Update the sel field in the descriptor.

class deepmd.dpmodel.utils.update_sel.UpdateSel

Bases: *deepmd.utils.update_sel.BaseUpdateSel*

Update the sel field in the descriptor.

property neighbor_stat: Type[*deepmd.dpmodel.utils.neighbor_stat.NeighborStat*]

hook(min_nbor_dist, max_nbor_size)

Package Contents

Classes

EnvMat	The unit operation of a native model.
AtomExcludeMask	Computes the type exclusion mask for atoms.
PairExcludeMask	Computes the type exclusion mask for atom pairs.
NativeLayer	Native representation of a layer.
NetworkCollection	A collection of networks for multiple elements.

Functions

make_embedding_network(T_Network, T_NetworkLayer)	
make_fitting_network(T_EmbeddingNet, T_Network, ...)	
make_multilayer_network(T_NetworkLayer, ModuleBase)	
build_multiple_neighbor_list(→ Dict[str, numpy.ndarray])	Input one neighbor list, and produce multiple neighbor lists with
build_neighbor_list(→ numpy.ndarray)	Build neightbor list for a single frame. keeps nsel neighbors.
extend_coord_with_ghosts(coord, atype, cell, rcut)	Extend the coordinates of the atoms by appending peridoc images.
get_multiple_nlist_key(→ str)	
nlist_distinguish_types(nlist, atype, sel)	Given a nlist that does not distinguish atom types, return a nlist that
inter2phys(→ numpy.ndarray)	Convert internal(direct) coordinates to physical coordinates.
normalize_coord(→ numpy.ndarray)	Apply PBC according to the atomic coordinates.
phys2inter(→ numpy.ndarray)	Convert physical coordinates to internal(direct) coordinates.
to_face_distance(→ numpy.ndarray)	Compute the to-face-distance of the simulation cell.
load_dp_model(→ dict)	Load a DP model from a file in the native format.
save_dp_model(→ None)	Save a DP model to a file in the native format.
traverse_model_dict(model_obj, callback[, is_variable])	Traverse a model dict and call callback on each variable.

Attributes

EmbeddingNet
FittingNet
NativeNet

class deepmd.dpmodel.utils.**EnvMat**(rcut, rcut_smth, protection: float = 0.0)

Bases: *deepmd.dpmodel.NativeOP*

The unit operation of a native model.

call(coord_ext: numpy.ndarray, atype_ext: numpy.ndarray, nlist: numpy.ndarray, davg: numpy.ndarray | None = None, dstd: numpy.ndarray | None = None, radial_only: bool = False) → numpy.ndarray | numpy.ndarray

Compute the environment matrix.

Parameters

nlist
: The neighbor list. shape: nf x nloc x nnei

coord_ext
: The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
: The extended aotm types. shape: nf x nall

davg
: The data avg. shape: nt x nnei x (4 or 1)

dstd
: The inverse of data std. shape: nt x nnei x (4 or 1)

radial_only
: Whether to only compute radial part of the environment matrix. If True, the output will be of shape nf x nloc x nnei x 1. Otherwise, the output will be of shape nf x nloc x nnei x 4. Default: False.

Returns

env_mat
: The environment matrix. shape: nf x nloc x nnei x (4 or 1)

switch
: The value of switch function. shape: nf x nloc x nnei

_call(nlist, coord_ext, radial_only)

serialize() → dict

classmethod deserialize(data: dict) → EnvMat

class deepmd.dpmodel.utils.**AtomExcludeMask**(ntypes: int, exclude_types: List[int] = [])

Computes the type exclusion mask for atoms.

get_exclude_types()

get_type_mask()

build_type_exclude_mask(atype: numpy.ndarray)

Compute type exclusion mask for atoms.

Parameters

atype
The extended aotm types. shape: nf x natom

Returns

mask
The type exclusion mask for atoms. shape: nf x natom Element [ff,ii] being 0 if type(ii) is excluded, otherwise being 1.

class deepmd.dpmodel.utils.**PairExcludeMask**(ntypes: int, exclude_types: List[Tuple[int, int]] = [])

Computes the type exclusion mask for atom pairs.

get_exclude_types()

build_type_exclude_mask(nlist: numpy.ndarray, atype_ext: numpy.ndarray)

Compute type exclusion mask for atom pairs.

Parameters

nlist
The neighbor list. shape: nf x nloc x nnei

atype_ext
The extended aotm types. shape: nf x nall

Returns

mask
The type exclusion mask for pair atoms of shape: nf x nloc x nnei. Element [ff,ii,jj] being 0 if type(ii), type(nlist[ff,ii,jj]) is excluded, otherwise being 1.

__contains__(item)

deepmd.dpmodel.utils.**EmbeddingNet**

deepmd.dpmodel.utils.**FittingNet**

class deepmd.dpmodel.utils.**NativeLayer**(num_in, num_out, bias: bool = True, use_timestep: bool = False, activation_function: str | None = None, resnet: bool = False, precision: str = DEFAULT_PRECISION)

Bases: *deepmd.dpmodel.NativeOP*

Native representation of a layer.

Parameters

w
[np.ndarray, optional] The weights of the layer.

b
[np.ndarray, optional] The biases of the layer.

idt
[np.ndarray, optional] The identity matrix of the layer.

activation_function
[str, optional] The activation function of the layer.

resnet
[bool, optional] Whether the layer is a residual layer.

serialize() → dict

Serialize the layer to a dict.

Returns

dict
The serialized layer.

classmethod deserialize(data: dict) → NativeLayer

Deserialize the layer from a dict.

Parameters

data
[dict] The dict to deserialize from.

check_shape_consistency()

check_type_consistency()

__setitem__(key, value)

__getitem__(key)

dim_in() → int

dim_out() → int

call(x: numpy.ndarray) → numpy.ndarray

Forward pass.

Parameters

x
[np.ndarray] The input.

Returns

np.ndarray
The output.

deepmd.dpmodel.utils.NativeNet

class deepmd.dpmodel.utils.NetworkCollection(ndim: int, ntypes: int, network_type: str = 'network', networks: List[NativeNet | dict] = [])

A collection of networks for multiple elements.

The number of dimesions for types might be 0, 1, or 2. - 0: embedding or fitting with type embedding, in () - 1: embedding with type_one_side, or fitting, in (type_i) - 2: embedding without type_one_side, in (type_i, type_j)

Parameters

ndim
[int] The number of dimensions.

network_type
[str, optional] The type of the network.

networks
[dict, optional] The networks to initialize with.

NETWORK_TYPE_MAP: ClassVar[Dict[str, type]]

check_completeness()
Check whether the collection is complete.

Raises

RuntimeError
If the collection is incomplete.

_convert_key(key)

__getitem__(key)

__setitem__(key, value)

serialize() → dict
Serialize the networks to a dict.

Returns

dict
The serialized networks.

classmethod deserialize(data: dict) → NetworkCollection
Deserialize the networks from a dict.

Parameters

data
[dict] The dict to deserialize from.

deepmd.dpmodel.utils.**make_embedding_network**(T_Network, T_NetworkLayer)

deepmd.dpmodel.utils.**make_fitting_network**(T_EmbeddingNet, T_Network, T_NetworkLayer)

deepmd.dpmodel.utils.**make_multilayer_network**(T_NetworkLayer, ModuleBase)

deepmd.dpmodel.utils.**build_multiple_neighbor_list**(coord: numpy.ndarray, nlist: numpy.ndarray, rcuts: List[float], nsels: List[int]) → Dict[str, numpy.ndarray]
Input one neighbor list, and produce multiple neighbor lists with different cutoff radius and numbers of selection out of it. The required rcuts and nsels should be smaller or equal to the input nlist.

Parameters

coord
[np.ndarray] exptended coordinates of shape [batch_size, nall x 3]

nlist
[np.ndarray] Neighbor list of shape [batch_size, nloc, nsel], the neighbors should be stored in an ascending order.

rcuts
[List[float]] list of cut-off radius in ascending order.

nsels
[List[int]] maximal number of neighbors in ascending order.

Returns

nlist_dict
[Dict[str, np.ndarray]] A dict of nlists, key given by get_multiple_nlist_key(rc, nsel) value being the corresponding nlist.

deepmd.dpmodel.utils.build_neighbor_list(coord: numpy.ndarray, atype: numpy.ndarray, nloc: int, rcut: float, sel: int | List[int], distinguish_types: bool = True) → numpy.ndarray

Build neightbor list for a single frame. keeps nsel neighbors.

Parameters

coord
[np.ndarray] exptended coordinates of shape [batch_size, nall x 3]

atype
[np.ndarray] extended atomic types of shape [batch_size, nall] type < 0 the atom is treat as virtual atoms.

nloc
[int] number of local atoms.

rcut
[float] cut-off radius

sel
[int or List[int]] maximal number of neighbors (of each type). if distinguish_types==True, nsel should be list and the length of nsel should be equal to number of types.

distinguish_types
[bool] distinguish different types.

Returns

neighbor_list
[np.ndarray] Neighbor list of shape [batch_size, nloc, nsel], the neighbors are stored in an ascending order. If the number of neighbors is less than nsel, the positions are masked with -1. The neighbor list of an atom looks like |------ nsel ------| xx xx xx xx -1 -1 -1 if distinguish_types==True and we have two types |---- nsel[0] -----| |---- nsel[1] -----| xx xx xx xx -1 -1 -1 xx xx xx -1 -1 -1 -1 For virtual atoms all neighboring positions are filled with -1.

deepmd.dpmodel.utils.extend_coord_with_ghosts(coord: numpy.ndarray, atype: numpy.ndarray, cell: numpy.ndarray | None, rcut: float)

Extend the coordinates of the atoms by appending peridoc images. The number of images is large enough to ensure all the neighbors within rcut are appended.

Parameters

coord
[np.ndarray] original coordinates of shape [-1, nloc*3].

atype
[np.ndarray] atom type of shape [-1, nloc].

cell
[np.ndarray] simulation cell tensor of shape [-1, 9].

rcut
[float] the cutoff radius

Returns

extended_coord: np.ndarray
extended coordinates of shape [-1, nall*3].

extended_atype: np.ndarray
extended atom type of shape [-1, nall].

index_mapping: np.ndarray
maping extended index to the local index

deepmd.dpmodel.utils.get_multiple_nlist_key(rcut: float, nsel: int) → str

deepmd.dpmodel.utils.nlist_distinguish_types(nlist: numpy.ndarray, atype: numpy.ndarray, sel: List[int])

Given a nlist that does not distinguish atom types, return a nlist that distinguish atom types.

deepmd.dpmodel.utils.inter2phys(coord: numpy.ndarray, cell: numpy.ndarray) → numpy.ndarray

Convert internal(direct) coordinates to physical coordinates.

Parameters

coord
[np.ndarray] internal coordinates of shape [*, na, 3].

cell
[np.ndarray] simulation cell tensor of shape [*, 3, 3].

Returns

phys_coord: np.ndarray
the physical coordinates

deepmd.dpmodel.utils.normalize_coord(coord: numpy.ndarray, cell: numpy.ndarray) → numpy.ndarray

Apply PBC according to the atomic coordinates.

Parameters

coord
[np.ndarray] orignal coordinates of shape [*, na, 3].

cell
[np.ndarray] simulation cell shape [*, 3, 3].

Returns

wrapped_coord: np.ndarray
wrapped coordinates of shape [*, na, 3].

deepmd.dpmodel.utils.phys2inter(coord: numpy.ndarray, cell: numpy.ndarray) → numpy.ndarray

Convert physical coordinates to internal(direct) coordinates.

Parameters

coord
[np.ndarray] physical coordinates of shape [*, na, 3].

cell
[np.ndarray] simulation cell tensor of shape [*, 3, 3].

Returns

inter_coord: `np.ndarray`
the internal coordinates

`deepmd.dpmodel.utils.to_face_distance`(cell: numpy.ndarray) → numpy.ndarray

Compute the to-face-distance of the simulation cell.

Parameters

cell
[`np.ndarray`] simulation cell tensor of shape [*, 3, 3].

Returns

dist: `np.ndarray`
the to face distances of shape [*, 3]

`deepmd.dpmodel.utils.load_dp_model`(filename: str) → dict

Load a DP model from a file in the native format.

Parameters

filename
[`str`] The filename to load from.

Returns

`dict`
The loaded model dict, including meta information.

`deepmd.dpmodel.utils.save_dp_model`(filename: str, model_dict: dict) → None

Save a DP model to a file in the native format.

Parameters

filename
[`str`] The filename to save to.

model_dict
[`dict`] The model dict to save.

`deepmd.dpmodel.utils.traverse_model_dict`(model_obj, callback: Callable, is_variable: bool = False)

Traverse a model dict and call callback on each variable.

Parameters

model_obj
[`object`] The model object to traverse.

callback
[`callable()`] The callback function to call on each variable.

is_variable
[bool, `optional`] Whether the current node is a variable.

Returns

`object`
The model object after traversing.

Submodules

deepmd.dpmodel.common

Module Contents

Classes

NativeOP	The unit operation of a native model.

Attributes

GLOBAL_NP_FLOAT_PRECISION
PRECISION_DICT
RESERVED_PRECISON_DICT
DEFAULT_PRECISION

deepmd.dpmodel.common.**GLOBAL_NP_FLOAT_PRECISION**

deepmd.dpmodel.common.**PRECISION_DICT**

deepmd.dpmodel.common.**RESERVED_PRECISON_DICT**

deepmd.dpmodel.common.**DEFAULT_PRECISION = 'float64'**

class deepmd.dpmodel.common.**NativeOP**

Bases: abc.ABC

The unit operation of a native model.

abstract call(*args, **kwargs)

Forward pass in NumPy implementation.

__call__(*args, **kwargs)

Forward pass in NumPy implementation.

deepmd.dpmodel.output_def

Module Contents

Classes

OutputVariableOperation	Defines the operation of the output variable.
OutputVariableCategory	Defines the category of the output variable.
OutputVariableDef	Defines the shape and other properties of the one output variable.
FittingOutputDef	Defines the shapes and other properties of the fitting network outputs.
ModelOutputDef	Defines the shapes and other properties of the model outputs.

Functions

check_shape(shape, def_shape)	Check if the shape satisfies the defined shape.
check_var(var, var_def)	
model_check_output(cls)	Check if the output of the Model is consistent with the definition.
fitting_check_output(cls)	Check if the output of the Fitting is consistent with the definition.
get_reduce_name(→ str)	
get_deriv_name(→ Tuple[str, str])	
get_deriv_name_mag(→ Tuple[str, str])	
get_hessian_name(→ str)	
apply_operation(→ int)	Apply an operation to the category of a variable definition.
check_operation_applied(→ bool)	Check if a operation has been applied to a variable definition.
do_reduce(→ Dict[str, OutputVariableDef])	
do_mask(→ Dict[str, OutputVariableDef])	
do_derivative(→ Tuple[Dict[str, OutputVariableDef], ...)	

deepmd.dpmodel.output_def.**check_shape**(shape: List[int], def_shape: List[int])

Check if the shape satisfies the defined shape.

deepmd.dpmodel.output_def.**check_var**(var, var_def)

deepmd.dpmodel.output_def.**model_check_output**(cls)

Check if the output of the Model is consistent with the definition.

Two methods are assumed to be provided by the Model: 1. Model.output_def that gives the output definition. 2. Model.__call__ that defines the forward path of the model.

deepmd.dpmodel.output_def.fitting_check_output(cls)

Check if the output of the Fitting is consistent with the definition.

Two methods are assumed to be provided by the Fitting: 1. Fitting.output_def that gives the output definition. 2. Fitting.__call__ defines the forward path of the fitting.

class deepmd.dpmodel.output_def.OutputVariableOperation

Bases: enum.IntEnum

Defines the operation of the output variable.

_NONE = 0

No operation.

REDU = 1

Reduce the output variable.

DERV_R = 2

Derivative w.r.t. coordinates.

DERV_C = 4

Derivative w.r.t. cell.

_SEC_DERV_R = 8

Second derivative w.r.t. coordinates.

MAG = 16

Magnetic output.

class deepmd.dpmodel.output_def.OutputVariableCategory

Bases: enum.IntEnum

Defines the category of the output variable.

OUT

Output variable. (e.g. atom energy)

REDU

Reduced output variable. (e.g. system energy)

DERV_R

Negative derivative w.r.t. coordinates. (e.g. force)

DERV_C

Atomic component of the virial, see PRB 104, 224202 (2021)

DERV_C_REDU

Virial, the transposed negative gradient with cell tensor times cell tensor, see eq 40 JCP 159, 054801 (2023).

DERV_R_DERV_R

Hession matrix, the second derivative w.r.t. coordinates.

DERV_R_MAG

Magnetic part of negative derivative w.r.t. coordinates. (e.g. magnetic force)

DERV_C_MAG

Magnetic part of atomic component of the virial.

class deepmd.dpmodel.output_def.OutputVariableDef(name: str, shape: List[int], reduciable: bool = False, r_differentiable: bool = False, c_differentiable: bool = False, atomic: bool = True, category: int = OutputVariableCategory.OUT.value, r_hessian: bool = False, magnetic: bool = False)

Defines the shape and other properties of the one output variable.

It is assume that the fitting network output variables for each local atom. This class defines one output variable, including its name, shape, reducibility and differentiability.

Parameters

name
Name of the output variable. Notice that the xxxx_redu, xxxx_derv_c, xxxx_derv_r are reserved names that should not be used to define variables.

shape
The shape of the variable. e.g. energy should be [1], dipole should be [3], polarizabilty should be [3,3].

reduciable
If the variable is reduced.

r_differentiable
If the variable is differentiated with respect to coordinates of atoms. Only reduciable variable are differentiable. Negative derivative w.r.t. coordinates will be calcualted. (e.g. force)

c_differentiable
If the variable is differentiated with respect to the cell tensor (pbc case). Only reduciable variable are differentiable. Virial, the transposed negative gradient with cell tensor times cell tensor, will be calculated, see eq 40 JCP 159, 054801 (2023).

atomic
[bool] If the variable is defined for each atom.

category
[int] The category of the output variable.

r_hessian
[bool] If hessian is requred

magnetic
[bool] If the derivatives of variable have magnetic parts.

property size

class deepmd.dpmodel.output_def.FittingOutputDef(var_defs: List[OutputVariableDef])

Defines the shapes and other properties of the fitting network outputs.

It is assume that the fitting network output variables for each local atom. This class defines all the outputs.

Parameters

var_defs
List of output variable definitions.

__getitem__(key: str) → OutputVariableDef

get_data() → Dict[str, OutputVariableDef]

keys()

class deepmd.dpmodel.output_def.**ModelOutputDef**(fit_defs: FittingOutputDef)

Defines the shapes and other properties of the model outputs.

The model reduce and differentiate fitting outputs if applicable. If a variable is named by foo, then the reduced variable is called foo_redu, the derivative w.r.t. coordinates is called foo_derv_r and the derivative w.r.t. cell is called foo_derv_c.

Parameters

fit_defs
Definition for the fitting net output

__getitem__(key: str) → OutputVariableDef

get_data(key: str) → Dict[str, OutputVariableDef]

keys()

keys_outp()

keys_redu()

keys_derv_r()

keys_hess_r()

keys_derv_c()

keys_derv_c_redu()

deepmd.dpmodel.output_def.**get_reduce_name**(name: str) → str

deepmd.dpmodel.output_def.**get_deriv_name**(name: str) → Tuple[str, str]

deepmd.dpmodel.output_def.**get_deriv_name_mag**(name: str) → Tuple[str, str]

deepmd.dpmodel.output_def.**get_hessian_name**(name: str) → str

deepmd.dpmodel.output_def.**apply_operation**(var_def: OutputVariableDef, op: OutputVariableOperation) → int

Apply an operation to the category of a variable definition.

Parameters

var_def
[*OutputVariableDef*] The variable definition.

op
[*OutputVariableOperation*] The operation to be applied.

Returns

int
The new category of the variable definition.

Raises

ValueError
If the operation has been applied to the variable definition, and exceed the maximum limitation.

deepmd.dpmodel.output_def.**check_operation_applied**(var_def: OutputVariableDef, op: OutputVariableOperation) → bool

Check if a operation has been applied to a variable definition.

Parameters

var_def
[*OutputVariableDef*] The variable definition.

op
[*OutputVariableOperation*] The operation to be checked.

Returns

bool
True if the operation has been applied, False otherwise.

deepmd.dpmodel.output_def.**do_reduce**(def_outp_data: Dict[str, OutputVariableDef]) → Dict[str, OutputVariableDef]

deepmd.dpmodel.output_def.**do_mask**(def_outp_data: Dict[str, OutputVariableDef]) → Dict[str, OutputVariableDef]

deepmd.dpmodel.output_def.**do_derivative**(def_outp_data: Dict[str, OutputVariableDef]) → Tuple[Dict[str, OutputVariableDef], Dict[str, OutputVariableDef]]

Package Contents

Classes

NativeOP	The unit operation of a native model.
DPModelCommon	
FittingOutputDef	Defines the shapes and other properties of the fitting network outputs.
ModelOutputDef	Defines the shapes and other properties of the model outputs.
OutputVariableDef	Defines the shape and other properties of the one output variable.

Functions

fitting_check_output(cls)	Check if the output of the Fitting is consistent with the definition.
get_deriv_name(→ Tuple[str, str])	
get_hessian_name(→ str)	
get_reduce_name(→ str)	
model_check_output(cls)	Check if the output of the Model is consistent with the definition.

Attributes

DEFAULT_PRECISION
PRECISION_DICT

deepmd.dpmodel.**DEFAULT_PRECISION** = 'float64'

deepmd.dpmodel.**PRECISION_DICT**

class deepmd.dpmodel.**NativeOP**

Bases: abc.ABC

The unit operation of a native model.

abstract call(*args, **kwargs)

Forward pass in NumPy implementation.

__call__(*args, **kwargs)

Forward pass in NumPy implementation.

class deepmd.dpmodel.**DPModelCommon**

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

class deepmd.dpmodel.**FittingOutputDef**(var_defs: List[OutputVariableDef])

Defines the shapes and other properties of the fitting network outputs.

It is assume that the fitting network output variables for each local atom. This class defines all the outputs.

Parameters

var_defs
List of output variable definitions.

__getitem__(key: str) → OutputVariableDef

get_data() → Dict[str, OutputVariableDef]

keys()

class deepmd.dpmodel.ModelOutputDef(fit_defs: FittingOutputDef)

Defines the shapes and other properties of the model outputs.

The model reduce and differentiate fitting outputs if applicable. If a variable is named by foo, then the reduced variable is called foo_redu, the derivative w.r.t. coordinates is called foo_derv_r and the derivative w.r.t. cell is called foo_derv_c.

Parameters

fit_defs
Definition for the fitting net output

__getitem__(key: str) → OutputVariableDef

get_data(key: str) → Dict[str, OutputVariableDef]

keys()

keys_outp()

keys_redu()

keys_derv_r()

keys_hess_r()

keys_derv_c()

keys_derv_c_redu()

class deepmd.dpmodel.OutputVariableDef(name: str, shape: List[int], reduciable: bool = False, r_differentiable: bool = False, c_differentiable: bool = False, atomic: bool = True, category: int = OutputVariableCategory.OUT.value, r_hessian: bool = False, magnetic: bool = False)

Defines the shape and other properties of the one output variable.

It is assume that the fitting network output variables for each local atom. This class defines one output variable, including its name, shape, reducibility and differentiability.

Parameters

name
Name of the output variable. Notice that the xxxx_redu, xxxx_derv_c, xxxx_derv_r are reserved names that should not be used to define variables.

shape
The shape of the variable. e.g. energy should be [1], dipole should be [3], polarizabilty should be [3,3].

reduciable
: If the variable is reduced.

r_differentiable
: If the variable is differentiated with respect to coordinates of atoms. Only reduciable variable are differentiable. Negative derivative w.r.t. coordinates will be calcualted. (e.g. force)

c_differentiable
: If the variable is differentiated with respect to the cell tensor (pbc case). Only reduciable variable are differentiable. Virial, the transposed negative gradient with cell tensor times cell tensor, will be calculated, see eq 40 JCP 159, 054801 (2023).

atomic
: [bool] If the variable is defined for each atom.

category
: [int] The category of the output variable.

r_hessian
: [bool] If hessian is requred

magnetic
: [bool] If the derivatives of variable have magnetic parts.

property size

deepmd.dpmodel.**fitting_check_output**(cls)

Check if the output of the Fitting is consistent with the definition.

Two methods are assumed to be provided by the Fitting: 1. Fitting.output_def that gives the output definition. 2. Fitting.__call__ defines the forward path of the fitting.

deepmd.dpmodel.**get_deriv_name**(name: str) → Tuple[str, str]

deepmd.dpmodel.**get_hessian_name**(name: str) → str

deepmd.dpmodel.**get_reduce_name**(name: str) → str

deepmd.dpmodel.**model_check_output**(cls)

Check if the output of the Model is consistent with the definition.

Two methods are assumed to be provided by the Model: 1. Model.output_def that gives the output definition. 2. Model.__call__ that defines the forward path of the model.

20.1.3 deepmd.entrypoints

Submodules

deepmd.entrypoints.convert_backend

Module Contents

Functions

convert_backend(→ None)	Convert a model file from one backend to another.

deepmd.entrypoints.convert_backend.convert_backend(*, INPUT: str, OUTPUT: str, **kwargs) → None

Convert a model file from one backend to another.

Parameters

INPUT
[str] The input model file.

INPUT
[str] The output model file.

deepmd.entrypoints.doc

Module that prints train input arguments docstrings.

Module Contents

Functions

doc_train_input(*[, out_type])	Print out trining input arguments to console.

deepmd.entrypoints.doc.doc_train_input(*, out_type: str = 'rst', **kwargs)

Print out trining input arguments to console.

deepmd.entrypoints.gui

DP-GUI entrypoint.

Module Contents

Functions

start_dpgui(*, port, bind_all, **kwargs)	Host DP-GUI server.

deepmd.entrypoints.gui.start_dpgui(*, port: int, bind_all: bool, **kwargs)

Host DP-GUI server.

Parameters

port
[int] The port to serve DP-GUI on.

bind_all
[bool] Serve on all public interfaces. This will expose your DP-GUI instance to the network on both IPv4 and IPv6 (where available).

**kwargs
additional arguments

Raises

ModuleNotFoundError
The dpgui package is not installed

deepmd.entrypoints.main

Common entrypoints.

Module Contents

Functions

main(args)	DeePMD-Kit entry point.

deepmd.entrypoints.main.main(args: argparse.Namespace)

DeePMD-Kit entry point.

Parameters

args
[List[str] or argparse.Namespace, optional] list of command line arguments, used to avoid calling from the subprocess, as it is quite slow to import tensorflow; if Namespace is given, it will be used directly

Raises

RuntimeError
if no command was input

deepmd.entrypoints.neighbor_stat

Module Contents

Functions

neighbor_stat(*, system, rcut, type_map[, mixed_type, ...])	Calculate neighbor statistics.

Attributes

log

deepmd.entrypoints.neighbor_stat.log

deepmd.entrypoints.neighbor_stat.**neighbor_stat**(*, system: str, rcut: float, type_map: List[str], mixed_type: bool = False, backend: str = 'tensorflow', **kwargs)

Calculate neighbor statistics.

Parameters

system
: [str] system to stat

rcut
: [float] cutoff radius

type_map
: [list[str]] type map

mixed_type
: [bool, optional, default=False] treat all types as a single type

backend
: [str, optional, default="tensorflow"] backend to use

**kwargs
: additional arguments

Examples

```
>>> neighbor_stat(
...     system=".",
...     rcut=6.0,
...     type_map=[
...         "C",
...         "H",
...         "O",
...         "N",
...         "P",
...         "S",
...         "Mg",
...         "Na",
...         "HW",
...         "OW",
...         "mNa",
...         "mCl",
...         "mC",
...         "mH",
...         "mMg",
...         "mN",
...         "mO",
```

(continues on next page)

(continued from previous page)

```
...         "mP",
...     ],
... )
min_nbor_dist: 0.6599510670195264
max_nbor_size: [23, 26, 19, 16, 2, 2, 1, 1, 72, 37, 5, 0, 31, 29, 1, 21, 20, 5]
```

deepmd.entrypoints.test

Test trained DeePMD model.

Module Contents

Functions

test(*, model, system, datafile, numb_test, rand_seed, ...)	Test model predictions.

deepmd.entrypoints.test.test(*, model: str, system: str, datafile: str, numb_test: int, rand_seed: int | None, shuffle_test: bool, detail_file: str, atomic: bool, head: str | None = None, **kwargs)

Test model predictions.

Parameters

model
: [str] path where model is stored

system
: [str] system directory

datafile
: [str] the path to the list of systems to test

numb_test
: [int] munber of tests to do. 0 means all data.

rand_seed
: [Optional[int]] seed for random generator

shuffle_test
: [bool] whether to shuffle tests

detail_file
: [Optional[str]] file where test details will be output

atomic
: [bool] whether per atom quantities should be computed

head
: [Optional[str], optional] (Supported backend: PyTorch) Task head to test if in multi-task mode.

**kwargs
: additional arguments

Raises

`RuntimeError`
if no valid system was found

20.1.4 deepmd.infer

Submodules

`deepmd.infer.deep_dipole`

Module Contents

Classes

DeepDipole	Deep dipole model.

`class deepmd.infer.deep_dipole.DeepDipole(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])`

Bases: *deepmd.infer.deep_tensor.DeepTensor*

Deep dipole model.

Parameters

model_file
[`Path`] The name of the frozen model file.

*args
[`list`] Positional arguments.

auto_batch_size
[bool or `int` or `AutoBatchSize`, default: `True`] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[`ase.neighborlist.NewPrimitiveNeighborList`, `optional`] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[`dict`] Keyword arguments.

`property output_tensor_name: str`

The name of the tensor.

deepmd.infer.deep_dos

Module Contents

Classes

DeepDOS	Deep density of states model.

class deepmd.infer.deep_dos.DeepDOS(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_eval.DeepEval*

Deep density of states model.

Parameters

model_file
[Path] The name of the frozen model file.

*args
[list] Positional arguments.

auto_batch_size
[bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

property output_def: *deepmd.dpmodel.output_def.ModelOutputDef*

Get the output definition of this model.

property numb_dos: int

Get the number of DOS.

eval(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: List[int] | numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: Dict[str, Any]) → Tuple[numpy.ndarray, Ellipsis]

Evaluate energy, force, and virial. If atomic is True, also return atomic energy and atomic virial.

Parameters

coords
[np.ndarray] The coordinates of the atoms, in shape (nframes, natoms, 3).

cells
[np.ndarray] The cell vectors of the system, in shape (nframes, 9). If the system is not periodic, set it to None.

atom_types
[List[int] or np.ndarray] The types of the atoms. If mixed_type is False, the shape is (natoms,); otherwise, the shape is (nframes, natoms).

atomic
[bool, optional] Whether to return atomic energy and atomic virial, by default False.

fparam
[np.ndarray, optional] The frame parameters, by default None.

aparam
[np.ndarray, optional] The atomic parameters, by default None.

mixed_type
[bool, optional] Whether the atom_types is mixed type, by default False.

**kwargs
[Dict[str, *Any*]] Keyword arguments.

Returns

energy
The energy of the system, in shape (nframes,).

force
The force of the system, in shape (nframes, natoms, 3).

virial
The virial of the system, in shape (nframes, 9).

atomic_energy
The atomic energy of the system, in shape (nframes, natoms). Only returned when atomic is True.

atomic_virial
The atomic virial of the system, in shape (nframes, natoms, 9). Only returned when atomic is True.

`get_numb_dos()` → int

deepmd.infer.deep_eval

Module Contents

Classes

DeepEvalBackend	Low-level Deep Evaluator interface.
DeepEval	High-level Deep Evaluator interface.

class deepmd.infer.deep_eval.DeepEvalBackend(model_file: str, output_def: deepmd.dpmodel.output_def.ModelOutputDef, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: abc.ABC

Low-level Deep Evaluator interface.

Backends should inherbit implement this interface. High-level interface will be built on top of this.

Parameters

model_file
[Path] The name of the frozen model file.

*args
[list] Positional arguments.

auto_batch_size
[bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

abstract property model_type: *DeepEval*

The the evaluator of the model type.

_OUTDEF_DP2BACKEND: ClassVar[dict]

abstract eval(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, **kwargs: Dict[str, Any]) → Dict[str, numpy.ndarray]

Evaluate the energy, force and virial by using this DP.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

atomic
Calculate the atomic energy and virial

fparam
The frame parameter. The array can be of size : - nframes x dim_fparam. - dim_fparam. Then all frames are assumed to be provided with the same fparam.

aparam
The atomic parameter The array can be of size : - nframes x natoms x dim_aparam. - natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. - dim_aparam. Then all frames and atoms are provided with the same aparam.

**kwargs
Other parameters

Returns

output_dict
[dict] The output of the evaluation. The keys are the names of the output variables, and the values are the corresponding output arrays.

abstract get_rcut() → float

Get the cutoff radius of this model.

abstract get_ntypes() → int

Get the number of atom types of this model.

abstract get_type_map() → List[str]

Get the type map (element name of the atom types) of this model.

abstract get_dim_fparam() → int

Get the number (dimension) of frame parameters of this DP.

abstract get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this DP.

abstract eval_descriptor(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: numpy.ndarray, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, efield: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: Dict[str, Any]) → numpy.ndarray

Evaluate descriptors by using this DP.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

fparam
The frame parameter. The array can be of size : - nframes x dim_fparam. - dim_fparam. Then all frames are assumed to be provided with the same fparam.

aparam
The atomic parameter The array can be of size : - nframes x natoms x dim_aparam. - natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. - dim_aparam. Then all frames and atoms are provided with the same aparam.

efield
The external field on atoms. The array should be of size nframes x natoms x 3

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

descriptor
Descriptors.

abstract eval_typeebd() → numpy.ndarray

Evaluate output of type embedding network by using this model.

Returns

np.ndarray
The output of type embedding network. The shape is [ntypes, o_size], where ntypes is the number of types, and o_size is the number of nodes in the output layer.

Raises

KeyError
If the model does not enable type embedding.

_check_mixed_types(atom_types: numpy.ndarray) → bool

Check if atom types of all frames are the same.

Traditional descriptors like se_e2_a requires all the frames to have the same atom types.

Parameters

atom_types
[np.ndarray] The atom types of all frames, in shape nframes * natoms.

abstract get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

abstract get_numb_dos() → int

Get the number of DOS.

get_has_efield()

Check if the model has efield.

get_has_spin()

Check if the model has spin atom types.

abstract get_ntypes_spin() → int

Get the number of spin atom types of this model. Only used in old implement.

class deepmd.infer.deep_eval.DeepEval(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: abc.ABC

High-level Deep Evaluator interface.

The specific DeepEval, such as DeepPot and DeepTensor, should inherit from this class. This class provides a high-level interface on the top of the low-level interface.

Parameters

model_file
[Path] The name of the frozen model file.

*args
[list] Positional arguments.

auto_batch_size
: [bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
: [ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
: [dict] Keyword arguments.

abstract property output_def: *deepmd.dpmodel.output_def.ModelOutputDef*

Returns the output variable definitions.

property has_efield: bool

Check if the model has efield.

property has_spin: bool

Check if the model has spin.

get_rcut() → float

Get the cutoff radius of this model.

get_ntypes() → int

Get the number of atom types of this model.

get_type_map() → List[str]

Get the type map (element name of the atom types) of this model.

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this DP.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this DP.

_get_natoms_and_nframes(coords: numpy.ndarray, atom_types: numpy.ndarray, mixed_type: bool = False) → Tuple[int, int]

_expande_atype(atype: numpy.ndarray, nframes: int, mixed_type: bool)

eval_descriptor(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: numpy.ndarray, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: Dict[str, Any]) → numpy.ndarray

Evaluate descriptors by using this DP.

Parameters

coords
: The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
: The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
: The atom types The list should contain natoms ints

fparam
: The frame parameter. The array can be of size : - nframes x dim_fparam. - dim_fparam. Then all frames are assumed to be provided with the same fparam.

aparam
: The atomic parameter The array can be of size : - nframes x natoms x dim_aparam. - natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. - dim_aparam. Then all frames and atoms are provided with the same aparam.

efield
: The external field on atoms. The array should be of size nframes x natoms x 3

mixed_type
: Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

descriptor
: Descriptors.

eval_typeebd() → numpy.ndarray

Evaluate output of type embedding network by using this model.

Returns

np.ndarray
: The output of type embedding network. The shape is [ntypes, o_size], where ntypes is the number of types, and o_size is the number of nodes in the output layer.

Raises

KeyError
: If the model does not enable type embedding.

See also:

deepmd.tf.utils.type_embed.TypeEmbedNet
: The type embedding network.

Examples

Get the output of type embedding network of graph.pb:

```
>>> from deepmd.infer import DeepPotential
>>> dp = DeepPotential("graph.pb")
>>> dp.eval_typeebd()
```

_standard_input(coords, cells, atom_types, fparam, aparam, mixed_type)

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

_get_sel_natoms(atype) → int

`get_ntypes_spin()` → int

Get the number of spin atom types of this model. Only used in old implement.

`deepmd.infer.deep_polar`

Module Contents

Classes

DeepPolar	Deep polar model.
DeepGlobalPolar	Deep Tensor Model.

`class deepmd.infer.deep_polar.DeepPolar(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])`

Bases: *deepmd.infer.deep_tensor.DeepTensor*

Deep polar model.

Parameters

model_file
[`Path`] The name of the frozen model file.

*args
[`list`] Positional arguments.

auto_batch_size
[bool or `int` or `AutoBatchSize`, default: `True`] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[`ase.neighborlist.NewPrimitiveNeighborList`, `optional`] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[`dict`] Keyword arguments.

`property output_tensor_name: str`

The name of the tensor.

`class deepmd.infer.deep_polar.DeepGlobalPolar(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])`

Bases: *deepmd.infer.deep_tensor.DeepTensor*

Deep Tensor Model.

Parameters

model_file
[`Path`] The name of the frozen model file.

*args
[list] Positional arguments.

auto_batch_size
[bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

property output_tensor_name: str

The name of the tensor.

eval(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: List[int] | numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: dict) → numpy.ndarray

Evaluate the model.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
[list[int] or np.ndarray] The atom types The list should contain natoms ints

atomic
If True (default), return the atomic tensor Otherwise return the global tensor

fparam
Not used in this model

aparam
Not used in this model

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

tensor
The returned tensor If atomic == False then of size nframes x output_dim else of size nframes x natoms x output_dim

deepmd.infer.deep_pot

Module Contents

Classes

DeepPot	Potential energy model.

class deepmd.infer.deep_pot.DeepPot(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_eval.DeepEval*

Potential energy model.

Parameters

model_file
: [Path] The name of the frozen model file.

*args
: [list] Positional arguments.

auto_batch_size
: [bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
: [ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
: [dict] Keyword arguments.

Examples

```
>>> from deepmd.infer import DeepPot
>>> import numpy as np
>>> dp = DeepPot("graph.pb")
>>> coord = np.array([[1, 0, 0], [0, 0, 1.5], [1, 0, 3]]).reshape([1, -1])
>>> cell = np.diag(10 * np.ones(3)).reshape([1, -1])
>>> atype = [1, 0, 1]
>>> e, f, v = dp.eval(coord, cell, atype)
```

where e, f and v are predicted energy, force and virial of the system, respectively.

property output_def: *deepmd.dpmodel.output_def.ModelOutputDef*
: Get the output definition of this model.

property output_def_mag: *deepmd.dpmodel.output_def.ModelOutputDef*
: Get the output definition of this model with magnetic parts.

eval(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: List[int] | numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: Dict[str, Any]) → Tuple[numpy.ndarray, Ellipsis]

Evaluate energy, force, and virial. If atomic is True, also return atomic energy and atomic virial.

Parameters

coords
[np.ndarray] The coordinates of the atoms, in shape (nframes, natoms, 3).

cells
[np.ndarray] The cell vectors of the system, in shape (nframes, 9). If the system is not periodic, set it to None.

atom_types
[List[int] or np.ndarray] The types of the atoms. If mixed_type is False, the shape is (natoms,); otherwise, the shape is (nframes, natoms).

atomic
[bool, optional] Whether to return atomic energy and atomic virial, by default False.

fparam
[np.ndarray, optional] The frame parameters, by default None.

aparam
[np.ndarray, optional] The atomic parameters, by default None.

mixed_type
[bool, optional] Whether the atom_types is mixed type, by default False.

**kwargs
[Dict[str, *Any*]] Keyword arguments.

Returns

energy
The energy of the system, in shape (nframes,).

force
The force of the system, in shape (nframes, natoms, 3).

virial
The virial of the system, in shape (nframes, 9).

atomic_energy
The atomic energy of the system, in shape (nframes, natoms). Only returned when atomic is True.

atomic_virial
The atomic virial of the system, in shape (nframes, natoms, 9). Only returned when atomic is True.

deepmd.infer.deep_tensor

Module Contents

Classes

DeepTensor	Deep Tensor Model.

class deepmd.infer.deep_tensor.DeepTensor(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_eval.DeepEval*

Deep Tensor Model.

Parameters

model_file
[Path] The name of the frozen model file.

*args
[list] Positional arguments.

auto_batch_size
[bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

abstract property output_tensor_name: str

The name of the tensor.

property output_def: *deepmd.dpmodel.output_def.ModelOutputDef*

Get the output definition of this model.

eval(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: List[int] | numpy.ndarray, atomic: bool = True, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: dict) → numpy.ndarray

Evaluate the model.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
: [list[int] or np.ndarray] The atom types The list should contain natoms ints

atomic
: If True (default), return the atomic tensor Otherwise return the global tensor

fparam
: Not used in this model

aparam
: Not used in this model

efield
: Not used in this model

mixed_type
: Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

tensor
: The returned tensor If atomic == False then of size nframes x output_dim else of size nframes x natoms x output_dim

eval_full(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: dict) → Tuple[numpy.ndarray, Ellipsis]

Evaluate the model with interface similar to the energy model. Will return global tensor, component-wise force and virial and optionally atomic tensor and atomic virial.

Parameters

coords
: The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
: The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
: The atom types The list should contain natoms ints

atomic
: Whether to calculate atomic tensor and virial

fparam
: Not used in this model

aparam
: Not used in this model

mixed_type
: Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

tensor
: The global tensor. shape: [nframes x nout]

force
The component-wise force (negative derivative) on each atom. shape: [nframes x nout x natoms x 3]

virial
The component-wise virial of the tensor. shape: [nframes x nout x 9]

atom_tensor
The atomic tensor. Only returned when atomic == True shape: [nframes x natoms x nout]

atom_virial
The atomic virial. Only returned when atomic == True shape: [nframes x nout x natoms x 9]

deepmd.infer.deep_wfc

Module Contents

Classes

DeepWFC	Deep WFC model.

class deepmd.infer.deep_wfc.DeepWFC(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_tensor.DeepTensor*

Deep WFC model.

Parameters

model_file
[**Path**] The name of the frozen model file.

*args
[**list**] Positional arguments.

auto_batch_size
[bool or **int** or **AutoBatchSize**, default: **True**] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[**ase.neighborlist.NewPrimitiveNeighborList**, **optional**] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[**dict**] Keyword arguments.

property output_tensor_name: str
The name of the tensor.

deepmd.infer.model_devi

Module Contents

Functions

calc_model_devi_f(...)	Calculate model deviation of force.
calc_model_devi_e(→ numpy.ndarray)	Calculate model deviation of total energy per atom.
calc_model_devi_v(→ Tuple[numpy.ndarray, ...)	Calculate model deviation of virial.
write_model_devi_out(devi, fname[, header, atomic])	Write output of model deviation.
_check_tmaps(tmaps[, ref_tmap])	Check whether type maps are identical.
calc_model_devi(coord, box, atype, models[, fname, ...])	Python interface to calculate model deviation.
make_model_devi(*, models, system, output, frequency)	Make model deviation calculation.

deepmd.infer.model_devi.**calc_model_devi_f**(fs: numpy.ndarray, real_f: numpy.ndarray | None = None, relative: float | None = None, atomic: Literal[False] = False) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

deepmd.infer.model_devi.**calc_model_devi_f**(fs: numpy.ndarray, real_f: numpy.ndarray | None = None, relative: float | None = None, *, atomic: Literal[True]) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray]

Calculate model deviation of force.

Parameters

fs
: [numpy.ndarray] size of n_models x n_frames x n_atoms x 3

real_f
: [numpy.ndarray or None] real force, size of n_frames x n_atoms x 3. If given, the RMS real error is calculated instead.

relative
: [float, default: None] If given, calculate the relative model deviation of force. The value is the level parameter for computing the relative model deviation of the force.

atomic
: [bool, default: False] Whether return deviation of force in all atoms

Returns

max_devi_f
: [numpy.ndarray] maximum deviation of force in all atoms

min_devi_f
: [numpy.ndarray] minimum deviation of force in all atoms

avg_devi_f
: [numpy.ndarray] average deviation of force in all atoms

fs_devi
: [numpy.ndarray] deviation of force in all atoms, returned if atomic=True

deepmd.infer.model_devi.calc_model_devi_e(es: numpy.ndarray, real_e: numpy.ndarray | None = None) → numpy.ndarray

Calculate model deviation of total energy per atom.

Here we don't use the atomic energy, as the decomposition of energy is arbitrary and not unique. There is no fitting target for atomic energy.

Parameters

es
[numpy.ndarray] size of `n_models x n_frames x 1

real_e
[numpy.ndarray] real energy, size of n_frames x 1. If given, the RMS real error is calculated instead.

Returns

max_devi_e
[numpy.ndarray] maximum deviation of energy

deepmd.infer.model_devi.calc_model_devi_v(vs: numpy.ndarray, real_v: numpy.ndarray | None = None, relative: float | None = None) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Calculate model deviation of virial.

Parameters

vs
[numpy.ndarray] size of n_models x n_frames x 9

real_v
[numpy.ndarray] real virial, size of n_frames x 9. If given, the RMS real error is calculated instead.

relative
[float, default: None] If given, calculate the relative model deviation of virial. The value is the level parameter for computing the relative model deviation of the virial.

Returns

max_devi_v
[numpy.ndarray] maximum deviation of virial in 9 elements

min_devi_v
[numpy.ndarray] minimum deviation of virial in 9 elements

avg_devi_v
[numpy.ndarray] average deviation of virial in 9 elements

deepmd.infer.model_devi.write_model_devi_out(devi: numpy.ndarray, fname: str, header: str = '', atomic: bool = False)

Write output of model deviation.

Parameters

devi
[numpy.ndarray] the first column is the steps index

fname
[str] the file name to dump

header
[str, default=""] the header to dump

atomic
[bool, default: False] whether atomic model deviation is printed

deepmd.infer.model_devi._check_tmaps(tmaps, ref_tmap=None)

Check whether type maps are identical.

deepmd.infer.model_devi.calc_model_devi(coord, box, atype, models, fname=None, frequency=1, mixed_type=False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, real_data: dict | None = None, atomic: bool = False, relative: float | None = None, relative_v: float | None = None)

Python interface to calculate model deviation.

Parameters

coord
[numpy.ndarray, n_frames x n_atoms x 3] Coordinates of system to calculate

box
[numpy.ndarray or None, n_frames x 3 x 3] Box to specify periodic boundary condition. If None, no pbc will be used

atype
[numpy.ndarray, n_atoms x 1] Atom types

models
[list of DeepPot models] Models used to evaluate deviation

fname
[str or None] File to dump results, default None

frequency
[int] Steps between frames (if the system is given by molecular dynamics engine), default 1

mixed_type
[bool] Whether the input atype is in mixed_type format or not

fparam
[numpy.ndarray] frame specific parameters

aparam
[numpy.ndarray] atomic specific parameters

real_data
[dict, optional] real data to calculate RMS real error

atomic
[bool, default: False] If True, calculate the force model deviation of each atom.

relative
[float, default: None] If given, calculate the relative model deviation of force. The value is the level parameter for computing the relative model deviation of the force.

relative_v
[float, default: None] If given, calculate the relative model deviation of virial. The value is the level parameter for computing the relative model deviation of the virial.

Returns

model_devi
: [numpy.ndarray, n_frames x 8] Model deviation results. The first column is index of steps, the other 7 columns are max_devi_v, min_devi_v, avg_devi_v, max_devi_f, min_devi_f, avg_devi_f, devi_e.

Examples

```
>>> from deepmd.tf.infer import calc_model_devi
>>> from deepmd.tf.infer import DeepPot as DP
>>> import numpy as np
>>> coord = np.array([[1, 0, 0], [0, 0, 1.5], [1, 0, 3]]).reshape([1, -1])
>>> cell = np.diag(10 * np.ones(3)).reshape([1, -1])
>>> atype = [1, 0, 1]
>>> graphs = [DP("graph.000.pb"), DP("graph.001.pb")]
>>> model_devi = calc_model_devi(coord, cell, atype, graphs)
```

deepmd.infer.model_devi.**make_model_devi**(*, models: list, system: str, output: str, frequency: int, real_error: bool = False, atomic: bool = False, relative: float | None = None, relative_v: float | None = None, **kwargs)

Make model deviation calculation.

Parameters

models
: [list] A list of paths of models to use for making model deviation

system
: [str] The path of system to make model deviation calculation

output
: [str] The output file for model deviation results

frequency
: [int] The number of steps that elapse between writing coordinates in a trajectory by a MD engine (such as Gromacs / Lammps). This paramter is used to determine the index in the output file.

real_error
: [bool, default: False] If True, calculate the RMS real error instead of model deviation.

atomic
: [bool, default: False] If True, calculate the force model deviation of each atom.

relative
: [float, default: None] If given, calculate the relative model deviation of force. The value is the level parameter for computing the relative model deviation of the force.

relative_v
: [float, default: None] If given, calculate the relative model deviation of virial. The value is the level parameter for computing the relative model deviation of the virial.

**kwargs
: Arbitrary keyword arguments.

Package Contents

Classes

DeepEval	High-level Deep Evaluator interface.
DeepPot	Potential energy model.

Functions

calc_model_devi(coord, box, atype, models[, fname, ...])	Python interface to calculate model deviation.
DeepPotential(→ deep_eval.DeepEval)	Factory function that forwards to DeepEval (for compatbility).

class deepmd.infer.DeepEval(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: abc.ABC

High-level Deep Evaluator interface.

The specific DeepEval, such as DeepPot and DeepTensor, should inherit from this class. This class provides a high-level interface on the top of the low-level interface.

Parameters

model_file
[Path] The name of the frozen model file.

*args
[list] Positional arguments.

auto_batch_size
[bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

abstract property output_def: *deepmd.dpmodel.output_def.ModelOutputDef*

Returns the output variable definitions.

property has_efield: bool

Check if the model has efield.

property has_spin: bool

Check if the model has spin.

get_rcut() → float

Get the cutoff radius of this model.

get_ntypes() → int

Get the number of atom types of this model.

get_type_map() → List[str]

Get the type map (element name of the atom types) of this model.

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this DP.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this DP.

_get_natoms_and_nframes(coords: numpy.ndarray, atom_types: numpy.ndarray, mixed_type: bool = False) → Tuple[int, int]

_expande_atype(atype: numpy.ndarray, nframes: int, mixed_type: bool)

eval_descriptor(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: numpy.ndarray, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: Dict[str, Any]) → numpy.ndarray

Evaluate descriptors by using this DP.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

fparam
The frame parameter. The array can be of size : - nframes x dim_fparam. - dim_fparam. Then all frames are assumed to be provided with the same fparam.

aparam
The atomic parameter The array can be of size : - nframes x natoms x dim_aparam. - natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. - dim_aparam. Then all frames and atoms are provided with the same aparam.

efield
The external field on atoms. The array should be of size nframes x natoms x 3

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

descriptor
Descriptors.

eval_typeebd() → numpy.ndarray

Evaluate output of type embedding network by using this model.

Returns

np.ndarray
The output of type embedding network. The shape is [ntypes, o_size], where ntypes is the number of types, and o_size is the number of nodes in the output layer.

Raises

KeyError
If the model does not enable type embedding.

See also:

deepmd.tf.utils.type_embed.TypeEmbedNet
The type embedding network.

Examples

Get the output of type embedding network of graph.pb:

```
>>> from deepmd.infer import DeepPotential
>>> dp = DeepPotential("graph.pb")
>>> dp.eval_typeebd()
```

_standard_input(coords, cells, atom_types, fparam, aparam, mixed_type)

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

_get_sel_natoms(atype) → int

get_ntypes_spin() → int

Get the number of spin atom types of this model. Only used in old implement.

class deepmd.infer.DeepPot(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_eval.DeepEval*

Potential energy model.

Parameters

model_file
[**Path**] The name of the frozen model file.

*args
[**list**] Positional arguments.

auto_batch_size
[bool or **int** or **AutoBatchSize**, default: **True**] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
: [ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
: [dict] Keyword arguments.

Examples

```
>>> from deepmd.infer import DeepPot
>>> import numpy as np
>>> dp = DeepPot("graph.pb")
>>> coord = np.array([[1, 0, 0], [0, 0, 1.5], [1, 0, 3]]).reshape([1, -1])
>>> cell = np.diag(10 * np.ones(3)).reshape([1, -1])
>>> atype = [1, 0, 1]
>>> e, f, v = dp.eval(coord, cell, atype)
```

where e, f and v are predicted energy, force and virial of the system, respectively.

property output_def: *deepmd.dpmodel.output_def.ModelOutputDef*

Get the output definition of this model.

property output_def_mag: *deepmd.dpmodel.output_def.ModelOutputDef*

Get the output definition of this model with magnetic parts.

eval(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: List[int] | numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: Dict[str, Any]) → Tuple[numpy.ndarray, Ellipsis]

Evaluate energy, force, and virial. If atomic is True, also return atomic energy and atomic virial.

Parameters

coords
: [np.ndarray] The coordinates of the atoms, in shape (nframes, natoms, 3).

cells
: [np.ndarray] The cell vectors of the system, in shape (nframes, 9). If the system is not periodic, set it to None.

atom_types
: [List[int] or np.ndarray] The types of the atoms. If mixed_type is False, the shape is (natoms,); otherwise, the shape is (nframes, natoms).

atomic
: [bool, optional] Whether to return atomic energy and atomic virial, by default False.

fparam
: [np.ndarray, optional] The frame parameters, by default None.

aparam
: [np.ndarray, optional] The atomic parameters, by default None.

mixed_type
: [bool, optional] Whether the atom_types is mixed type, by default False.

**kwargs
: [Dict[str, *Any*]] Keyword arguments.

Returns

energy
The energy of the system, in shape (nframes,).

force
The force of the system, in shape (nframes, natoms, 3).

virial
The virial of the system, in shape (nframes, 9).

atomic_energy
The atomic energy of the system, in shape (nframes, natoms). Only returned when atomic is True.

atomic_virial
The atomic virial of the system, in shape (nframes, natoms, 9). Only returned when atomic is True.

`deepmd.infer.calc_model_devi`(coord, box, atype, models, fname=None, frequency=1, mixed_type=False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, real_data: dict | None = None, atomic: bool = False, relative: float | None = None, relative_v: float | None = None)

Python interface to calculate model deviation.

Parameters

coord
[`numpy.ndarray`, n_frames x n_atoms x 3] Coordinates of system to calculate

box
[`numpy.ndarray` or `None`, n_frames x 3 x 3] Box to specify periodic boundary condition. If None, no pbc will be used

atype
[`numpy.ndarray`, n_atoms x 1] Atom types

models
[`list of DeepPot models`] Models used to evaluate deviation

fname
[`str` or `None`] File to dump results, default None

frequency
[`int`] Steps between frames (if the system is given by molecular dynamics engine), default 1

mixed_type
[bool] Whether the input atype is in mixed_type format or not

fparam
[`numpy.ndarray`] frame specific parameters

aparam
[`numpy.ndarray`] atomic specific parameters

real_data
[`dict`, `optional`] real data to calculate RMS real error

atomic
[bool, default: `False`] If True, calculate the force model deviation of each atom.

relative
: [float, default: None] If given, calculate the relative model deviation of force. The value is the level parameter for computing the relative model deviation of the force.

relative_v
: [float, default: None] If given, calculate the relative model deviation of virial. The value is the level parameter for computing the relative model deviation of the virial.

Returns

model_devi
: [numpy.ndarray, n_frames x 8] Model deviation results. The first column is index of steps, the other 7 columns are max_devi_v, min_devi_v, avg_devi_v, max_devi_f, min_devi_f, avg_devi_f, devi_e.

Examples

```
>>> from deepmd.tf.infer import calc_model_devi
>>> from deepmd.tf.infer import DeepPot as DP
>>> import numpy as np
>>> coord = np.array([[1, 0, 0], [0, 0, 1.5], [1, 0, 3]]).reshape([1, -1])
>>> cell = np.diag(10 * np.ones(3)).reshape([1, -1])
>>> atype = [1, 0, 1]
>>> graphs = [DP("graph.000.pb"), DP("graph.001.pb")]
>>> model_devi = calc_model_devi(coord, cell, atype, graphs)
```

deepmd.infer.DeepPotential(*args, **kwargs) → deep_eval.DeepEval

Factory function that forwards to DeepEval (for compatbility).

Parameters

*args
: positional arguments

**kwargs
: keyword arguments

Returns

DeepEval
: potentials

20.1.5 deepmd.loggers

Module taking care of logging duties.

Submodules

deepmd.loggers.loggers

Logger initialization for package.

Module Contents

Functions

set_log_handles(level[, log_path, mpi_log])	Set desired level for package loggers and add file handlers.

deepmd.loggers.loggers.**set_log_handles**(level: int, log_path: pathlib.Path | None = None, mpi_log: str | None = None)

Set desired level for package loggers and add file handlers.

Parameters

level
[int] logging level

log_path
[Optional[str]] path to log file, if None logs will be send only to console. If the parent directory does not exist it will be automatically created, by default None

mpi_log
[Optional[str], optional] mpi log type. Has three options. master will output logs to file and console only from rank==0. collect will write messages from all ranks to one file opened under rank==0 and to console. workers will open one log file for each worker designated by its rank, console behaviour is the same as for collect. If this argument is specified, package 'mpi4py' must be already installed. by default None

Raises

RuntimeError
If the argument mpi_log is specified, package mpi4py is not installed.

Notes

Logging levels:

	our notation	python logging	tensorflow cpp	OpenMP
debug	10	10	0	1/on/true/yes
info	20	20	1	0/off/false/no
warning	30	30	2	0/off/false/no
error	40	40	3	0/off/false/no

References

https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U https://stackoverflow.com/questions/35869137/avoid-tensorflow-print-on-standard-error https://stackoverflow.com/questions/56085015/suppress-openmp-debug-messages-when-running-tensorflow-on-cpu

`deepmd.loggers.training`

Module Contents

Functions

format_training_message(batch, wall_time)	Format a training message.
format_training_message_per_task(batch, task_name, ...)	

`deepmd.loggers.training.format_training_message(batch: int, wall_time: float)`

Format a training message.

`deepmd.loggers.training.format_training_message_per_task(batch: int, task_name: str, rmse: Dict[str, float], learning_rate: float | None)`

Package Contents

Functions

set_log_handles(level[, log_path, mpi_log])	Set desired level for package loggers and add file handlers.

`deepmd.loggers.set_log_handles(level: int, log_path: pathlib.Path | None = None, mpi_log: str | None = None)`

Set desired level for package loggers and add file handlers.

Parameters

level
: [`int`] logging level

log_path
: [`Optional[str]`] path to log file, if None logs will be send only to console. If the parent directory does not exist it will be automatically created, by default None

mpi_log
: [`Optional[str]`, `optional`] mpi log type. Has three options. master will output logs to file and console only from rank==0. collect will write messages from all ranks to one file opened under rank==0 and to console. workers will open one log file for each worker designated by its rank, console behaviour is the same as for collect. If this argument is specified, package 'mpi4py' must be already installed. by default None

Raises

`RuntimeError`
If the argument mpi_log is specified, package mpi4py is not installed.

Notes

Logging levels:

	our notation	python logging	tensorflow cpp	OpenMP
debug	10	10	0	1/on/true/yes
info	20	20	1	0/off/false/no
warning	30	30	2	0/off/false/no
error	40	40	3	0/off/false/no

References

https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U https://stackoverflow.com/questions/35869137/avoid-tensorflow-print-on-standard-error https://stackoverflow.com/questions/56085015/suppress-openmp-debug-messages-when-running-tensorflow-on-cpu

20.1.6 `deepmd.pt`

Subpackages

`deepmd.pt.entrypoints`

Submodules

`deepmd.pt.entrypoints.main`

Module Contents

Classes

SummaryPrinter	Summary printer for PyTorch.

Functions

get_trainer(config[, init_model, restart_model, ...])	
train(FLAGS)	
freeze(FLAGS)	
main([args])	

Attributes

log

deepmd.pt.entrypoints.main.**log**

deepmd.pt.entrypoints.main.**get_trainer**(config, init_model=None, restart_model=None, finetune_model=None, model_branch='', force_load=False, init_frz_model=None, shared_links=None)

class deepmd.pt.entrypoints.main.**SummaryPrinter**

Bases: *deepmd.utils.summary.SummaryPrinter*

Summary printer for PyTorch.

is_built_with_cuda() → bool

Check if the backend is built with CUDA.

is_built_with_rocm() → bool

Check if the backend is built with ROCm.

get_compute_device() → str

Get Compute device.

get_ngpus() → int

Get the number of GPUs.

get_backend_info() → dict

Get backend information.

deepmd.pt.entrypoints.main.**train**(FLAGS)

deepmd.pt.entrypoints.main.**freeze**(FLAGS)

deepmd.pt.entrypoints.main.**main**(args: List[str] | argparse.Namespace | None = None)

deepmd.pt.infer

Submodules

deepmd.pt.infer.deep_eval

Module Contents

Classes

DeepEval	PyTorch backend implementaion of DeepEval.

Functions

eval_model(model, coords, cells, atom_types[, spins, ...])

class deepmd.pt.infer.deep_eval.DeepEval(model_file: str, output_def: deepmd.dpmodel.output_def.ModelOutputDef, *args: List[Any], auto_batch_size: bool | int | deepmd.pt.utils.auto_batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, head: str | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_eval.DeepEvalBackend*

PyTorch backend implementaion of DeepEval.

Parameters

model_file
[Path] The name of the frozen model file.

output_def
[ModelOutputDef] The output definition of the model.

*args
[list] Positional arguments.

auto_batch_size
[bool or int or AutomaticBatchSize, default: False] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

property model_type: *deepmd.infer.deep_eval.DeepEval*

The the evaluator of the model type.

get_rcut() → float

Get the cutoff radius of this model.

get_ntypes() → int

Get the number of atom types of this model.

get_type_map() → List[str]

Get the type map (element name of the atom types) of this model.

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this DP.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this DP.

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

get_numb_dos() → int

Get the number of DOS.

get_has_efield()

Check if the model has efield.

get_ntypes_spin()

Get the number of spin atom types of this model. Only used in old implement.

get_has_spin()

Check if the model has spin atom types.

eval(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, **kwargs: Dict[str, Any]) → Dict[str, numpy.ndarray]

Evaluate the energy, force and virial by using this DP.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

atomic
Calculate the atomic energy and virial

fparam
The frame parameter. The array can be of size : - nframes x dim_fparam. - dim_fparam. Then all frames are assumed to be provided with the same fparam.

aparam
The atomic parameter The array can be of size : - nframes x natoms x dim_aparam. - natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. - dim_aparam. Then all frames and atoms are provided with the same aparam.

**kwargs
Other parameters

Returns

output_dict
[dict] The output of the evaluation. The keys are the names of the output variables, and the values are the corresponding output arrays.

_get_request_defs(atomic: bool) → List[deepmd.dpmodel.output_def.OutputVariableDef]

Get the requested output definitions.

When atomic is True, all output_def are requested. When atomic is False, only energy (tensor), force, and virial are requested.

Parameters

atomic
[bool] Whether to request the atomic output.

Returns

list[**OutputVariableDef**]
The requested output definitions.

_eval_func(inner_func: Callable, numb_test: int, natoms: int) → Callable

Wrapper method with auto batch size.

Parameters

inner_func
[Callable] the method to be wrapped

numb_test
[int] number of tests

natoms
[int] number of atoms

Returns

Callable
the wrapper

_get_natoms_and_nframes(coords: numpy.ndarray, atom_types: numpy.ndarray, mixed_type: bool = False) → Tuple[int, int]

_eval_model(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: numpy.ndarray, fparam: numpy.ndarray | None, aparam: numpy.ndarray | None, request_defs: List[deepmd.dpmodel.output_def.OutputVariableDef])

_eval_model_spin(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: numpy.ndarray, spins: numpy.ndarray, fparam: numpy.ndarray | None, aparam: numpy.ndarray | None, request_defs: List[deepmd.dpmodel.output_def.OutputVariableDef])

_get_output_shape(odef, nframes, natoms)

deepmd.pt.infer.deep_eval.eval_model(model, coords: numpy.ndarray | torch.Tensor, cells: numpy.ndarray | torch.Tensor | None, atom_types: numpy.ndarray | torch.Tensor | List[int], spins: numpy.ndarray | torch.Tensor | None = None, atomic: bool = False, infer_batch_size: int = 2, denoise: bool = False)

deepmd.pt.infer.inference

Module Contents

Classes

Tester	

Attributes

log	

deepmd.pt.infer.inference.log

class deepmd.pt.infer.inference.Tester(model_ckpt, head=None)

deepmd.pt.loss

Submodules

deepmd.pt.loss.denoise

Module Contents

Classes

DenoiseLoss	Base class for all neural network modules.

class deepmd.pt.loss.denoise.DenoiseLoss(ntypes, masked_token_loss=1.0, masked_coord_loss=1.0, norm_loss=0.01, use_l1=True, beta=1.0, mask_loss_coord=True, mask_loss_token=True, **kwargs)

Bases: *deepmd.pt.loss.loss.TaskLoss*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(model_pred, label, natoms, learning_rate, mae=False)

Return loss on coord and type denoise.

Returns

- loss: **Loss to** minimize.

deepmd.pt.loss.dos

Module Contents

Classes

DOSLoss	Base class for all neural network modules.

class deepmd.pt.loss.dos.DOSLoss(starter_learning_rate: float, numb_dos: int, start_pref_dos: float = 1.0, limit_pref_dos: float = 1.0, start_pref_cdf: float = 1000, limit_pref_cdf: float = 1.0, start_pref_ados: float = 0.0, limit_pref_ados: float = 0.0, start_pref_acdf: float = 0.0, limit_pref_acdf: float = 0.0, inference=False, **kwargs)

Bases: *deepmd.pt.loss.loss.TaskLoss*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

property label_requirement: List[*deepmd.utils.data.DataRequirementItem*]

Return data label requirements needed for this loss calculation.

forward(input_dict, model, label, natoms, learning_rate=0.0, mae=False)

Return loss on local and global tensors.

Parameters

input_dict
[dict[str, torch.Tensor]] Model inputs.

model
[torch.nn.Module] Model to be used to output the predictions.

label
[dict[str, torch.Tensor]] Labels.

natoms
[int] The local atom number.

Returns

model_pred: dict[str, torch.Tensor]
Model predictions.

loss: **torch.Tensor**
Loss for model to minimize.

more_loss: **dict**[**str**, **torch.Tensor**]
Other losses for display.

deepmd.pt.loss.ener

Module Contents

Classes

EnergyStdLoss	Base class for all neural network modules.

class deepmd.pt.loss.ener.EnergyStdLoss(starter_learning_rate=1.0, start_pref_e=0.0, limit_pref_e=0.0, start_pref_f=0.0, limit_pref_f=0.0, start_pref_v=0.0, limit_pref_v=0.0, start_pref_ae: float = 0.0, limit_pref_ae: float = 0.0, start_pref_pf: float = 0.0, limit_pref_pf: float = 0.0, use_l1_all: bool = False, inference=False, **kwargs)

Bases: *deepmd.pt.loss.loss.TaskLoss*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

property label_requirement: List[*deepmd.utils.data.DataRequirementItem*]

Return data label requirements needed for this loss calculation.

forward(input_dict, model, label, natoms, learning_rate, mae=False)

Return loss on energy and force.

Parameters

input_dict
[dict[str, torch.Tensor]] Model inputs.

model
[torch.nn.Module] Model to be used to output the predictions.

label
[dict[str, torch.Tensor]] Labels.

natoms
[int] The local atom number.

Returns

model_pred: dict[str, torch.Tensor]
Model predictions.

loss: torch.Tensor
Loss for model to minimize.

more_loss: dict[str, torch.Tensor]
Other losses for display.

deepmd.pt.loss.ener_spin

Module Contents

Classes

EnergySpinLoss	Base class for all neural network modules.

class deepmd.pt.loss.ener_spin.**EnergySpinLoss**(starter_learning_rate=1.0, start_pref_e=0.0, limit_pref_e=0.0, start_pref_fr=0.0, limit_pref_fr=0.0, start_pref_fm=0.0, limit_pref_fm=0.0, start_pref_v=0.0, limit_pref_v=0.0, start_pref_ae: float = 0.0, limit_pref_ae: float = 0.0, start_pref_pf: float = 0.0, limit_pref_pf: float = 0.0, use_l1_all: bool = False, inference=False, **kwargs)

Bases: *deepmd.pt.loss.loss.TaskLoss*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

property label_requirement: List[*deepmd.utils.data.DataRequirementItem*]

Return data label requirements needed for this loss calculation.

forward(input_dict, model, label, natoms, learning_rate, mae=False)

Return energy loss with magnetic labels.

Parameters

input_dict
[dict[str, torch.Tensor]] Model inputs.

model
[torch.nn.Module] Model to be used to output the predictions.

label
[dict[str, torch.Tensor]] Labels.

natoms
[int] The local atom number.

Returns

model_pred: dict[str, **torch.Tensor**]
Model predictions.

loss: **torch.Tensor**
Loss for model to minimize.

more_loss: dict[str, **torch.Tensor**]
Other losses for display.

deepmd.pt.loss.loss

Module Contents

Classes

TaskLoss	Base class for all neural network modules.

class deepmd.pt.loss.loss.**TaskLoss**(**kwargs)

Bases: torch.nn.Module, abc.ABC

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

abstract property label_requirement: List[*deepmd.utils.data.DataRequirementItem***]**

Return data label requirements needed for this loss calculation.

abstract forward(input_dict, model, label, natoms, learning_rate)

Return loss .

static display_if_exist(loss: torch.Tensor, find_property: float) → torch.Tensor

Display NaN if labeled property is not found.

Parameters

loss

[torch.Tensor] the loss tensor

find_property
[float] whether the property is found

deepmd.pt.loss.tensor

Module Contents

Classes

TensorLoss	Base class for all neural network modules.

class deepmd.pt.loss.tensor.TensorLoss(tensor_name: str, tensor_size: int, label_name: str, pref_atomic: float = 0.0, pref: float = 0.0, inference=False, **kwargs)

Bases: *deepmd.pt.loss.loss.TaskLoss*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

property label_requirement: List[*deepmd.utils.data.DataRequirementItem*]

Return data label requirements needed for this loss calculation.

forward(input_dict, model, label, natoms, learning_rate=0.0, mae=False)

Return loss on local and global tensors.

Parameters

input_dict
[dict[str, torch.Tensor]] Model inputs.

model
[torch.nn.Module] Model to be used to output the predictions.

label
[dict[str, torch.Tensor]] Labels.

natoms
[int] The local atom number.

Returns

model_pred: dict[str, torch.Tensor]
Model predictions.

loss: torch.Tensor
Loss for model to minimize.

more_loss: dict[str, torch.Tensor]
Other losses for display.

Package Contents

Classes

DenoiseLoss	Base class for all neural network modules.
DOSLoss	Base class for all neural network modules.
EnergyStdLoss	Base class for all neural network modules.
EnergySpinLoss	Base class for all neural network modules.
TaskLoss	Base class for all neural network modules.
TensorLoss	Base class for all neural network modules.

class deepmd.pt.loss.DenoiseLoss(ntypes, masked_token_loss=1.0, masked_coord_loss=1.0, norm_loss=0.01, use_l1=True, beta=1.0, mask_loss_coord=True, mask_loss_token=True, **kwargs)

Bases: *deepmd.pt.loss.loss.TaskLoss*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)
```

(continues on next page)

(continued from previous page)

```
    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(model_pred, label, natoms, learning_rate, mae=False)

Return loss on coord and type denoise.

Returns

- loss: **Loss to** minimize.

class deepmd.pt.loss.DOSLoss(starter_learning_rate: float, numb_dos: int, start_pref_dos: float = 1.0, limit_pref_dos: float = 1.0, start_pref_cdf: float = 1000, limit_pref_cdf: float = 1.0, start_pref_ados: float = 0.0, limit_pref_ados: float = 0.0, start_pref_acdf: float = 0.0, limit_pref_acdf: float = 0.0, inference=False, **kwargs)

Bases: *deepmd.pt.loss.loss.TaskLoss*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

property label_requirement: List[*deepmd.utils.data.DataRequirementItem*]
Return data label requirements needed for this loss calculation.

forward(input_dict, model, label, natoms, learning_rate=0.0, mae=False)
Return loss on local and global tensors.

Parameters

input_dict
[dict[str, torch.Tensor]] Model inputs.

model
[torch.nn.Module] Model to be used to output the predictions.

label
[dict[str, torch.Tensor]] Labels.

natoms
[int] The local atom number.

Returns

model_pred: dict[str, **torch.Tensor**]
Model predictions.

loss: **torch.Tensor**
Loss for model to minimize.

more_loss: dict[str, **torch.Tensor**]
Other losses for display.

class deepmd.pt.loss.EnergyStdLoss(starter_learning_rate=1.0, start_pref_e=0.0, limit_pref_e=0.0, start_pref_f=0.0, limit_pref_f=0.0, start_pref_v=0.0, limit_pref_v=0.0, start_pref_ae: float = 0.0, limit_pref_ae: float = 0.0, start_pref_pf: float = 0.0, limit_pref_pf: float = 0.0, use_l1_all: bool = False, inference=False, **kwargs)

Bases: *deepmd.pt.loss.loss.TaskLoss*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

property label_requirement: List[*deepmd.utils.data.DataRequirementItem*]

Return data label requirements needed for this loss calculation.

forward(input_dict, model, label, natoms, learning_rate, mae=False)

Return loss on energy and force.

Parameters

input_dict
[dict[str, torch.Tensor]] Model inputs.

model
[torch.nn.Module] Model to be used to output the predictions.

label
[dict[str, torch.Tensor]] Labels.

natoms
[int] The local atom number.

Returns

model_pred: **dict**[**str**, **torch.Tensor**]
Model predictions.

loss: **torch.Tensor**
Loss for model to minimize.

more_loss: **dict**[**str**, **torch.Tensor**]
Other losses for display.

class deepmd.pt.loss.EnergySpinLoss(starter_learning_rate=1.0, start_pref_e=0.0, limit_pref_e=0.0, start_pref_fr=0.0, limit_pref_fr=0.0, start_pref_fm=0.0, limit_pref_fm=0.0, start_pref_v=0.0, limit_pref_v=0.0, start_pref_ae: float = 0.0, limit_pref_ae: float = 0.0, start_pref_pf: float = 0.0, limit_pref_pf: float = 0.0, use_l1_all: bool = False, inference=False, **kwargs)

Bases: *deepmd.pt.loss.loss.TaskLoss*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

property label_requirement: List[*deepmd.utils.data.DataRequirementItem*]

Return data label requirements needed for this loss calculation.

forward(input_dict, model, label, natoms, learning_rate, mae=False)

Return energy loss with magnetic labels.

Parameters

input_dict
[dict[str, torch.Tensor]] Model inputs.

model
[torch.nn.Module] Model to be used to output the predictions.

label
[dict[str, torch.Tensor]] Labels.

natoms
[int] The local atom number.

Returns

model_pred: dict[str, **torch.Tensor**]
Model predictions.

loss: **torch.Tensor**
Loss for model to minimize.

more_loss: dict[str, **torch.Tensor**]
Other losses for display.

class deepmd.pt.loss.TaskLoss(**kwargs)

Bases: torch.nn.Module, abc.ABC

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
: **training** (*bool*) – Boolean represents whether this module is in training or evaluation mode.

abstract property label_requirement: List[*deepmd.utils.data.DataRequirementItem***]**

Return data label requirements needed for this loss calculation.

abstract forward(input_dict, model, label, natoms, learning_rate)

Return loss .

static display_if_exist(loss: torch.Tensor, find_property: float) → torch.Tensor

Display NaN if labeled property is not found.

Parameters
: loss
  : [`torch.Tensor`] the loss tensor

  find_property
  : [`float`] whether the property is found

class deepmd.pt.loss.TensorLoss(tensor_name: str, tensor_size: int, label_name: str, pref_atomic: float = 0.0, pref: float = 0.0, inference=False, **kwargs)

Bases: *deepmd.pt.loss.loss.TaskLoss*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

property label_requirement: List[*deepmd.utils.data.DataRequirementItem*]

Return data label requirements needed for this loss calculation.

forward(input_dict, model, label, natoms, learning_rate=0.0, mae=False)

Return loss on local and global tensors.

Parameters

input_dict
[dict[str, torch.Tensor]] Model inputs.

model
[torch.nn.Module] Model to be used to output the predictions.

label
[dict[str, torch.Tensor]] Labels.

natoms
[int] The local atom number.

Returns

model_pred: dict[str, **torch.Tensor**]
Model predictions.

loss: **torch.Tensor**
Loss for model to minimize.

more_loss: dict[str, **torch.Tensor**]
Other losses for display.

deepmd.pt.model

Subpackages

deepmd.pt.model.atomic_model

The atomic model provides the prediction of some property on each atom. All the atomic models are not supposed to be directly accessed by users, but it provides a convenient interface for the implementation of models.

Taking the energy models for example, the developeres only needs to implement the atomic energy prediction via an atomic model, and the model can be automatically made by the deepmd.dpmodel.make_model method. The DPModel is made by ` DPModel = make_model(DPAtomicModel) `

Submodules

deepmd.pt.model.atomic_model.base_atomic_model

Module Contents

Classes

BaseAtomicModel	The base of atomic model.

Attributes

log
dtype
device
*BaseAtomicModel_*

deepmd.pt.model.atomic_model.base_atomic_model.**log**

deepmd.pt.model.atomic_model.base_atomic_model.**dtype**

deepmd.pt.model.atomic_model.base_atomic_model.**device**

deepmd.pt.model.atomic_model.base_atomic_model.**BaseAtomicModel_**

class deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel(type_map: List[str], atom_exclude_types: List[int] = [], pair_exclude_types: List[Tuple[int, int]] = [], rcond: float | None = None, preset_out_bias: Dict[str, torch.Tensor] | None = None)

Bases: torch.nn.Module, *BaseAtomicModel_*

The base of atomic model.

Parameters

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

atom_exclude_types
: Exclude the atomic contribution of the given types

pair_exclude_types
: Exclude the pair of atoms of the given types from computing the output of the atomic model. Implemented by removing the pairs from the nlist.

rcond
: [float, optional] The condition number for the regression of atomic energy.

preset_out_bias
: [Dict[str, List[Optional[torch.Tensor]]], optional] Specifying atomic energy contribution in vacuum. Given by key:value pairs. The value is a list specifying the bias. the elements can be None or np.array of output shape. For example: [None, [2.]] means type 0 is not set, type 1 is set to [2.] The set_davg_zero key in the descrptor should be set.

init_out_stat()
: Initialize the output bias.

__setitem__(key, value)

__getitem__(key)

get_type_map() → List[str]
: Get the type map.

reinit_atom_exclude(exclude_types: List[int] = [])

reinit_pair_exclude(exclude_types: List[Tuple[int, int]] = [])

make_atom_mask(atype: torch.Tensor) → torch.Tensor
: The atoms with type < 0 are treated as virutal atoms, which serves as place-holders for multi-frame calculations with different number of atoms in different frames.

Parameters

atype
: Atom types. >= 0 for real atoms <0 for virtual atoms.

Returns

mask
True for real atoms and False for virutal atoms.

atomic_output_def() → deepmd.dpmodel.output_def.FittingOutputDef
Get the output def of the atomic model.

By default it is the same as FittingOutputDef, but it allows model level wrapper of the output defined by the developer.

forward_common_atomic(extended_coord: torch.Tensor, extended_atype: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None) → Dict[str, torch.Tensor]
Common interface for atomic inference.

This method accept extended coordinates, extended atom typs, neighbor list, and predict the atomic contribution of the fit property.

Parameters

extended_coord
extended coodinates, shape: nf x (nall x 3)

extended_atype
extended atom typs, shape: nf x nall for a type < 0 indicating the atomic is virtual.

nlist
neighbor list, shape: nf x nloc x nsel

mapping
extended to local index mapping, shape: nf x nall

fparam
frame parameters, shape: nf x dim_fparam

aparam
atomic parameter, shape: nf x nloc x dim_aparam

comm_dict
The data needed for communication for parallel inference.

Returns

ret_dict
dict of output atomic properties. should implement the definition of fitting_output_def. ret_dict["mask"] of shape nf x nloc will be provided. ret_dict["mask"][ff,ii] == 1 indicating the ii-th atom of the ff-th frame is real. ret_dict["mask"][ff,ii] == 0 indicating the ii-th atom of the ff-th frame is virtual.

serialize() → dict

classmethod deserialize(data: dict) → BaseAtomicModel

abstract compute_or_load_stat(merged: Callable[[], List[dict]] | List[dict], stat_file_path: deepmd.utils.path.DPPath | None = None)
Compute the output statistics (e.g. energy bias) for the fitting net from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

stat_file_path
[Optional[DPPath]] The path to the stat file.

compute_or_load_out_stat(merged: Callable[[], List[dict]] | List[dict], stat_file_path: deepmd.utils.path.DPPath | None = None)

Compute the output statistics (e.g. energy bias) for the fitting net from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

stat_file_path
[Optional[DPPath]] The path to the stat file.

apply_out_stat(ret: Dict[str, torch.Tensor], atype: torch.Tensor)

Apply the stat to each atomic output. The developer may override the method to define how the bias is applied to the atomic output of the model.

Parameters

ret
The returned dict by the forward_atomic method

atype
The atom types. nf x nloc

change_out_bias(sample_merged, stat_file_path: deepmd.utils.path.DPPath | None = None, bias_adjust_mode='change-by-statistic') → None

Change the output bias according to the input data and the pretrained model.

Parameters

sample_merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

bias_adjust_mode
[str] The mode for changing output bias : ['change-by-statistic', 'set-by-statistic']
'change-by-statistic' : perform predictions on labels of target dataset,

and do least square on the errors to obtain the target shift as bias.

'set-by-statistic' : directly use the statistic output bias in the target dataset.

stat_file_path
[Optional[DPPath]] The path to the stat file.

_get_forward_wrapper_func() → Callable[Ellipsis, torch.Tensor]

Get a forward wrapper of the atomic model for output bias calculation.

_default_bias()

_default_std()

_varsize(shape: List[int]) → int

_get_bias_index(kk: str) → int

_store_out_stat(out_bias: Dict[str, torch.Tensor], out_std: Dict[str, torch.Tensor], add: bool = False)

_fetch_out_stat(keys: List[str]) → Tuple[Dict[str, torch.Tensor], Dict[str, torch.Tensor]]

deepmd.pt.model.atomic_model.dipole_atomic_model

Module Contents

Classes

DPDipoleAtomicModel	Model give atomic prediction of some physical property.

class deepmd.pt.model.atomic_model.dipole_atomic_model.DPDipoleAtomicModel(descriptor, fitting, type_map, **kwargs)

Bases: *deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel*

Model give atomic prediction of some physical property.

Parameters

descriptor
Descriptor

fitting_net
Fitting net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

apply_out_stat(ret: Dict[str, torch.Tensor], atype: torch.Tensor)

Apply the stat to each atomic output. The developer may override the method to define how the bias is applied to the atomic output of the model.

Parameters

ret
The returned dict by the forward_atomic method

atype
The atom types. nf x nloc

deepmd.pt.model.atomic_model.dos_atomic_model

Module Contents

Classes

DPDOSAtomicModel	Model give atomic prediction of some physical property.

class deepmd.pt.model.atomic_model.dos_atomic_model.**DPDOSAtomicModel**(descriptor, fitting, type_map, **kwargs)

Bases: *deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel*

Model give atomic prediction of some physical property.

Parameters

descriptor
Descriptor

fitting_net
Fitting net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

deepmd.pt.model.atomic_model.dp_atomic_model

Module Contents

Classes

DPAtomicModel	Model give atomic prediction of some physical property.

Attributes

log

deepmd.pt.model.atomic_model.dp_atomic_model.log

class deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel(descriptor, fitting, type_map: List[str], **kwargs)

Bases: *deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel*

Model give atomic prediction of some physical property.

Parameters

descriptor
: Descriptor

fitting_net
: Fitting net

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

fitting_output_def() → deepmd.dpmodel.FittingOutputDef

Get the output def of the fitting net.

get_rcut() → float

Get the cut-off radius.

get_sel() → List[int]

Get the neighbor selection.

mixed_types() → bool

If true, the model 1. assumes total number of atoms aligned across frames; 2. uses a neighbor list that does not distinguish different atomic types.

If false, the model 1. assumes total number of atoms of each atom type aligned across frames; 2. uses a neighbor list that distinguishes different atomic types.

serialize() → dict

classmethod deserialize(data) → DPAtomicModel

forward_atomic(extended_coord, extended_atype, nlist, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None) → Dict[str, torch.Tensor]

Return atomic prediction.

Parameters

extended_coord
: coodinates in extended region

extended_atype
: atomic type in extended region

nlist
neighbor list. nf x nloc x nsel

mapping
mapps the extended indices to local indices

fparam
frame parameter. nf x ndf

aparam
atomic parameter. nf x nloc x nda

Returns

result_dict
the result dict, defined by the FittingOutputDef.

get_out_bias() → torch.Tensor

compute_or_load_stat(sampled_func, stat_file_path: deepmd.utils.path.DPPath | None = None)

Compute or load the statistics parameters of the model, such as mean and standard deviation of descriptors or the energy bias of the fitting net. When sampled is provided, all the statistics parameters will be calculated (or re-calculated for update), and saved in the stat_file_path`(s). When `sampled is not provided, it will check the existence of `stat_file_path`(s) and load the calculated statistics parameters.

Parameters

sampled_func
The lazy sampled function to get data frames from different data systems.

stat_file_path
The dictionary of paths to the statistics files.

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim).

If False, the shape is (nframes, nloc, ndim).

deepmd.pt.model.atomic_model.energy_atomic_model

Module Contents

Classes

DPEnergyAtomicModel	Model give atomic prediction of some physical property.

class deepmd.pt.model.atomic_model.energy_atomic_model.**DPEnergyAtomicModel**(descriptor, fitting, type_map, **kwargs)

Bases: *deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel*

Model give atomic prediction of some physical property.

Parameters

descriptor
Descriptor

fitting_net
Fitting net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

deepmd.pt.model.atomic_model.linear_atomic_model

Module Contents

Classes

LinearEnergyAtomicModel	Linear model make linear combinations of several existing models.
DPZBLLinearEnergyAtomicModel	Model linearly combine a list of AtomicModels.

class deepmd.pt.model.atomic_model.linear_atomic_model.**LinearEnergyAtomicModel**(models: List[deepmd.pt.model.atomic type_map: List[str], **kwargs)

Bases: *deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel*

Linear model make linear combinations of several existing models.

Parameters

models
: [list[DPAtomicModel or PairTabAtomicModel]] A list of models to be combined. PairTabAtomicModel must be used together with a DPAtomicModel.

type_map
: [list[str]] Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

mixed_types() → bool

If true, the model 1. assumes total number of atoms aligned across frames; 2. uses a neighbor list that does not distinguish different atomic types.

If false, the model 1. assumes total number of atoms of each atom type aligned across frames; 2. uses a neighbor list that distinguishes different atomic types.

get_out_bias() → torch.Tensor

get_rcut() → float

Get the cut-off radius.

get_type_map() → List[str]

Get the type map.

get_model_rcuts() → List[float]

Get the cut-off radius for each individual models.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_model_nsels() → List[int]

Get the processed sels for each individual models. Not distinguishing types.

get_model_sels() → List[List[int]]

Get the sels for each individual models.

_sort_rcuts_sels() → Tuple[List[float], List[int]]

forward_atomic(extended_coord: torch.Tensor, extended_atype: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None) → Dict[str, torch.Tensor]

Return atomic prediction.

Parameters

extended_coord
: coodinates in extended region, (nframes, nall * 3)

extended_atype
: atomic type in extended region, (nframes, nall)

nlist
: neighbor list, (nframes, nloc, nsel).

mapping
: mapps the extended indices to local indices.

fparam
: frame parameter. (nframes, ndf)

aparam
atomic parameter. (nframes, nloc, nda)

Returns

result_dict
the result dict, defined by the fitting net output def.

apply_out_stat(ret: Dict[str, torch.Tensor], atype: torch.Tensor)

Apply the stat to each atomic output. The developer may override the method to define how the bias is applied to the atomic output of the model.

Parameters

ret
The returned dict by the forward_atomic method

atype
The atom types. nf x nloc

static remap_atype(ori_map: List[str], new_map: List[str]) → torch.Tensor

This method is used to map the atype from the common type_map to the original type_map of indivial AtomicModels. It creates a index mapping for the conversion.

Parameters

ori_map
[List[str]] The original type map of an AtomicModel.

new_map
[List[str]] The common type map of the DPZBLLinearEnergyAtomicModel, created by the get_type_map method, must be a subset of the ori_map.

Returns

torch.Tensor

fitting_output_def() → deepmd.dpmodel.FittingOutputDef

Get the output def of developer implemented atomic models.

serialize() → dict

classmethod deserialize(data: dict) → LinearEnergyAtomicModel

_compute_weight(extended_coord, extended_atype, nlists_) → List[torch.Tensor]

This should be a list of user defined weights that matches the number of models to be combined.

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim).

If False, the shape is (nframes, nloc, ndim).

compute_or_load_out_stat(merged: Callable[[], List[dict]] | List[dict], stat_file_path: deepmd.utils.path.DPPath | None = None)

Compute the output statistics (e.g. energy bias) for the fitting net from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

stat_file_path
[Optional[DPPath]] The path to the stat file.

compute_or_load_stat(sampled_func, stat_file_path: deepmd.utils.path.DPPath | None = None)

Compute or load the statistics parameters of the model, such as mean and standard deviation of descriptors or the energy bias of the fitting net. When sampled is provided, all the statistics parameters will be calculated (or re-calculated for update), and saved in the stat_file_path`(s). When `sampled is not provided, it will check the existence of `stat_file_path`(s) and load the calculated statistics parameters.

Parameters

sampled_func
The lazy sampled function to get data frames from different data systems.

stat_file_path
The dictionary of paths to the statistics files.

class deepmd.pt.model.atomic_model.linear_atomic_model.**DPZBLLinearEnergyAtomicModel**(dp_model: deepmd.pt.model.atom
zbl_model: deepmd.pt.model.atom
sw_rmin: float,
sw_rmax: float,
type_map: List[str],
smin_alpha: float | None = 0.1,
**kwargs)

Bases: *LinearEnergyAtomicModel*

Model linearly combine a list of AtomicModels.

Parameters

dp_model
The DPAtomicModel being combined.

zbl_model
: The PairTable model being combined.

sw_rmin
: The lower boundary of the interpolation between short-range tabulated interaction and DP.

sw_rmax
: The upper boundary of the interpolation between short-range tabulated interaction and DP.

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

smin_alpha
: The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin.

serialize() → dict

classmethod deserialize(data) → DPZBLLinearEnergyAtomicModel

_compute_weight(extended_coord: torch.Tensor, extended_atype: torch.Tensor, nlists_: List[torch.Tensor]) → List[torch.Tensor]

ZBL weight.

Returns

List[**torch.Tensor**]
: the atomic ZBL weight for interpolation. (nframes, nloc, 1)

deepmd.pt.model.atomic_model.pairtab_atomic_model

Module Contents

Classes

PairTabAtomicModel	Pairwise tabulation energy model.

class deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel(tab_file: str, rcut: float, sel: int | List[int], type_map: List[str], **kwargs)

Bases: *deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel*

Pairwise tabulation energy model.

This model can be used to tabulate the pairwise energy between atoms for either short-range or long-range interactions, such as D3, LJ, ZBL, etc. It should not be used alone, but rather as one submodel of a linear (sum) model, such as DP+D3.

Do not put the model on the first model of a linear model, since the linear model fetches the type map from the first model.

At this moment, the model does not smooth the energy at the cutoff radius, so one needs to make sure the energy has been smoothed to zero.

Parameters

tab_file
: [`str`] The path to the tabulation file.

rcut
: [`float`] The cutoff radius.

sel
: [`int` or `list`[`int`]] The maxmum number of atoms in the cut-off radius.

type_map
: [`List`[`str`]] Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

rcond
: [`float`, `optional`] The condition number for the regression of atomic energy.

atom_ener
: Specifying atomic energy contribution in vacuum. The set_davg_zero key in the descrptor should be set.

_set_pairtab(tab_file: str, rcut: float) → deepmd.utils.pair_tab.PairTab

fitting_output_def() → deepmd.dpmodel.FittingOutputDef

Get the output def of developer implemented atomic models.

get_out_bias() → torch.Tensor

get_rcut() → float

Get the cut-off radius.

get_type_map() → List[str]

Get the type map.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_nsel() → int

Returns the total number of selected neighboring atoms in the cut-off radius.

mixed_types() → bool

If true, the model 1. assumes total number of atoms aligned across frames; 2. uses a neighbor list that does not distinguish different atomic types.

If false, the model 1. assumes total number of atoms of each atom type aligned across frames; 2. uses a neighbor list that distinguishes different atomic types.

serialize() → dict

classmethod deserialize(data) → PairTabAtomicModel

compute_or_load_stat(merged: Callable[[], List[dict]] | List[dict], stat_file_path: deepmd.utils.path.DPPath | None = None)

Compute the output statistics (e.g. energy bias) for the fitting net from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

stat_file_path
[Optional[DPPath]] The path to the stat file.

forward_atomic(extended_coord: torch.Tensor, extended_atype: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False, comm_dict: Dict[str, torch.Tensor] | None = None) → Dict[str, torch.Tensor]

_pair_tabulated_inter(nlist: torch.Tensor, i_type: torch.Tensor, j_type: torch.Tensor, rr: torch.Tensor) → torch.Tensor

Pairwise tabulated energy.

Parameters

nlist
[**torch.Tensor**] The unmasked neighbour list. (nframes, nloc)

i_type
[**torch.Tensor**] The integer representation of atom type for all local atoms for all frames. (nframes, nloc)

j_type
[**torch.Tensor**] The integer representation of atom type for all neighbour atoms of all local atoms for all frames. (nframes, nloc, nnei)

rr
[**torch.Tensor**] The salar distance vector between two atoms. (nframes, nloc, nnei)

Returns

torch.Tensor
The masked atomic energy for all local atoms for all frames. (nframes, nloc, nnei)

Raises

Exception
If the distance is beyond the table.

Notes

This function is used to calculate the pairwise energy between two atoms. It uses a table containing cubic spline coefficients calculated in PairTab.

static _get_pairwise_dist(coords: torch.Tensor, nlist: torch.Tensor) → torch.Tensor

Get pairwise distance dr.

Parameters

coords
[`torch.Tensor`] The coordinate of the atoms, shape of (nframes, nall, 3).

nlist
The masked nlist, shape of (nframes, nloc, nnei)

Returns

torch.Tensor
The pairwise distance between the atoms (nframes, nloc, nnei).

static _extract_spline_coefficient(i_type: torch.Tensor, j_type: torch.Tensor, idx: torch.Tensor, tab_data: torch.Tensor, nspline: int) → torch.Tensor

Extract the spline coefficient from the table.

Parameters

i_type
[`torch.Tensor`] The integer representation of atom type for all local atoms for all frames. (nframes, nloc)

j_type
[`torch.Tensor`] The integer representation of atom type for all neighbour atoms of all local atoms for all frames. (nframes, nloc, nnei)

idx
[`torch.Tensor`] The index of the spline coefficient. (nframes, nloc, nnei)

tab_data
[`torch.Tensor`] The table storing all the spline coefficient. (ntype, ntype, nspline, 4)

nspline
[int] The number of splines in the table.

Returns

torch.Tensor
The spline coefficient. (nframes, nloc, nnei, 4), shape may be squeezed.

static _calculate_ener(coef: torch.Tensor, uu: torch.Tensor) → torch.Tensor

Calculate energy using spline coeeficients.

Parameters

coef
[`torch.Tensor`] The spline coefficients. (nframes, nloc, nnei, 4)

uu
[`torch.Tensor`] The atom displancemnt used in interpolation and extrapolation (nframes, nloc, nnei)

Returns

torch.Tensor
The atomic energy for all local atoms for all frames. (nframes, nloc, nnei)

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this atomic model.

`get_dim_aparam()` → int

Get the number (dimension) of atomic parameters of this atomic model.

`get_sel_type()` → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

`is_aparam_nall()` → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim).

If False, the shape is (nframes, nloc, ndim).

deepmd.pt.model.atomic_model.polar_atomic_model

Module Contents

Classes

DPPolarAtomicModel	Model give atomic prediction of some physical property.

`class deepmd.pt.model.atomic_model.polar_atomic_model.DPPolarAtomicModel(descriptor, fitting, type_map, **kwargs)`

Bases: *deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel*

Model give atomic prediction of some physical property.

Parameters

descriptor
: Descriptor

fitting_net
: Fitting net

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

`apply_out_stat(ret: Dict[str, torch.Tensor], atype: torch.Tensor)`

Apply the stat to each atomic output.

Parameters

ret
: The returned dict by the forward_atomic method

atype
: The atom types. nf x nloc

Package Contents

Classes

BaseAtomicModel	The base of atomic model.
DPDipoleAtomicModel	Model give atomic prediction of some physical property.
DPDOSAtomicModel	Model give atomic prediction of some physical property.
DPAtomicModel	Model give atomic prediction of some physical property.
DPEnergyAtomicModel	Model give atomic prediction of some physical property.
DPZBLLinearEnergyAtomicModel	Model linearly combine a list of AtomicModels.
LinearEnergyAtomicModel	Linear model make linear combinations of several existing models.
PairTabAtomicModel	Pairwise tabulation energy model.
DPPolarAtomicModel	Model give atomic prediction of some physical property.

class deepmd.pt.model.atomic_model.**BaseAtomicModel**(type_map: List[str], atom_exclude_types: List[int] = [], pair_exclude_types: List[Tuple[int, int]] = [], rcond: float | None = None, preset_out_bias: Dict[str, torch.Tensor] | None = None)

Bases: `torch.nn.Module`, `BaseAtomicModel_`

The base of atomic model.

Parameters

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

atom_exclude_types
: Exclude the atomic contribution of the given types

pair_exclude_types
: Exclude the pair of atoms of the given types from computing the output of the atomic model. Implemented by removing the pairs from the nlist.

rcond
: [`float`, `optional`] The condition number for the regression of atomic energy.

preset_out_bias
: [`Dict[str, List[Optional[torch.Tensor]]]`, `optional`] Specifying atomic energy contribution in vacuum. Given by key:value pairs. The value is a list specifying the bias. the elements can be None or np.array of output shape. For example: [None, [2.]] means type 0 is not set, type 1 is set to [2.] The set_davg_zero key in the descrptor should be set.

init_out_stat()

Initialize the output bias.

__setitem__(key, value)

__getitem__(key)

get_type_map() → List[str]

Get the type map.

reinit_atom_exclude(exclude_types: List[int] = [])

reinit_pair_exclude(exclude_types: List[Tuple[int, int]] = [])

make_atom_mask(atype: torch.Tensor) → torch.Tensor

The atoms with type < 0 are treated as virutal atoms, which serves as place-holders for multi-frame calculations with different number of atoms in different frames.

Parameters

atype
Atom types. >= 0 for real atoms <0 for virtual atoms.

Returns

mask
True for real atoms and False for virutal atoms.

atomic_output_def() → deepmd.dpmodel.output_def.FittingOutputDef

Get the output def of the atomic model.

By default it is the same as FittingOutputDef, but it allows model level wrapper of the output defined by the developer.

forward_common_atomic(extended_coord: torch.Tensor, extended_atype: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None) → Dict[str, torch.Tensor]

Common interface for atomic inference.

This method accept extended coordinates, extended atom typs, neighbor list, and predict the atomic contribution of the fit property.

Parameters

extended_coord
extended coodinates, shape: nf x (nall x 3)

extended_atype
extended atom typs, shape: nf x nall for a type < 0 indicating the atomic is virtual.

nlist
neighbor list, shape: nf x nloc x nsel

mapping
extended to local index mapping, shape: nf x nall

fparam
frame parameters, shape: nf x dim_fparam

aparam
atomic parameter, shape: nf x nloc x dim_aparam

comm_dict
The data needed for communication for parallel inference.

Returns

ret_dict
dict of output atomic properties. should implement the definition of fitting_output_def. ret_dict["mask"] of shape nf x nloc will be provided. ret_dict["mask"][ff,ii] == 1 indicating the ii-th atom of the ff-th frame is real. ret_dict["mask"][ff,ii] == 0 indicating the ii-th atom of the ff-th frame is virtual.

serialize() → dict

classmethod deserialize(data: dict) → BaseAtomicModel

abstract compute_or_load_stat(merged: Callable[[], List[dict]] | List[dict], stat_file_path: deepmd.utils.path.DPPath | None = None)

Compute the output statistics (e.g. energy bias) for the fitting net from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

stat_file_path
[Optional[DPPath]] The path to the stat file.

compute_or_load_out_stat(merged: Callable[[], List[dict]] | List[dict], stat_file_path: deepmd.utils.path.DPPath | None = None)

Compute the output statistics (e.g. energy bias) for the fitting net from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

stat_file_path
[Optional[DPPath]] The path to the stat file.

apply_out_stat(ret: Dict[str, torch.Tensor], atype: torch.Tensor)

Apply the stat to each atomic output. The developer may override the method to define how the bias is applied to the atomic output of the model.

Parameters

ret
The returned dict by the forward_atomic method

atype
The atom types. nf x nloc

change_out_bias(sample_merged, stat_file_path: deepmd.utils.path.DPPath | None = None, bias_adjust_mode='change-by-statistic') → None

Change the output bias according to the input data and the pretrained model.

Parameters

sample_merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

bias_adjust_mode
[str] The mode for changing output bias : ['change-by-statistic', 'set-by-statistic']
'change-by-statistic' : perform predictions on labels of target dataset,

and do least square on the errors to obtain the target shift as bias.

'set-by-statistic' : directly use the statistic output bias in the target dataset.

stat_file_path
[Optional[DPPath]] The path to the stat file.

_get_forward_wrapper_func() → Callable[Ellipsis, torch.Tensor]

Get a forward wrapper of the atomic model for output bias calculation.

_default_bias()

_default_std()

_varsize(shape: List[int]) → int

_get_bias_index(kk: str) → int

_store_out_stat(out_bias: Dict[str, torch.Tensor], out_std: Dict[str, torch.Tensor], add: bool = False)

_fetch_out_stat(keys: List[str]) → Tuple[Dict[str, torch.Tensor], Dict[str, torch.Tensor]]

class deepmd.pt.model.atomic_model.DPDipoleAtomicModel(descriptor, fitting, type_map, **kwargs)

Bases: *deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel*

Model give atomic prediction of some physical property.

Parameters

descriptor
Descriptor

fitting_net
Fitting net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

apply_out_stat(ret: Dict[str, torch.Tensor], atype: torch.Tensor)

Apply the stat to each atomic output. The developer may override the method to define how the bias is applied to the atomic output of the model.

Parameters

ret
The returned dict by the forward_atomic method

atype
The atom types. nf x nloc

class deepmd.pt.model.atomic_model.DPDOSAtomicModel(descriptor, fitting, type_map, **kwargs)

Bases: *deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel*

Model give atomic prediction of some physical property.

Parameters

descriptor
Descriptor

fitting_net
Fitting net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

class deepmd.pt.model.atomic_model.DPAtomicModel(descriptor, fitting, type_map: List[str], **kwargs)

Bases: *deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel*

Model give atomic prediction of some physical property.

Parameters

descriptor
Descriptor

fitting_net
Fitting net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

fitting_output_def() → deepmd.dpmodel.FittingOutputDef

Get the output def of the fitting net.

get_rcut() → float

Get the cut-off radius.

get_sel() → List[int]

Get the neighbor selection.

mixed_types() → bool

If true, the model 1. assumes total number of atoms aligned across frames; 2. uses a neighbor list that does not distinguish different atomic types.

If false, the model 1. assumes total number of atoms of each atom type aligned across frames; 2. uses a neighbor list that distinguishes different atomic types.

serialize() → dict

classmethod deserialize(data) → DPAtomicModel

forward_atomic(extended_coord, extended_atype, nlist, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None) → Dict[str, torch.Tensor]

Return atomic prediction.

Parameters

extended_coord
coodinates in extended region

extended_atype
atomic type in extended region

nlist
neighbor list. nf x nloc x nsel

mapping
mapps the extended indices to local indices

fparam
frame parameter. nf x ndf

aparam
atomic parameter. nf x nloc x nda

Returns

result_dict
the result dict, defined by the FittingOutputDef.

get_out_bias() → torch.Tensor

compute_or_load_stat(sampled_func, stat_file_path: deepmd.utils.path.DPPath | None = None)

Compute or load the statistics parameters of the model, such as mean and standard deviation of descriptors or the energy bias of the fitting net. When sampled is provided, all the statistics parameters will be calculated (or re-calculated for update), and saved in the stat_file_path`(s). When `sampled is not provided, it will check the existence of `stat_file_path`(s) and load the calculated statistics parameters.

Parameters

sampled_func
The lazy sampled function to get data frames from different data systems.

stat_file_path
The dictionary of paths to the statistics files.

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this atomic model.

`get_dim_aparam()` → int

Get the number (dimension) of atomic parameters of this atomic model.

`get_sel_type()` → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

`is_aparam_nall()` → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim).

If False, the shape is (nframes, nloc, ndim).

`class deepmd.pt.model.atomic_model.DPEnergyAtomicModel(descriptor, fitting, type_map, **kwargs)`

Bases: *deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel*

Model give atomic prediction of some physical property.

Parameters

descriptor
Descriptor

fitting_net
Fitting net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

`class deepmd.pt.model.atomic_model.DPZBLLinearEnergyAtomicModel(dp_model: deepmd.pt.model.atomic_model.dp_atomic_mo zbl_model: deepmd.pt.model.atomic_model.pairtab_atomi sw_rmin: float, sw_rmax: float, type_map: List[str], smin_alpha: float | None = 0.1, **kwargs)`

Bases: *LinearEnergyAtomicModel*

Model linearly combine a list of AtomicModels.

Parameters

dp_model
The DPAtomicModel being combined.

zbl_model
The PairTable model being combined.

sw_rmin
The lower boundary of the interpolation between short-range tabulated interaction and DP.

sw_rmax
The upper boundary of the interpolation between short-range tabulated interaction and DP.

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

smin_alpha
The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin.

serialize() → dict

classmethod deserialize(data) → DPZBLLinearEnergyAtomicModel

_compute_weight(extended_coord: torch.Tensor, extended_atype: torch.Tensor, nlists_: List[torch.Tensor]) → List[torch.Tensor]

ZBL weight.

Returns

List[**torch.Tensor**]
the atomic ZBL weight for interpolation. (nframes, nloc, 1)

class deepmd.pt.model.atomic_model.LinearEnergyAtomicModel(models: List[deepmd.pt.model.atomic_model.base_atomic_m
type_map: List[str], **kwargs)

Bases: *deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel*

Linear model make linear combinations of several existing models.

Parameters

models
[list[*DPAtomicModel* or *PairTabAtomicModel*]] A list of models to be combined. PairTabAtomicModel must be used together with a DPAtomicModel.

type_map
[list[str]] Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

mixed_types() → bool

If true, the model 1. assumes total number of atoms aligned across frames; 2. uses a neighbor list that does not distinguish different atomic types.

If false, the model 1. assumes total number of atoms of each atom type aligned across frames; 2. uses a neighbor list that distinguishes different atomic types.

get_out_bias() → torch.Tensor

get_rcut() → float

Get the cut-off radius.

get_type_map() → List[str]

Get the type map.

get_model_rcuts() → List[float]

Get the cut-off radius for each individual models.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_model_nsels() → List[int]

Get the processed sels for each individual models. Not distinguishing types.

get_model_sels() → List[List[int]]

Get the sels for each individual models.

`_sort_rcuts_sels()` → Tuple[List[float], List[int]]

`forward_atomic`(extended_coord: torch.Tensor, extended_atype: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None) → Dict[str, torch.Tensor]

Return atomic prediction.

Parameters

extended_coord
coodinates in extended region, (nframes, nall * 3)

extended_atype
atomic type in extended region, (nframes, nall)

nlist
neighbor list, (nframes, nloc, nsel).

mapping
mapps the extended indices to local indices.

fparam
frame parameter. (nframes, ndf)

aparam
atomic parameter. (nframes, nloc, nda)

Returns

result_dict
the result dict, defined by the fitting net output def.

`apply_out_stat`(ret: Dict[str, torch.Tensor], atype: torch.Tensor)

Apply the stat to each atomic output. The developer may override the method to define how the bias is applied to the atomic output of the model.

Parameters

ret
The returned dict by the forward_atomic method

atype
The atom types. nf x nloc

`static remap_atype`(ori_map: List[str], new_map: List[str]) → torch.Tensor

This method is used to map the atype from the common type_map to the original type_map of indivial AtomicModels. It creates a index mapping for the conversion.

Parameters

ori_map
[`List[str]`] The original type map of an AtomicModel.

new_map
[`List[str]`] The common type map of the DPZBLLinearEnergyAtomicModel, created by the get_type_map method, must be a subset of the ori_map.

Returns

torch.Tensor

fitting_output_def() → deepmd.dpmodel.FittingOutputDef
Get the output def of developer implemented atomic models.

serialize() → dict

classmethod deserialize(data: dict) → LinearEnergyAtomicModel

_compute_weight(extended_coord, extended_atype, nlists_) → List[torch.Tensor]
This should be a list of user defined weights that matches the number of models to be combined.

get_dim_fparam() → int
Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam() → int
Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]
Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool
Check whether the shape of atomic parameters is (nframes, nall, ndim).

If False, the shape is (nframes, nloc, ndim).

compute_or_load_out_stat(merged: Callable[[], List[dict]] | List[dict], stat_file_path: deepmd.utils.path.DPPath | None = None)
Compute the output statistics (e.g. energy bias) for the fitting net from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

stat_file_path
[Optional[DPPath]] The path to the stat file.

compute_or_load_stat(sampled_func, stat_file_path: deepmd.utils.path.DPPath | None = None)
Compute or load the statistics parameters of the model, such as mean and standard deviation of descriptors or the energy bias of the fitting net. When sampled is provided, all the statistics parameters will be calculated (or re-calculated for update), and saved in the stat_file_path`(s). When `sampled is not provided, it will check the existence of `stat_file_path`(s) and load the calculated statistics parameters.

Parameters

sampled_func
The lazy sampled function to get data frames from different data systems.

stat_file_path
The dictionary of paths to the statistics files.

class deepmd.pt.model.atomic_model.PairTabAtomicModel(tab_file: str, rcut: float, sel: int | List[int], type_map: List[str], **kwargs)

Bases: *deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel*

Pairwise tabulation energy model.

This model can be used to tabulate the pairwise energy between atoms for either short-range or long-range interactions, such as D3, LJ, ZBL, etc. It should not be used alone, but rather as one submodel of a linear (sum) model, such as DP+D3.

Do not put the model on the first model of a linear model, since the linear model fetches the type map from the first model.

At this moment, the model does not smooth the energy at the cutoff radius, so one needs to make sure the energy has been smoothed to zero.

Parameters

tab_file
: [str] The path to the tabulation file.

rcut
: [float] The cutoff radius.

sel
: [int or list[int]] The maxmum number of atoms in the cut-off radius.

type_map
: [List[str]] Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

rcond
: [float, optional] The condition number for the regression of atomic energy.

atom_ener
: Specifying atomic energy contribution in vacuum. The set_davg_zero key in the descrptor should be set.

_set_pairtab(tab_file: str, rcut: float) → deepmd.utils.pair_tab.PairTab

fitting_output_def() → deepmd.dpmodel.FittingOutputDef

Get the output def of developer implemented atomic models.

get_out_bias() → torch.Tensor

get_rcut() → float

Get the cut-off radius.

get_type_map() → List[str]

Get the type map.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_nsel() → int

Returns the total number of selected neighboring atoms in the cut-off radius.

mixed_types() → bool

If true, the model 1. assumes total number of atoms aligned across frames; 2. uses a neighbor list that does not distinguish different atomic types.

If false, the model 1. assumes total number of atoms of each atom type aligned across frames; 2. uses a neighbor list that distinguishes different atomic types.

serialize() → dict

classmethod deserialize(data) → PairTabAtomicModel

compute_or_load_stat(merged: Callable[[], List[dict]] | List[dict], stat_file_path: deepmd.utils.path.DPPath | None = None)

Compute the output statistics (e.g. energy bias) for the fitting net from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

stat_file_path
[Optional[DPPath]] The path to the stat file.

forward_atomic(extended_coord: torch.Tensor, extended_atype: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False, comm_dict: Dict[str, torch.Tensor] | None = None) → Dict[str, torch.Tensor]

_pair_tabulated_inter(nlist: torch.Tensor, i_type: torch.Tensor, j_type: torch.Tensor, rr: torch.Tensor) → torch.Tensor

Pairwise tabulated energy.

Parameters

nlist
[**torch.Tensor**] The unmasked neighbour list. (nframes, nloc)

i_type
[**torch.Tensor**] The integer representation of atom type for all local atoms for all frames. (nframes, nloc)

j_type
[**torch.Tensor**] The integer representation of atom type for all neighbour atoms of all local atoms for all frames. (nframes, nloc, nnei)

rr
[**torch.Tensor**] The salar distance vector between two atoms. (nframes, nloc, nnei)

Returns

torch.Tensor
The masked atomic energy for all local atoms for all frames. (nframes, nloc, nnei)

Raises

Exception
If the distance is beyond the table.

Notes

This function is used to calculate the pairwise energy between two atoms. It uses a table containing cubic spline coefficients calculated in PairTab.

static _get_pairwise_dist(coords: torch.Tensor, nlist: torch.Tensor) → torch.Tensor

Get pairwise distance dr.

Parameters

coords
[torch.Tensor] The coordinate of the atoms, shape of (nframes, nall, 3).

nlist
The masked nlist, shape of (nframes, nloc, nnei)

Returns

torch.Tensor
The pairwise distance between the atoms (nframes, nloc, nnei).

static _extract_spline_coefficient(i_type: torch.Tensor, j_type: torch.Tensor, idx: torch.Tensor, tab_data: torch.Tensor, nspline: int) → torch.Tensor

Extract the spline coefficient from the table.

Parameters

i_type
[torch.Tensor] The integer representation of atom type for all local atoms for all frames. (nframes, nloc)

j_type
[torch.Tensor] The integer representation of atom type for all neighbour atoms of all local atoms for all frames. (nframes, nloc, nnei)

idx
[torch.Tensor] The index of the spline coefficient. (nframes, nloc, nnei)

tab_data
[torch.Tensor] The table storing all the spline coefficient. (ntype, ntype, nspline, 4)

nspline
[int] The number of splines in the table.

Returns

torch.Tensor
The spline coefficient. (nframes, nloc, nnei, 4), shape may be squeezed.

static _calculate_ener(coef: torch.Tensor, uu: torch.Tensor) → torch.Tensor

Calculate energy using spline coeeficients.

Parameters

coef
[torch.Tensor] The spline coefficients. (nframes, nloc, nnei, 4)

uu
[torch.Tensor] The atom displancemnt used in interpolation and extrapolation (nframes, nloc, nnei)

Returns

torch.Tensor
The atomic energy for all local atoms for all frames. (nframes, nloc, nnei)

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim).

If False, the shape is (nframes, nloc, ndim).

class deepmd.pt.model.atomic_model.**DPPolarAtomicModel**(descriptor, fitting, type_map, **kwargs)

Bases: *deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel*

Model give atomic prediction of some physical property.

Parameters

descriptor
Descriptor

fitting_net
Fitting net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

apply_out_stat(ret: Dict[str, torch.Tensor], atype: torch.Tensor)

Apply the stat to each atomic output.

Parameters

ret
The returned dict by the forward_atomic method

atype
The atom types. nf x nloc

deepmd.pt.model.backbone

Submodules

deepmd.pt.model.backbone.backbone

Module Contents

Classes

BackBone	Base class for all neural network modules.

class deepmd.pt.model.backbone.backbone.**BackBone**(**kwargs)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

abstract forward(**kwargs)

Calculate backBone.

deepmd.pt.model.backbone.evoformer2b

Module Contents

Classes

Evoformer2bBackBone	Base class for all neural network modules.

class deepmd.pt.model.backbone.evoformer2b.**Evoformer2bBackBone**(nnei, layer_num=6, attn_head=8, atomic_dim=1024, pair_dim=100, feature_dim=1024, ffn_dim=2048, post_ln=False, final_layer_norm=True, final_head_layer_norm=False, emb_layer_norm=False, atomic_residual=False, evo_residual=False, residual_factor=1.0, activation_function='gelu', **kwargs)

Bases: *deepmd.pt.model.backbone.BackBone*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(atomic_rep, pair_rep, nlist, nlist_type, nlist_mask)

Encoder the atomic and pair representations.

Args: - atomic_rep: Atomic representation with shape [nframes, nloc, atomic_dim]. - pair_rep: Pair representation with shape [nframes, nloc, nnei, pair_dim]. - nlist: Neighbor list with shape [nframes, nloc, nnei]. - nlist_type: Neighbor types with shape [nframes, nloc, nnei]. - nlist_mask: Neighbor mask with shape [nframes, nloc, nnei], False if blank.

Returns

- atomic_rep: **Atomic representation after encoder with shape** [**nframes**, **nloc**, **feature_dim**].
- transformed_atomic_rep: **Transformed atomic representation after encoder with shape** [**nframes**, **nloc**, **atomic_dim**].
- pair_rep: **Pair representation after encoder with shape** [**nframes**, **nloc**, **nnei**, **attn_head**].
- delta_pair_rep: **Delta pair representation after encoder with shape** [**nframes**, **nloc**, **nnei**, **attn_head**].
- norm_x: **Normalization loss of** atomic_rep.
- norm_delta_pair_rep: **Normalization loss of** delta_pair_rep.

Package Contents

Classes

BackBone	Base class for all neural network modules.

class deepmd.pt.model.backbone.**BackBone**(**kwargs)

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

abstract forward(**kwargs)

Calculate backBone.

deepmd.pt.model.descriptor

Submodules

deepmd.pt.model.descriptor.base_descriptor

Module Contents

deepmd.pt.model.descriptor.base_descriptor.**BaseDescriptor**

deepmd.pt.model.descriptor.descriptor

Module Contents

Classes

DescriptorBlock	The building block of descriptor.

Functions

make_default_type_embedding(ntypes)

Attributes

log

deepmd.pt.model.descriptor.descriptor.**log**

class deepmd.pt.model.descriptor.descriptor.**DescriptorBlock**(*args, **kwargs)

Bases: torch.nn.Module, abc.ABC, make_plugin_registry('DescriptorBlock')

The building block of descriptor. Given the input descriptor, provide with the atomic coordinates, atomic types and neighbor list, calculate the new descriptor.

local_cluster = False

abstract get_rcut() → float

Returns the cut-off radius.

abstract get_nsel() → int

Returns the number of selected atoms in the cut-off radius.

abstract get_sel() → List[int]

Returns the number of selected atoms for each type.

abstract get_ntypes() → int

Returns the number of element types.

abstract get_dim_out() → int

Returns the output dimension.

abstract get_dim_in() → int

Returns the output dimension.

abstract get_dim_emb() → int

Returns the embedding dimension.

abstract compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)

Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

abstract get_stats() → Dict[str, deepmd.utils.env_mat_stat.StatItem]

Get the statistics of the descriptor.

share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

abstract forward(nlist: torch.Tensor, extended_coord: torch.Tensor, extended_atype: torch.Tensor, extended_atype_embd: torch.Tensor | None = None, mapping: torch.Tensor | None = None)

Calculate DescriptorBlock.

deepmd.pt.model.descriptor.descriptor.**make_default_type_embedding**(ntypes)

deepmd.pt.model.descriptor.dpa1

Module Contents

Classes

DescrptDPA1	Attention-based descriptor which is proposed in the pretrainable DPA-1[1] model.

class deepmd.pt.model.descriptor.dpa1.DescrptDPA1(rcut: float, rcut_smth: float, sel: List[int] | int, ntypes: int, neuron: list = [25, 50, 100], axis_neuron: int = 16, tebd_dim: int = 8, tebd_input_mode: str = 'concat', set_davg_zero: bool = True, attn: int = 128, attn_layer: int = 2, attn_dotr: bool = True, attn_mask: bool = False, activation_function: str = 'tanh', precision: str = 'float64', resnet_dt: bool = False, exclude_types: List[Tuple[int, int]] = [], env_protection: float = 0.0, scaling_factor: int = 1.0, normalize=True, temperature=None, concat_output_tebd: bool = True, trainable: bool = True, trainable_ln: bool = True, ln_eps: float | None = 1e-05, smooth_type_embedding: bool = True, type_one_side: bool = False, stripped_type_embedding: bool | None = None, spin=None, type: str | None = None, seed: int | None = None, old_impl: bool = False)

Bases: *deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor*, torch.nn.Module

Attention-based descriptor which is proposed in the pretrainable DPA-1[1] model.

This descriptor, $\mathcal{D}^i \in \mathbb{R}^{M \times M_<}$, is given by

$$\mathcal{D}^i = \frac{1}{N_c^2}(\hat{\mathcal{G}}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \hat{\mathcal{G}}^i_<,$$

where $\hat{\mathcal{G}}^i$ represents the embedding matrix:math:mathcal{G}^i after additional self-attention mechanism and $\mathcal{R}^i$ is defined by the full case in the se_e2_a descriptor. Note that we obtain $\mathcal{G}^i$ using the type embedding method by default in this descriptor.

To perform the self-attention mechanism, the queries $\mathcal{Q}^{i,l} \in \mathbb{R}^{N_c \times d_k}$, keys $\mathcal{K}^{i,l} \in \mathbb{R}^{N_c \times d_k}$, and values $\mathcal{V}^{i,l} \in \mathbb{R}^{N_c \times d_v}$ are first obtained:

$$\left(\mathcal{Q}^{i,l}\right)_j = Q_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$

$$\left(\mathcal{K}^{i,l}\right)_j = K_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$

$$\left(\mathcal{V}^{i,l}\right)_j = V_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$

where Q_l, K_l, V_l represent three trainable linear transformations that output the queries and keys of dimension d_k and values of dimension d_v, and l is the index of the attention layer. The input embedding matrix to the attention layers, denoted by $\mathcal{G}^{i,0}$, is chosen as the two-body embedding matrix.

Then the scaled dot-product attention method is adopted:

$$A(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{V}^{i,l}, \mathcal{R}^{i,l}) = \varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right) \mathcal{V}^{i,l},$$

where $\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right) \in \mathbb{R}^{N_c \times N_c}$ is attention weights. In the original attention method, one typically has $\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}\right) = \mathrm{softmax}\left(\frac{\mathcal{Q}^{i,l}(\mathcal{K}^{i,l})^T}{\sqrt{d_k}}\right)$, with $\sqrt{d_k}$ being the normalization temperature. This is slightly modified to incorporate the angular information:

$$\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right) = \mathrm{softmax}\left(\frac{\mathcal{Q}^{i,l}(\mathcal{K}^{i,l})^T}{\sqrt{d_k}}\right) \odot \hat{\mathcal{R}}^i(\hat{\mathcal{R}}^i)^T,$$

where $\hat{\mathcal{R}}^i \in \mathbb{R}^{N_c \times 3}$ denotes normalized relative coordinates,
$\hat{\mathcal{R}}^i_j = \frac{\boldsymbol{r}_{ij}}{\|\boldsymbol{r}_{ij}\|}$ and $\odot$ means element-wise multiplication.

Then layer normalization is added in a residual way to finally obtain the self-attention local embedding matrix
$\hat{\mathcal{G}}^i = \mathcal{G}^{i,L_a}$ after L_a attention layers:[^1]

$$\mathcal{G}^{i,l} = \mathcal{G}^{i,l-1} + \mathrm{LayerNorm}(A(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{V}^{i,l}, \mathcal{R}^{i,l})).$$

Parameters

rcut: float
: The cut-off radius r_c

rcut_smth: float
: From where the environment matrix should be smoothed r_s

sel
: [`list`[`int`], `int`] list[int]: sel[i] specifies the maxmum number of type i atoms in the cut-off radius int: the total maxmum number of atoms in the cut-off radius

ntypes
: [`int`] Number of element types

neuron
: [`list`[`int`]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron: int
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

tebd_dim: int
: Dimension of the type embedding

tebd_input_mode: str
: The input mode of the type embedding. Supported modes are [“concat”, “strip”]. - “concat”: Concatenate the type embedding with the smoothed radial information as the union input for the embedding network. - “strip”: Use a separated embedding network for the type embedding and combine the output with the radial embedding network output.

resnet_dt: bool
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable: bool
: If the weights of this descriptors are trainable.

trainable_ln: bool
: Whether to use trainable shift and scale weights in layer normalization.

ln_eps: float, Optional
The epsilon value for layer normalization.

type_one_side: bool
If 'False', type embeddings of both neighbor and central atoms are considered. If 'True', only type embeddings of neighbor atoms are considered. Default is 'False'.

attn: int
Hidden dimension of the attention vectors

attn_layer: int
Number of attention layers

attn_dotr: bool
If dot the angular gate to the attention weights

attn_mask: bool
(Only support False to keep consistent with other backend references.) (Not used in this version. True option is not implemented.) If mask the diagonal of attention weights

exclude_types
[List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

env_protection: float
Protection parameter to prevent division by zero errors during environment matrix calculations.

set_davg_zero: bool
Set the shift of embedding net input to zero.

activation_function: str
The activation function in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision: str
The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

scaling_factor: float
The scaling factor of normalization in calculations of attention weights. If temperature is None, the scaling of attention weights is (N_dim * scaling_factor)**0.5

normalize: bool
Whether to normalize the hidden vectors in attention weights calculation.

temperature: float
If not None, the scaling of attention weights is temperature itself.

smooth_type_embedding: bool
Whether to use smooth process in attention weights calculation.

concat_output_tebd: bool
Whether to concat type embedding at the output of the descriptor.

stripped_type_embedding: bool, Optional
(Deprecated, kept only for compatibility.) Whether to strip the type embedding into a separate embedding network. Setting this parameter to True is equivalent to setting tebd_input_mode to 'strip'. Setting it to False is equivalent to setting tebd_input_mode to 'concat'. The default value is None, which means the tebd_input_mode setting will be used instead.

spin
(Only support None to keep consistent with other backend references.) (Not used in this version. Not-none option is not implemented.) The old implementation of deepspin.

References

[1]

property dim_out

property dim_emb

get_rcut() → float
Returns the cut-off radius.

get_nsel() → int
Returns the number of selected atoms in the cut-off radius.

get_sel() → List[int]
Returns the number of selected atoms for each type.

get_ntypes() → int
Returns the number of element types.

get_dim_out() → int
Returns the output dimension.

get_dim_emb() → int
Returns the embedding dimension of g2.

mixed_types() → bool
If true, the discriptor 1. assumes total number of atoms aligned across frames; 2. requires a neighbor list that does not distinguish different atomic types.

If false, the discriptor 1. assumes total number of atoms of each atom type aligned across frames; 2. requires a neighbor list that distinguishes different atomic types.

share_params(base_class, shared_level, resume=False)
Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)
Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

•List[dict]: A list of data samples from various data systems.
Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.

- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

set_stat_mean_and_stddev(mean: torch.Tensor, stddev: torch.Tensor) → None

serialize() → dict

Serialize the obj to dict.

classmethod deserialize(data: dict) → DescrptDPA1

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

BD
The deserialized descriptor

forward(extended_coord: torch.Tensor, extended_atype: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None)

Compute the descriptor.

Parameters

extended_coord
The extended coordinates of atoms. shape: nf x (nallx3)

extended_atype
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping, not required by this descriptor.

comm_dict
The data needed for communication for parallel inference.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. shape: nf x nloc x nnei x ng

h2
The rotationally equivariant pair-partical representation. shape: nf x nloc x nnei x 3

sw
The smooth switch function. shape: nf x nloc x nnei

classmethod update_sel(global_jdata: dict, local_jdata: dict)
Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

deepmd.pt.model.descriptor.dpa2

Module Contents

Classes

DescrptDPA2	Base class for all neural network modules.

class deepmd.pt.model.descriptor.dpa2.DescrptDPA2(ntypes: int, repinit_rcut: float, repinit_rcut_smth: float, repinit_nsel: int, repformer_rcut: float, repformer_rcut_smth: float, repformer_nsel: int, tebd_dim: int = 8, concat_output_tebd: bool = True, repinit_neuron: List[int] = [25, 50, 100], repinit_axis_neuron: int = 16, repinit_set_davg_zero: bool = True, repinit_activation='tanh', repformer_nlayers: int = 3, repformer_g1_dim: int = 128, repformer_g2_dim: int = 16, repformer_axis_dim: int = 4, repformer_do_bn_mode: str = 'no', repformer_bn_momentum: float = 0.1, repformer_update_g1_has_conv: bool = True, repformer_update_g1_has_drrd: bool = True, repformer_update_g1_has_grrg: bool = True, repformer_update_g1_has_attn: bool = True, repformer_update_g2_has_g1g1: bool = True, repformer_update_g2_has_attn: bool = True, repformer_update_h2: bool = False, repformer_attn1_hidden: int = 64, repformer_attn1_nhead: int = 4, repformer_attn2_hidden: int = 16, repformer_attn2_nhead: int = 4, repformer_attn2_has_gate: bool = False, repformer_activation: str = 'tanh', repformer_update_style: str = 'res_avg', repformer_set_davg_zero: bool = True, repformer_add_type_ebd_to_seq: bool = False, env_protection: float = 0.0, trainable: bool = True, exclude_types: List[Tuple[int, int]] = [], type: str | None = None, rcut: float | None = None, rcut_smth: float | None = None, sel: int | None = None)

Bases: `torch.nn.Module`, *deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

property dim_out

property dim_emb

Returns the embedding dimension g2.

get_rcut() → float

Returns the cut-off radius.

get_nsel() → int

Returns the number of selected atoms in the cut-off radius.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_ntypes() → int

Returns the number of element types.

get_dim_out() → int

Returns the output dimension of this descriptor.

get_dim_emb() → int

Returns the embedding dimension of this descriptor.

mixed_types() → bool

If true, the discriptor 1. assumes total number of atoms aligned across frames; 2. requires a neighbor list that does not distinguish different atomic types.

If false, the discriptor 1. assumes total number of atoms of each atom type aligned across frames; 2. requires a neighbor list that distinguishes different atomic types.

share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)

Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

•List[dict]: A list of data samples from various data systems.
Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.

- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

abstract serialize() → dict

Serialize the obj to dict.

abstract classmethod deserialize() → DescrptDPA2

Deserialize from a dict.

forward(extended_coord: torch.Tensor, extended_atype: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None)

Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping, mapps extended region index to local region.

comm_dict
The data needed for communication for parallel inference.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. shape: nf x nloc x nnei x ng

h2
The rotationally equivariant pair-partical representation. shape: nf x nloc x nnei x 3

sw
The smooth switch function. shape: nf x nloc x nnei

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

deepmd.pt.model.descriptor.env_mat

Module Contents

Functions

_make_env_mat(nlist, coord, rcut, ruct_smth[, ...])	Make smooth environment matrix.
prod_env_mat(extended_coord, nlist, atype, mean, ...)	Generate smooth environment matrix from atom coordinates and other context.

deepmd.pt.model.descriptor.env_mat._make_env_mat(nlist, coord, rcut: float, ruct_smth: float, radial_only: bool = False, protection: float = 0.0)

Make smooth environment matrix.

deepmd.pt.model.descriptor.env_mat.prod_env_mat(extended_coord, nlist, atype, mean, stddev, rcut: float, rcut_smth: float, radial_only: bool = False, protection: float = 0.0)

Generate smooth environment matrix from atom coordinates and other context.

Args: - extended_coord: Copied atom coordinates with shape [nframes, nall*3]. - atype: Atom types with shape [nframes, nloc]. - mean: Average value of descriptor per element type with shape [len(sec), nnei, 4 or 1]. - stddev: Standard deviation of descriptor per element type with shape [len(sec), nnei, 4 or 1]. - rcut: Cut-off radius. - rcut_smth: Smooth hyper-parameter for pair force & energy. - radial_only: Whether to return a full description or a radial-only descriptor. - protection: Protection parameter to prevent division by zero errors during calculations.

Returns

- env_mat: **Shape is** [**nframes**, **natoms**[1]*nnei*4].

deepmd.pt.model.descriptor.gaussian_lcc

Module Contents

Classes

DescrptGaussianLcc	Base class for all neural network modules.

class deepmd.pt.model.descriptor.gaussian_lcc.DescrptGaussianLcc(rcut, rcut_smth, sel: int, ntypes: int, num_pair: int, embed_dim: int = 768, kernel_num: int = 128, pair_embed_dim: int = 64, num_block: int = 1, layer_num: int = 12, attn_head: int = 48, pair_hidden_dim: int = 16, ffn_embedding_dim: int = 768, dropout: float = 0.0, droppath_prob: float = 0.1, pair_dropout: float = 0.25, attention_dropout: float = 0.1, activation_dropout: float = 0.1, pre_ln: bool = True, do_tag_embedding: bool = False, tag_ener_pref: bool = False, atomic_sum_gbf: bool = False, pre_add_seq: bool = True, tri_update: bool = True, **kwargs)

Bases: torch.nn.Module, *deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

property dim_out

Returns the output dimension of atomic representation.

property dim_in

Returns the atomic input dimension of this descriptor.

property dim_emb

Returns the output dimension of pair representation.

compute_input_stats(merged: List[dict], path: deepmd.utils.path.DPPath | None = None)

Update mean and stddev for descriptor elements.

forward(extended_coord, nlist, atype, nlist_type, nlist_loc=None, atype_tebd=None, nlist_tebd=None, seq_input=None)

Calculate the atomic and pair representations of this descriptor.

Args: - extended_coord: Copied atom coordinates with shape [nframes, nall, 3]. - nlist: Neighbor list with shape [nframes, nloc, nnei]. - atype: Atom type with shape [nframes, nloc]. - nlist_type: Atom type of neighbors with shape [nframes, nloc, nnei]. - nlist_loc: Local index of neighbor list with shape [nframes, nloc, nnei]. - atype_tebd: Atomic type embedding with shape [nframes, nloc, tebd_dim]. - nlist_tebd: Type embeddings of neighbor with shape [nframes, nloc, nnei, tebd_dim]. - seq_input: The sequential input from other descriptor with

shape [nframes, nloc, tebd_dim] or [nframes * nloc, 1 + nnei, tebd_dim]

Returns

- result: **descriptor with shape** [**nframes**, **nloc**, **self.filter_neuron**[-1] * **self.axis_neuron**].
- ret: **environment matrix with shape** [**nframes**, **nloc**, **self.neei**, **out_size**]

deepmd.pt.model.descriptor.hybrid

Module Contents

Classes

DescrptHybrid	Concate a list of descriptors to form a new descriptor.

class deepmd.pt.model.descriptor.hybrid.DescrptHybrid(list: List[deepmd.pt.model.descriptor.base_descriptor.BaseDescr | Dict[str, Any]], **kwargs)

Bases: *deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor*, torch.nn.Module

Concate a list of descriptors to form a new descriptor.

Parameters

list

[list][List[Union[BaseDescriptor, Dict[str, *Any*]]]] Build a descriptor from the concatenation of the list of descriptors. The descriptor can be either an object or a dictionary.

get_rcut() → float

Returns the cut-off radius.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_ntypes() → int

Returns the number of element types.

get_dim_out() → int

Returns the output dimension.

get_dim_emb() → int

Returns the output dimension.

mixed_types()

Returns if the descriptor requires a neighbor list that distinguish different atomic types or not.

share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

compute_input_stats(merged: List[dict], path: deepmd.utils.path.DPPath | None = None)

Update mean and stddev for descriptor elements.

forward(coord_ext: torch.Tensor, atype_ext: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None)

Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping, not required by this descriptor.

comm_dict
The data needed for communication for parallel inference.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3. This descriptor returns None

g2
The rotationally invariant pair-partical representation. this descriptor returns None

h2
The rotationally equivariant pair-partical representation. this descriptor returns None

sw
The smooth switch function. this descriptor returns None

classmethod update_sel(global_jdata: dict, local_jdata: dict) → dict

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

serialize() → dict

Serialize the obj to dict.

classmethod deserialize(data: dict) → DescrptHybrid

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

BD
The deserialized descriptor

deepmd.pt.model.descriptor.repformer_layer

Module Contents

Classes

Atten2Map	Base class for all neural network modules.
Atten2MultiHeadApply	Base class for all neural network modules.
Atten2EquiVarApply	Base class for all neural network modules.
LocalAtten	Base class for all neural network modules.
RepformerLayer	Base class for all neural network modules.

Functions

torch_linear(*args, **kwargs)	
_make_nei_g1(→ torch.Tensor)	
_apply_nlist_mask(→ torch.Tensor)	
_apply_switch(→ torch.Tensor)	
_apply_h_norm(→ torch.Tensor)	Normalize h by the std of vector length.

deepmd.pt.model.descriptor.repformer_layer.**torch_linear**(*args, **kwargs)

deepmd.pt.model.descriptor.repformer_layer.**_make_nei_g1**(g1_ext: torch.Tensor, nlist: torch.Tensor) → torch.Tensor

deepmd.pt.model.descriptor.repformer_layer.**_apply_nlist_mask**(gg: torch.Tensor, nlist_mask: torch.Tensor) → torch.Tensor

deepmd.pt.model.descriptor.repformer_layer.**_apply_switch**(gg: torch.Tensor, sw: torch.Tensor) → torch.Tensor

deepmd.pt.model.descriptor.repformer_layer.**_apply_h_norm**(hh: torch.Tensor) → torch.Tensor

Normalize h by the std of vector length. do not have an idea if this is a good way.

class deepmd.pt.model.descriptor.repformer_layer.**Atten2Map**(ni: int, nd: int, nh: int, has_gate: bool = False, smooth: bool = True, attnw_shift: float = 20.0)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(g2: torch.Tensor, h2: torch.Tensor, nlist_mask: torch.Tensor, sw: torch.Tensor) → torch.Tensor

class deepmd.pt.model.descriptor.repformer_layer.**Atten2MultiHeadApply**(ni: int, nh: int)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(AA: torch.Tensor, g2: torch.Tensor) → torch.Tensor

class deepmd.pt.model.descriptor.repformer_layer.**Atten2EquiVarApply**(ni: int, nh: int)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(AA: torch.Tensor, h2: torch.Tensor) → torch.Tensor

class deepmd.pt.model.descriptor.repformer_layer.**LocalAtten**(ni: int, nd: int, nh: int, smooth: bool = True, attnw_shift: float = 20.0)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assign-

ment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(g1: torch.Tensor, gg1: torch.Tensor, nlist_mask: torch.Tensor, sw: torch.Tensor) → torch.Tensor

class deepmd.pt.model.descriptor.repformer_layer.RepformerLayer(rcut, rcut_smth, sel: int, ntypes: int, g1_dim=128, g2_dim=16, axis_dim: int = 4, update_chnnl_2: bool = True, do_bn_mode: str = 'no', bn_momentum: float = 0.1, update_g1_has_conv: bool = True, update_g1_has_drrd: bool = True, update_g1_has_grrg: bool = True, update_g1_has_attn: bool = True, update_g2_has_g1g1: bool = True, update_g2_has_attn: bool = True, update_h2: bool = False, attn1_hidden: int = 64, attn1_nhead: int = 4, attn2_hidden: int = 16, attn2_nhead: int = 4, attn2_has_gate: bool = False, activation_function: str = 'tanh', update_style: str = 'res_avg', set_davg_zero: bool = True, smooth: bool = True)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

cal_1_dim(g1d: int, g2d: int, ax: int) → int

_update_h2(g2: torch.Tensor, h2: torch.Tensor, nlist_mask: torch.Tensor, sw: torch.Tensor) → torch.Tensor

_update_g1_conv(gg1: torch.Tensor, g2: torch.Tensor, nlist_mask: torch.Tensor, sw: torch.Tensor) → torch.Tensor

_cal_h2g2(g2: torch.Tensor, h2: torch.Tensor, nlist_mask: torch.Tensor, sw: torch.Tensor) → torch.Tensor

_cal_grrg(h2g2: torch.Tensor) → torch.Tensor

_update_g1_grrg(g2: torch.Tensor, h2: torch.Tensor, nlist_mask: torch.Tensor, sw: torch.Tensor) → torch.Tensor

_update_g2_g1g1(g1: torch.Tensor, gg1: torch.Tensor, nlist_mask: torch.Tensor, sw: torch.Tensor) → torch.Tensor

_apply_bn(bn_number: int, gg: torch.Tensor)

_apply_nb_1(bn_number: int, gg: torch.Tensor) → torch.Tensor

_apply_nb_2(bn_number: int, gg: torch.Tensor) → torch.Tensor

_apply_bn_uni(bn_number: int, gg: torch.Tensor, mode: str = '1') → torch.Tensor

_apply_bn_comp(bn_number: int, gg: torch.Tensor) → torch.Tensor

forward(g1_ext: torch.Tensor, g2: torch.Tensor, h2: torch.Tensor, nlist: torch.Tensor, nlist_mask: torch.Tensor, sw: torch.Tensor)

Parameters

g1_ext
[nf x nall x ng1 extended single-atom chanel]

g2
[nf x nloc x nnei x ng2 pair-atom channel, invariant]

h2
[nf x nloc x nnei x 3 pair-atom channel, equivariant]

nlist
[nf x nloc x nnei neighbor list (padded neis are set to 0)]

nlist_mask
[nf x nloc x nnei masks of the neighbor list. real nei 1 otherwise 0]

sw
[nf x nloc x nnei switch function]

Returns

g1: **nf x nloc x ng1 updated** single-atom **chanel**
g2: **nf x nloc x nnei x ng2 updated** pair-atom **channel**, **invariant**
h2: **nf x nloc x nnei x** 3 **updated** pair-atom **channel**, **equivariant**

list_update_res_avg(update_list: List[torch.Tensor]) → torch.Tensor

list_update_res_incr(update_list: List[torch.Tensor]) → torch.Tensor

list_update(update_list: List[torch.Tensor]) → torch.Tensor

_bn_layer(nf: int = 1) → Callable

deepmd.pt.model.descriptor.repformers

Module Contents

Classes

DescrptBlockRepformers	The building block of descriptor.

Functions

torch_linear(*args, **kwargs)
border_op(→ torch.Tensor)

Attributes

mydtype
mydev
simple_linear
mylinear

deepmd.pt.model.descriptor.repformers.**mydtype**

deepmd.pt.model.descriptor.repformers.**mydev**

deepmd.pt.model.descriptor.repformers.**torch_linear**(*args, **kwargs)

deepmd.pt.model.descriptor.repformers.**simple_linear**

deepmd.pt.model.descriptor.repformers.**mylinear**

deepmd.pt.model.descriptor.repformers.**border_op**(argument0, argument1, argument2, argument3, argument4, argument5, argument6, argument7, argument8) → torch.Tensor

class deepmd.pt.model.descriptor.repformers.**DescrptBlockRepformers**(rcut, rcut_smth, sel: int, ntypes: int, nlayers: int = 3, g1_dim=128, g2_dim=16, axis_dim: int = 4, direct_dist: bool = False, do_bn_mode: str = 'no', bn_momentum: float = 0.1, update_g1_has_conv: bool = True, update_g1_has_drrd: bool = True, update_g1_has_grrg: bool = True, update_g1_has_attn: bool = True, update_g2_has_g1g1: bool = True, update_g2_has_attn: bool = True, update_h2: bool = False, attn1_hidden: int = 64, attn1_nhead: int = 4, attn2_hidden: int = 16, attn2_nhead: int = 4, attn2_has_gate: bool = False, activation_function: str = 'tanh', update_style: str = 'res_avg', set_davg_zero: bool = True, smooth: bool = True, add_type_ebd_to_seq: bool = False, exclude_types: List[Tuple[int, int]] = [], env_protection: float = 0.0, type: str | None = None)

Bases: *deepmd.pt.model.descriptor.descriptor.DescriptorBlock*

The building block of descriptor. Given the input descriptor, provide with the atomic coordinates, atomic types and neighbor list, calculate the new descriptor.

property dim_out

Returns the output dimension of this descriptor.

property dim_in

Returns the atomic input dimension of this descriptor.

property dim_emb

Returns the embedding dimension g2.

get_rcut() → float

Returns the cut-off radius.

get_nsel() → int

Returns the number of selected atoms in the cut-off radius.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_ntypes() → int

Returns the number of element types.

get_dim_out() → int

Returns the output dimension.

get_dim_in() → int

Returns the input dimension.

get_dim_emb() → int

Returns the embedding dimension g2.

mixed_types() → bool

If true, the discriptor 1. assumes total number of atoms aligned across frames; 2. requires a neighbor list that does not distinguish different atomic types.

If false, the discriptor 1. assumes total number of atoms of each atom type aligned across frames; 2. requires a neighbor list that distinguishes different atomic types.

reinit_exclude(exclude_types: List[Tuple[int, int]] = [])

forward(nlist: torch.Tensor, extended_coord: torch.Tensor, extended_atype: torch.Tensor, extended_atype_embd: torch.Tensor | None = None, mapping: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None)

Calculate DescriptorBlock.

compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)

Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

get_stats() → Dict[str, deepmd.utils.env_mat_stat.StatItem]
Get the statistics of the descriptor.

deepmd.pt.model.descriptor.se_a

Module Contents

Classes

DescrptSeA	Base descriptor provides the interfaces of descriptor.
DescrptBlockSeA	The building block of descriptor.

class deepmd.pt.model.descriptor.se_a.**DescrptSeA**(rcut, rcut_smth, sel, neuron=[25, 50, 100], axis_neuron=16, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'float64', resnet_dt: bool = False, exclude_types: List[Tuple[int, int]] = [], env_protection: float = 0.0, old_impl: bool = False, type_one_side: bool = True, **kwargs)

Bases: *deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor*, torch.nn.Module

Base descriptor provides the interfaces of descriptor.

property dim_out
Returns the output dimension of this descriptor.

get_rcut() → float
Returns the cut-off radius.

get_nsel() → int
Returns the number of selected atoms in the cut-off radius.

get_sel() → List[int]
Returns the number of selected atoms for each type.

get_ntypes() → int
Returns the number of element types.

get_dim_out() → int
Returns the output dimension.

get_dim_emb() → int
Returns the output dimension.

mixed_types()
Returns if the descriptor requires a neighbor list that distinguish different atomic types or not.

share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)

Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

reinit_exclude(exclude_types: List[Tuple[int, int]] = [])

Update the type exclusions.

forward(coord_ext: torch.Tensor, atype_ext: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None)

Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping, not required by this descriptor.

comm_dict
The data needed for communication for parallel inference.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. this descriptor returns None

h2
The rotationally equivariant pair-partical representation. this descriptor returns None

sw
The smooth switch function.

set_stat_mean_and_stddev(mean: torch.Tensor, stddev: torch.Tensor) → None

serialize() → dict
Serialize the obj to dict.

classmethod deserialize(data: dict) → DescrptSeA
Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

BD
The deserialized descriptor

classmethod update_sel(global_jdata: dict, local_jdata: dict)
Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

class deepmd.pt.model.descriptor.se_a.DescrptBlockSeA(rcut, rcut_smth, sel, neuron=[25, 50, 100], axis_neuron=16, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'float64', resnet_dt: bool = False, exclude_types: List[Tuple[int, int]] = [], env_protection: float = 0.0, old_impl: bool = False, type_one_side: bool = True, trainable: bool = True, **kwargs)

Bases: *deepmd.pt.model.descriptor.DescriptorBlock*

The building block of descriptor. Given the input descriptor, provide with the atomic coordinates, atomic types and neighbor list, calculate the new descriptor.

property dim_out
Returns the output dimension of this descriptor.

property dim_in
Returns the atomic input dimension of this descriptor.

ndescrpt: Final[int]

__constants__: ClassVar[list] = ['ndescrpt']

get_rcut() → float
Returns the cut-off radius.

get_nsel() → int

Returns the number of selected atoms in the cut-off radius.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_ntypes() → int

Returns the number of element types.

get_dim_out() → int

Returns the output dimension.

get_dim_emb() → int

Returns the output dimension.

get_dim_in() → int

Returns the input dimension.

mixed_types() → bool

If true, the discriptor 1. assumes total number of atoms aligned across frames; 2. requires a neighbor list that does not distinguish different atomic types.

If false, the discriptor 1. assumes total number of atoms of each atom type aligned across frames; 2. requires a neighbor list that distinguishes different atomic types.

__setitem__(key, value)

__getitem__(key)

compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)

Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

get_stats() → Dict[str, deepmd.utils.env_mat_stat.StatItem]

Get the statistics of the descriptor.

reinit_exclude(exclude_types: List[Tuple[int, int]] = [])

forward(nlist: torch.Tensor, extended_coord: torch.Tensor, extended_atype: torch.Tensor, extended_atype_embd: torch.Tensor | None = None, mapping: torch.Tensor | None = None)

Calculate decoded embedding for each atom.

Args: - coord: Tell atom coordinates with shape [nframes, natoms[1]*3]. - atype: Tell atom types with shape [nframes, natoms[1]]. - natoms: Tell atom count and element count. Its shape is [2+self.ntypes]. - box: Tell simulation box with shape [nframes, 9].

Returns

- torch.Tensor: **descriptor matrix with shape** [**nframes**, **natoms**[0]*self.filter_neuron[-1]*self.axis_neuron].

deepmd.pt.model.descriptor.se_atten

Module Contents

Classes

DescrptBlockSeAtten	The building block of descriptor.
NeighborGatedAttention	Base class for all neural network modules.
NeighborGatedAttentionLayer	Base class for all neural network modules.
GatedAttentionLayer	Base class for all neural network modules.

class deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtten(rcut: float, rcut_smth: float, sel: List[int] | int, ntypes: int, neuron: list = [25, 50, 100], axis_neuron: int = 16, tebd_dim: int = 8, tebd_input_mode: str = 'concat', set_davg_zero: bool = True, attn: int = 128, attn_layer: int = 2, attn_dotr: bool = True, attn_mask: bool = False, activation_function='tanh', precision: str = 'float64', resnet_dt: bool = False, scaling_factor=1.0, normalize=True, temperature=None, smooth: bool = True, type_one_side: bool = False, exclude_types: List[Tuple[int, int]] = [], env_protection: float = 0.0, trainable_ln: bool = True, ln_eps: float | None = 1e-05, type: str | None = None, old_impl: bool = False)

Bases: *deepmd.pt.model.descriptor.descriptor.DescriptorBlock*

The building block of descriptor. Given the input descriptor, provide with the atomic coordinates, atomic types and neighbor list, calculate the new descriptor.

property dim_out

Returns the output dimension of this descriptor.

property dim_in

Returns the atomic input dimension of this descriptor.

property dim_emb

Returns the output dimension of embedding.

get_rcut() → float

Returns the cut-off radius.

get_nsel() → int

Returns the number of selected atoms in the cut-off radius.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_ntypes() → int

Returns the number of element types.

get_dim_in() → int

Returns the output dimension.

get_dim_out() → int

Returns the output dimension.

get_dim_emb() → int

Returns the output dimension of embedding.

__setitem__(key, value)

__getitem__(key)

mixed_types() → bool

If true, the discriptor 1. assumes total number of atoms aligned across frames; 2. requires a neighbor list that does not distinguish different atomic types.

If false, the discriptor 1. assumes total number of atoms of each atom type aligned across frames; 2. requires a neighbor list that distinguishes different atomic types.

compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)

Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

get_stats() → Dict[str, deepmd.utils.env_mat_stat.StatItem]

Get the statistics of the descriptor.

reinit_exclude(exclude_types: List[Tuple[int, int]] = [])

forward(nlist: torch.Tensor, extended_coord: torch.Tensor, extended_atype: torch.Tensor, extended_atype_embd: torch.Tensor | None = None, mapping: torch.Tensor | None = None)

Compute the descriptor.

Parameters

nlist
The neighbor list. shape: nf x nloc x nnei

extended_coord
The extended coordinates of atoms. shape: nf x (nallx3)

extended_atype
The extended aotm types. shape: nf x nall x nt

extended_atype_embd
The extended type embedding of atoms. shape: nf x nall

mapping
The index mapping, not required by this descriptor.

Returns

result
The descriptor. shape: nf x nloc x (ng x axis_neuron)

g2
The rotationally invariant pair-partical representation. shape: nf x nloc x nnei x ng

h2
The rotationally equivariant pair-partical representation. shape: nf x nloc x nnei x 3

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

sw
The smooth switch function. shape: nf x nloc x nnei

class deepmd.pt.model.descriptor.se_atten.NeighborGatedAttention(layer_num: int, nnei: int, embed_dim: int, hidden_dim: int, dotr: bool = False, do_mask: bool = False, scaling_factor: float = 1.0, normalize: bool = True, temperature: float | None = None, trainable_ln: bool = True, ln_eps: float = 1e-05, smooth: bool = True, precision: str = DEFAULT_PRECISION)

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
: **training** (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(input_G, nei_mask, input_r: torch.Tensor | None = None, sw: torch.Tensor | None = None)

Compute the multi-layer gated self-attention.

Parameters

input_G
: inputs with shape: (nf x nloc) x nnei x embed_dim.

nei_mask
: neighbor mask, with paddings being 0. shape: (nf x nloc) x nnei.

input_r
: normalized radial. shape: (nf x nloc) x nnei x 3.

sw
: The smooth switch function. shape: nf x nloc x nnei

__getitem__(key)

__setitem__(key, value)

serialize() → dict

Serialize the networks to a dict.

Returns

dict
: The serialized networks.

classmethod deserialize(data: dict) → NeighborGatedAttention

Deserialize the networks from a dict.

Parameters

data
[dict] The dict to deserialize from.

class deepmd.pt.model.descriptor.se_atten.NeighborGatedAttentionLayer(nnei: int, embed_dim: int, hidden_dim: int, dotr: bool = False, do_mask: bool = False, scaling_factor: float = 1.0, normalize: bool = True, temperature: float | None = None, smooth: bool = True, trainable_ln: bool = True, ln_eps: float = 1e-05, precision: str = DEFAULT_PRECISION)

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call to(), etc.

Note: As per the example above, an __init__() call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(x, nei_mask, input_r: torch.Tensor | None = None, sw: torch.Tensor | None = None)

serialize() → dict

Serialize the networks to a dict.

Returns

dict
The serialized networks.

classmethod deserialize(data: dict) → NeighborGatedAttentionLayer

Deserialize the networks from a dict.

Parameters

data
[dict] The dict to deserialize from.

class deepmd.pt.model.descriptor.se_atten.GatedAttentionLayer(nnei: int, embed_dim: int, hidden_dim: int, dotr: bool = False, do_mask: bool = False, scaling_factor: float = 1.0, normalize: bool = True, temperature: float | None = None, bias: bool = True, smooth: bool = True, precision: str = DEFAULT_PRECISION)

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call to(), etc.

Note: As per the example above, an __init__() call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(query, nei_mask, input_r: torch.Tensor | None = None, sw: torch.Tensor | None = None, attnw_shift: float = 20.0)

Compute the gated self-attention.

Parameters

query
inputs with shape: (nf x nloc) x nnei x embed_dim.

nei_mask
neighbor mask, with paddings being 0. shape: (nf x nloc) x nnei.

input_r
normalized radial. shape: (nf x nloc) x nnei x 3.

sw
The smooth switch function. shape: (nf x nloc) x nnei

attnw_shift
[float] The attention weight shift to preserve smoothness when doing padding before softmax.

serialize() → dict

Serialize the networks to a dict.

Returns

dict
The serialized networks.

classmethod deserialize(data: dict) → GatedAttentionLayer

Deserialize the networks from a dict.

Parameters

data
[dict] The dict to deserialize from.

deepmd.pt.model.descriptor.se_r

Module Contents

Classes

DescrptSeR	Base descriptor provides the interfaces of descriptor.

class deepmd.pt.model.descriptor.se_r.DescrptSeR(rcut, rcut_smth, sel, neuron=[25, 50, 100], set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'float64', resnet_dt: bool = False, exclude_types: List[Tuple[int, int]] = [], env_protection: float = 0.0, old_impl: bool = False, trainable: bool = True, **kwargs)

Bases: *deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor*, torch.nn.Module

Base descriptor provides the interfaces of descriptor.

get_rcut() → float

Returns the cut-off radius.

get_nsel() → int

Returns the number of selected atoms in the cut-off radius.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_ntypes() → int

Returns the number of element types.

get_dim_out() → int

Returns the output dimension.

abstract get_dim_emb() → int

Returns the output dimension.

get_dim_in() → int

Returns the input dimension.

mixed_types() → bool

If true, the discriptor 1. assumes total number of atoms aligned across frames; 2. requires a neighbor list that does not distinguish different atomic types.

If false, the discriptor 1. assumes total number of atoms of each atom type aligned across frames; 2. requires a neighbor list that distinguishes different atomic types.

share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)

Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

get_stats() → Dict[str, deepmd.utils.env_mat_stat.StatItem]

Get the statistics of the descriptor.

__setitem__(key, value)

__getitem__(key)

reinit_exclude(exclude_types: List[Tuple[int, int]] = [])

forward(coord_ext: torch.Tensor, atype_ext: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None)

Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping, not required by this descriptor.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. this descriptor returns None

h2
The rotationally equivariant pair-partical representation. this descriptor returns None

sw
The smooth switch function.

set_stat_mean_and_stddev(mean: torch.Tensor, stddev: torch.Tensor) → None

serialize() → dict

Serialize the obj to dict.

classmethod deserialize(data: dict) → DescrptSeR

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

BD
The deserialized descriptor

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

Package Contents

Classes

DescriptorBlock	The building block of descriptor.
DescrptBlockSeAtten	The building block of descriptor.
DescrptDPA1	Attention-based descriptor which is proposed in the pretrainable DPA-1[1] model.
DescrptDPA2	Base class for all neural network modules.
DescrptGaussianLcc	Base class for all neural network modules.
DescrptHybrid	Concate a list of descriptors to form a new descriptor.
DescrptBlockRepformers	The building block of descriptor.
DescrptBlockSeA	The building block of descriptor.
DescrptSeA	Base descriptor provides the interfaces of descriptor.
DescrptSeR	Base descriptor provides the interfaces of descriptor.

Functions

make_default_type_embedding(ntypes)	
prod_env_mat(extended_coord, nlist, atype, mean, ...)	Generate smooth environment matrix from atom coordinates and other context.

Attributes

BaseDescriptor	

deepmd.pt.model.descriptor.BaseDescriptor

class deepmd.pt.model.descriptor.DescriptorBlock(*args, **kwargs)

Bases: torch.nn.Module, abc.ABC, make_plugin_registry('DescriptorBlock')

The building block of descriptor. Given the input descriptor, provide with the atomic coordinates, atomic types and neighbor list, calculate the new descriptor.

local_cluster = False

abstract get_rcut() → float

Returns the cut-off radius.

abstract get_nsel() → int

Returns the number of selected atoms in the cut-off radius.

abstract get_sel() → List[int]

Returns the number of selected atoms for each type.

abstract get_ntypes() → int

Returns the number of element types.

abstract get_dim_out() → int

Returns the output dimension.

abstract get_dim_in() → int

Returns the output dimension.

abstract get_dim_emb() → int

Returns the embedding dimension.

abstract compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)

Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

abstract get_stats() → Dict[str, deepmd.utils.env_mat_stat.StatItem]

Get the statistics of the descriptor.

share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

abstract forward(nlist: torch.Tensor, extended_coord: torch.Tensor, extended_atype: torch.Tensor, extended_atype_embd: torch.Tensor | None = None, mapping: torch.Tensor | None = None)

Calculate DescriptorBlock.

deepmd.pt.model.descriptor.**make_default_type_embedding**(ntypes)

class deepmd.pt.model.descriptor.**DescrptBlockSeAtten**(rcut: float, rcut_smth: float, sel: List[int] | int, ntypes: int, neuron: list = [25, 50, 100], axis_neuron: int = 16, tebd_dim: int = 8, tebd_input_mode: str = 'concat', set_davg_zero: bool = True, attn: int = 128, attn_layer: int = 2, attn_dotr: bool = True, attn_mask: bool = False, activation_function='tanh', precision: str = 'float64', resnet_dt: bool = False, scaling_factor=1.0, normalize=True, temperature=None, smooth: bool = True, type_one_side: bool = False, exclude_types: List[Tuple[int, int]] = [], env_protection: float = 0.0, trainable_ln: bool = True, ln_eps: float | None = 1e-05, type: str | None = None, old_impl: bool = False)

Bases: *deepmd.pt.model.descriptor.descriptor.DescriptorBlock*

The building block of descriptor. Given the input descriptor, provide with the atomic coordinates, atomic types and neighbor list, calculate the new descriptor.

property dim_out

Returns the output dimension of this descriptor.

property dim_in

Returns the atomic input dimension of this descriptor.

property dim_emb

Returns the output dimension of embedding.

get_rcut() → float

Returns the cut-off radius.

get_nsel() → int

Returns the number of selected atoms in the cut-off radius.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_ntypes() → int

Returns the number of element types.

get_dim_in() → int

Returns the output dimension.

get_dim_out() → int

Returns the output dimension.

get_dim_emb() → int

Returns the output dimension of embedding.

__setitem__(key, value)

__getitem__(key)

mixed_types() → bool

If true, the discriptor 1. assumes total number of atoms aligned across frames; 2. requires a neighbor list that does not distinguish different atomic types.

If false, the discriptor 1. assumes total number of atoms of each atom type aligned across frames; 2. requires a neighbor list that distinguishes different atomic types.

compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)

Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

get_stats() → Dict[str, deepmd.utils.env_mat_stat.StatItem]

Get the statistics of the descriptor.

reinit_exclude(exclude_types: List[Tuple[int, int]] = [])

forward(nlist: torch.Tensor, extended_coord: torch.Tensor, extended_atype: torch.Tensor, extended_atype_embd: torch.Tensor | None = None, mapping: torch.Tensor | None = None)

Compute the descriptor.

Parameters

nlist
The neighbor list. shape: nf x nloc x nnei

extended_coord
The extended coordinates of atoms. shape: nf x (nallx3)

extended_atype
The extended aotm types. shape: nf x nall x nt

extended_atype_embd
The extended type embedding of atoms. shape: nf x nall

mapping
The index mapping, not required by this descriptor.

Returns

result
The descriptor. shape: nf x nloc x (ng x axis_neuron)

g2
The rotationally invariant pair-partical representation. shape: nf x nloc x nnei x ng

h2
The rotationally equivariant pair-partical representation. shape: nf x nloc x nnei x 3

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

sw
The smooth switch function. shape: nf x nloc x nnei

class deepmd.pt.model.descriptor.DescrptDPA1(rcut: float, rcut_smth: float, sel: List[int] | int, ntypes: int, neuron: list = [25, 50, 100], axis_neuron: int = 16, tebd_dim: int = 8, tebd_input_mode: str = 'concat', set_davg_zero: bool = True, attn: int = 128, attn_layer: int = 2, attn_dotr: bool = True, attn_mask: bool = False, activation_function: str = 'tanh', precision: str = 'float64', resnet_dt: bool = False, exclude_types: List[Tuple[int, int]] = [], env_protection: float = 0.0, scaling_factor: int = 1.0, normalize=True, temperature=None, concat_output_tebd: bool = True, trainable: bool = True, trainable_ln: bool = True, ln_eps: float | None = 1e-05, smooth_type_embedding: bool = True, type_one_side: bool = False, stripped_type_embedding: bool | None = None, spin=None, type: str | None = None, seed: int | None = None, old_impl: bool = False)

Bases: *deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor*, `torch.nn.Module`

Attention-based descriptor which is proposed in the pretrainable DPA-1[1] model.

This descriptor, $\mathcal{D}^i \in \mathbb{R}^{M \times M_<}$, is given by

$$\mathcal{D}^i = \frac{1}{N_c^2}(\hat{\mathcal{G}}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \hat{\mathcal{G}}^i_<,$$

where $\hat{\mathcal{G}}^i$ represents the embedding matrix:math:mathcal{G}^i after additional self-attention mechanism and $\mathcal{R}^i$ is defined by the full case in the se_e2_a descriptor. Note that we obtain $\mathcal{G}^i$ using the type embedding method by default in this descriptor.

To perform the self-attention mechanism, the queries $\mathcal{Q}^{i,l} \in \mathbb{R}^{N_c \times d_k}$, keys $\mathcal{K}^{i,l} \in \mathbb{R}^{N_c \times d_k}$, and values $\mathcal{V}^{i,l} \in \mathbb{R}^{N_c \times d_v}$ are first obtained:

$$\left(\mathcal{Q}^{i,l}\right)_j = Q_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$

$$\left(\mathcal{K}^{i,l}\right)_j = K_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$

$$\left(\mathcal{V}^{i,l}\right)_j = V_l\left(\left(\mathcal{G}^{i,l-1}\right)_j\right),$$

where Q_l, K_l, V_l represent three trainable linear transformations that output the queries and keys of dimension d_k and values of dimension d_v, and l is the index of the attention layer. The input embedding matrix to the attention layers, denoted by $\mathcal{G}^{i,0}$, is chosen as the two-body embedding matrix.

Then the scaled dot-product attention method is adopted:

$$A(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{V}^{i,l}, \mathcal{R}^{i,l}) = \varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right) \mathcal{V}^{i,l},$$

where $\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right) \in \mathbb{R}^{N_c \times N_c}$ is attention weights. In the original attention method, one typically has $\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}\right) = \mathrm{softmax}\left(\frac{\mathcal{Q}^{i,l}(\mathcal{K}^{i,l})^T}{\sqrt{d_k}}\right)$, with $\sqrt{d_k}$ being the normalization temperature. This is slightly modified to incorporate the angular information:

$$\varphi\left(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}\right) = \mathrm{softmax}\left(\frac{\mathcal{Q}^{i,l}(\mathcal{K}^{i,l})^T}{\sqrt{d_k}}\right) \odot \hat{\mathcal{R}}^i(\hat{\mathcal{R}}^i)^T,$$

where $\hat{\mathcal{R}}^i \in \mathbb{R}^{N_c \times 3}$ denotes normalized relative coordinates,
$\hat{\mathcal{R}}^i_j = \frac{\boldsymbol{r}_{ij}}{\|\boldsymbol{r}_{ij}\|}$ and $\odot$ means element-wise multiplication.

Then layer normalization is added in a residual way to finally obtain the self-attention local embedding matrix
$\hat{\mathcal{G}}^i = \mathcal{G}^{i,L_a}$ after L_a attention layers:[^1]

$$\mathcal{G}^{i,l} = \mathcal{G}^{i,l-1} + \mathrm{LayerNorm}(A(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{V}^{i,l}, \mathcal{R}^{i,l})).$$

Parameters

rcut: float
: The cut-off radius r_c

rcut_smth: float
: From where the environment matrix should be smoothed r_s

sel
: [`list`[`int`], `int`] list[int]: sel[i] specifies the maxmum number of type i atoms in the cut-off radius int: the total maxmum number of atoms in the cut-off radius

ntypes
: [`int`] Number of element types

neuron
: [`list`[`int`]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron: int
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

tebd_dim: int
: Dimension of the type embedding

tebd_input_mode: str
: The input mode of the type embedding. Supported modes are [“concat”, “strip”]. - “concat”: Concatenate the type embedding with the smoothed radial information as the union input for the embedding network. - “strip”: Use a separated embedding network for the type embedding and combine the output with the radial embedding network output.

resnet_dt: bool
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable: bool
: If the weights of this descriptors are trainable.

trainable_ln: bool
: Whether to use trainable shift and scale weights in layer normalization.

ln_eps: float, Optional
The epsilon value for layer normalization.

type_one_side: bool
If 'False', type embeddings of both neighbor and central atoms are considered. If 'True', only type embeddings of neighbor atoms are considered. Default is 'False'.

attn: int
Hidden dimension of the attention vectors

attn_layer: int
Number of attention layers

attn_dotr: bool
If dot the angular gate to the attention weights

attn_mask: bool
(Only support False to keep consistent with other backend references.) (Not used in this version. True option is not implemented.) If mask the diagonal of attention weights

exclude_types
[List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

env_protection: float
Protection parameter to prevent division by zero errors during environment matrix calculations.

set_davg_zero: bool
Set the shift of embedding net input to zero.

activation_function: str
The activation function in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision: str
The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

scaling_factor: float
The scaling factor of normalization in calculations of attention weights. If temperature is None, the scaling of attention weights is (N_dim * scaling_factor)**0.5

normalize: bool
Whether to normalize the hidden vectors in attention weights calculation.

temperature: float
If not None, the scaling of attention weights is temperature itself.

smooth_type_embedding: bool
Whether to use smooth process in attention weights calculation.

concat_output_tebd: bool
Whether to concat type embedding at the output of the descriptor.

stripped_type_embedding: bool, Optional
(Deprecated, kept only for compatibility.) Whether to strip the type embedding into a separate embedding network. Setting this parameter to True is equivalent to setting tebd_input_mode to 'strip'. Setting it to False is equivalent to setting tebd_input_mode to 'concat'. The default value is None, which means the tebd_input_mode setting will be used instead.

spin
(Only support None to keep consistent with other backend references.) (Not used in this version. Not-none option is not implemented.) The old implementation of deepspin.

References

[1]

property dim_out

property dim_emb

get_rcut() → float
Returns the cut-off radius.

get_nsel() → int
Returns the number of selected atoms in the cut-off radius.

get_sel() → List[int]
Returns the number of selected atoms for each type.

get_ntypes() → int
Returns the number of element types.

get_dim_out() → int
Returns the output dimension.

get_dim_emb() → int
Returns the embedding dimension of g2.

mixed_types() → bool
If true, the discriptor 1. assumes total number of atoms aligned across frames; 2. requires a neighbor list that does not distinguish different atomic types.

If false, the discriptor 1. assumes total number of atoms of each atom type aligned across frames; 2. requires a neighbor list that distinguishes different atomic types.

share_params(base_class, shared_level, resume=False)
Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)
Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.

- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

set_stat_mean_and_stddev(mean: torch.Tensor, stddev: torch.Tensor) → None

serialize() → dict

Serialize the obj to dict.

classmethod deserialize(data: dict) → DescrptDPA1

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

BD
The deserialized descriptor

forward(extended_coord: torch.Tensor, extended_atype: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None)

Compute the descriptor.

Parameters

extended_coord
The extended coordinates of atoms. shape: nf x (nallx3)

extended_atype
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping, not required by this descriptor.

comm_dict
The data needed for communication for parallel inference.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. shape: nf x nloc x nnei x ng

h2
The rotationally equivariant pair-partical representation. shape: nf x nloc x nnei x 3

sw
The smooth switch function. shape: nf x nloc x nnei

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

class deepmd.pt.model.descriptor.DescrptDPA2(ntypes: int, repinit_rcut: float, repinit_rcut_smth: float, repinit_nsel: int, repformer_rcut: float, repformer_rcut_smth: float, repformer_nsel: int, tebd_dim: int = 8, concat_output_tebd: bool = True, repinit_neuron: List[int] = [25, 50, 100], repinit_axis_neuron: int = 16, repinit_set_davg_zero: bool = True, repinit_activation='tanh', repformer_nlayers: int = 3, repformer_g1_dim: int = 128, repformer_g2_dim: int = 16, repformer_axis_dim: int = 4, repformer_do_bn_mode: str = 'no', repformer_bn_momentum: float = 0.1, repformer_update_g1_has_conv: bool = True, repformer_update_g1_has_drrd: bool = True, repformer_update_g1_has_grrg: bool = True, repformer_update_g1_has_attn: bool = True, repformer_update_g2_has_g1g1: bool = True, repformer_update_g2_has_attn: bool = True, repformer_update_h2: bool = False, repformer_attn1_hidden: int = 64, repformer_attn1_nhead: int = 4, repformer_attn2_hidden: int = 16, repformer_attn2_nhead: int = 4, repformer_attn2_has_gate: bool = False, repformer_activation: str = 'tanh', repformer_update_style: str = 'res_avg', repformer_set_davg_zero: bool = True, repformer_add_type_ebd_to_seq: bool = False, env_protection: float = 0.0, trainable: bool = True, exclude_types: List[Tuple[int, int]] = [], type: str | None = None, rcut: float | None = None, rcut_smth: float | None = None, sel: int | None = None)

Bases: `torch.nn.Module`, *deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F
```

(continues on next page)

(continued from previous page)

```
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

property dim_out

property dim_emb
Returns the embedding dimension g2.

get_rcut() → float
Returns the cut-off radius.

get_nsel() → int
Returns the number of selected atoms in the cut-off radius.

get_sel() → List[int]
Returns the number of selected atoms for each type.

get_ntypes() → int
Returns the number of element types.

get_dim_out() → int
Returns the output dimension of this descriptor.

get_dim_emb() → int
Returns the embedding dimension of this descriptor.

mixed_types() → bool
If true, the discriptor 1. assumes total number of atoms aligned across frames; 2. requires a neighbor list that does not distinguish different atomic types.

If false, the discriptor 1. assumes total number of atoms of each atom type aligned across frames; 2. requires a neighbor list that distinguishes different atomic types.

share_params(base_class, shared_level, resume=False)
Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)

Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

abstract serialize() → dict

Serialize the obj to dict.

abstract classmethod deserialize() → DescrptDPA2

Deserialize from a dict.

forward(extended_coord: torch.Tensor, extended_atype: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None)

Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping, mapps extended region index to local region.

comm_dict
The data needed for communication for parallel inference.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. shape: nf x nloc x nnei x ng

h2
The rotationally equivariant pair-partical representation. shape: nf x nloc x nnei x 3

sw
The smooth switch function. shape: nf x nloc x nnei

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

deepmd.pt.model.descriptor.prod_env_mat(extended_coord, nlist, atype, mean, stddev, rcut: float, rcut_smth: float, radial_only: bool = False, protection: float = 0.0)

Generate smooth environment matrix from atom coordinates and other context.

Args: - extended_coord: Copied atom coordinates with shape [nframes, nall*3]. - atype: Atom types with shape [nframes, nloc]. - mean: Average value of descriptor per element type with shape [len(sec), nnei, 4 or 1]. - stddev: Standard deviation of descriptor per element type with shape [len(sec), nnei, 4 or 1]. - rcut: Cut-off radius. - rcut_smth: Smooth hyper-parameter for pair force & energy. - radial_only: Whether to return a full description or a radial-only descriptor. - protection: Protection parameter to prevent division by zero errors during calculations.

Returns

- env_mat: **Shape is** [**nframes**, **natoms**[1]*nnei*4].

class deepmd.pt.model.descriptor.DescrptGaussianLcc(rcut, rcut_smth, sel: int, ntypes: int, num_pair: int, embed_dim: int = 768, kernel_num: int = 128, pair_embed_dim: int = 64, num_block: int = 1, layer_num: int = 12, attn_head: int = 48, pair_hidden_dim: int = 16, ffn_embedding_dim: int = 768, dropout: float = 0.0, droppath_prob: float = 0.1, pair_dropout: float = 0.25, attention_dropout: float = 0.1, activation_dropout: float = 0.1, pre_ln: bool = True, do_tag_embedding: bool = False, tag_ener_pref: bool = False, atomic_sum_gbf: bool = False, pre_add_seq: bool = True, tri_update: bool = True, **kwargs)

Bases: `torch.nn.Module`, *deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

property dim_out

Returns the output dimension of atomic representation.

property dim_in

Returns the atomic input dimension of this descriptor.

property dim_emb

Returns the output dimension of pair representation.

compute_input_stats(merged: List[dict], path: deepmd.utils.path.DPPath | None = None)

Update mean and stddev for descriptor elements.

forward(extended_coord, nlist, atype, nlist_type, nlist_loc=None, atype_tebd=None, nlist_tebd=None, seq_input=None)

Calculate the atomic and pair representations of this descriptor.

Args: - extended_coord: Copied atom coordinates with shape [nframes, nall, 3]. - nlist: Neighbor list with shape [nframes, nloc, nnei]. - atype: Atom type with shape [nframes, nloc]. - nlist_type: Atom type of neighbors with shape [nframes, nloc, nnei]. - nlist_loc: Local index of neighbor list with shape [nframes, nloc, nnei]. - atype_tebd: Atomic type embedding with shape [nframes, nloc, tebd_dim]. - nlist_tebd: Type embeddings of neighbor with shape [nframes, nloc, nnei, tebd_dim]. - seq_input: The sequential input from other descriptor with

shape [nframes, nloc, tebd_dim] or [nframes * nloc, 1 + nnei, tebd_dim]

Returns

- result: *descriptor* **with shape** [**nframes**, **nloc**, **self.filter_neuron**[-1] * **self.axis_neuron**].
- ret: **environment matrix with shape** [**nframes**, **nloc**, **self.neei**, **out_size**]

class deepmd.pt.model.descriptor.DescrptHybrid(list: List[deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor | Dict[str, Any]], **kwargs)

Bases: *deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor*, torch.nn.Module

Concate a list of descriptors to form a new descriptor.

Parameters

list
[list][List[Union[*BaseDescriptor*, Dict[str, *Any*]]]] Build a descriptor from the concatenation of the list of descriptors. The descriptor can be either an object or a dictionary.

get_rcut() → float

Returns the cut-off radius.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_ntypes() → int

Returns the number of element types.

get_dim_out() → int

Returns the output dimension.

get_dim_emb() → int

Returns the output dimension.

mixed_types()

Returns if the descriptor requires a neighbor list that distinguish different atomic types or not.

share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

compute_input_stats(merged: List[dict], path: deepmd.utils.path.DPPath | None = None)

Update mean and stddev for descriptor elements.

forward(coord_ext: torch.Tensor, atype_ext: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None)

Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping, not required by this descriptor.

comm_dict
The data needed for communication for parallel inference.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3. This descriptor returns None

g2
The rotationally invariant pair-partical representation. this descriptor returns None

h2
The rotationally equivariant pair-partical representation. this descriptor returns None

sw
The smooth switch function. this descriptor returns None

classmethod update_sel(global_jdata: dict, local_jdata: dict) → dict

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

serialize() → dict

Serialize the obj to dict.

classmethod deserialize(data: dict) → DescrptHybrid

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

BD
The deserialized descriptor

class deepmd.pt.model.descriptor.DescrptBlockRepformers(rcut, rcut_smth, sel: int, ntypes: int, nlayers: int = 3, g1_dim=128, g2_dim=16, axis_dim: int = 4, direct_dist: bool = False, do_bn_mode: str = 'no', bn_momentum: float = 0.1, update_g1_has_conv: bool = True, update_g1_has_drrd: bool = True, update_g1_has_grrg: bool = True, update_g1_has_attn: bool = True, update_g2_has_g1g1: bool = True, update_g2_has_attn: bool = True, update_h2: bool = False, attn1_hidden: int = 64, attn1_nhead: int = 4, attn2_hidden: int = 16, attn2_nhead: int = 4, attn2_has_gate: bool = False, activation_function: str = 'tanh', update_style: str = 'res_avg', set_davg_zero: bool = True, smooth: bool = True, add_type_ebd_to_seq: bool = False, exclude_types: List[Tuple[int, int]] = [], env_protection: float = 0.0, type: str | None = None)

Bases: *deepmd.pt.model.descriptor.descriptor.DescriptorBlock*

The building block of descriptor. Given the input descriptor, provide with the atomic coordinates, atomic types and neighbor list, calculate the new descriptor.

property dim_out

Returns the output dimension of this descriptor.

property dim_in

Returns the atomic input dimension of this descriptor.

property dim_emb

Returns the embedding dimension g2.

get_rcut() → float

Returns the cut-off radius.

get_nsel() → int

Returns the number of selected atoms in the cut-off radius.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_ntypes() → int

Returns the number of element types.

get_dim_out() → int

Returns the output dimension.

get_dim_in() → int

Returns the input dimension.

get_dim_emb() → int

Returns the embedding dimension g2.

mixed_types() → bool

If true, the discriptor 1. assumes total number of atoms aligned across frames; 2. requires a neighbor list that does not distinguish different atomic types.

If false, the discriptor 1. assumes total number of atoms of each atom type aligned across frames; 2. requires a neighbor list that distinguishes different atomic types.

reinit_exclude(exclude_types: List[Tuple[int, int]] = [])

forward(nlist: torch.Tensor, extended_coord: torch.Tensor, extended_atype: torch.Tensor, extended_atype_embd: torch.Tensor | None = None, mapping: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None)

Calculate DescriptorBlock.

compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)

Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

get_stats() → Dict[str, deepmd.utils.env_mat_stat.StatItem]

Get the statistics of the descriptor.

class deepmd.pt.model.descriptor.DescrptBlockSeA(rcut, rcut_smth, sel, neuron=[25, 50, 100], axis_neuron=16, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'float64', resnet_dt: bool = False, exclude_types: List[Tuple[int, int]] = [], env_protection: float = 0.0, old_impl: bool = False, type_one_side: bool = True, trainable: bool = True, **kwargs)

Bases: *deepmd.pt.model.descriptor.DescriptorBlock*

The building block of descriptor. Given the input descriptor, provide with the atomic coordinates, atomic types and neighbor list, calculate the new descriptor.

property dim_out

Returns the output dimension of this descriptor.

property dim_in

Returns the atomic input dimension of this descriptor.

ndescrpt: Final[int]

__constants__: ClassVar[list] = ['ndescrpt']

get_rcut() → float

Returns the cut-off radius.

get_nsel() → int

Returns the number of selected atoms in the cut-off radius.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_ntypes() → int

Returns the number of element types.

get_dim_out() → int

Returns the output dimension.

get_dim_emb() → int

Returns the output dimension.

get_dim_in() → int

Returns the input dimension.

mixed_types() → bool

If true, the discriptor 1. assumes total number of atoms aligned across frames; 2. requires a neighbor list that does not distinguish different atomic types.

If false, the discriptor 1. assumes total number of atoms of each atom type aligned across frames; 2. requires a neighbor list that distinguishes different atomic types.

__setitem__(key, value)

__getitem__(key)

compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)

Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

get_stats() → Dict[str, deepmd.utils.env_mat_stat.StatItem]

Get the statistics of the descriptor.

reinit_exclude(exclude_types: List[Tuple[int, int]] = [])

forward(nlist: torch.Tensor, extended_coord: torch.Tensor, extended_atype: torch.Tensor, extended_atype_embd: torch.Tensor | None = None, mapping: torch.Tensor | None = None)

Calculate decoded embedding for each atom.

Args: - coord: Tell atom coordinates with shape [nframes, natoms[1]*3]. - atype: Tell atom types with shape [nframes, natoms[1]]. - natoms: Tell atom count and element count. Its shape is [2+self.ntypes]. - box: Tell simulation box with shape [nframes, 9].

Returns

- torch.Tensor: *descriptor* **matrix with shape** [**nframes**, **natoms**[0]*self.filter_neuron[-1]*self.axis_neuron].

class deepmd.pt.model.descriptor.DescrptSeA(rcut, rcut_smth, sel, neuron=[25, 50, 100], axis_neuron=16, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'float64', resnet_dt: bool = False, exclude_types: List[Tuple[int, int]] = [], env_protection: float = 0.0, old_impl: bool = False, type_one_side: bool = True, **kwargs)

Bases: *deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor*, torch.nn.Module

Base descriptor provides the interfaces of descriptor.

property dim_out

Returns the output dimension of this descriptor.

get_rcut() → float

Returns the cut-off radius.

get_nsel() → int

Returns the number of selected atoms in the cut-off radius.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_ntypes() → int

Returns the number of element types.

get_dim_out() → int

Returns the output dimension.

get_dim_emb() → int

Returns the output dimension.

mixed_types()

Returns if the descriptor requires a neighbor list that distinguish different atomic types or not.

share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)

Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

reinit_exclude(exclude_types: List[Tuple[int, int]] = [])

Update the type exclusions.

forward(coord_ext: torch.Tensor, atype_ext: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None, comm_dict: Dict[str, torch.Tensor] | None = None)

Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping, not required by this descriptor.

comm_dict
The data needed for communication for parallel inference.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. this descriptor returns None

h2
The rotationally equivariant pair-partical representation. this descriptor returns None

sw
The smooth switch function.

set_stat_mean_and_stddev(mean: torch.Tensor, stddev: torch.Tensor) → None

serialize() → dict

Serialize the obj to dict.

classmethod deserialize(data: dict) → DescrptSeA

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

BD
The deserialized descriptor

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

class deepmd.pt.model.descriptor.DescrptSeR(rcut, rcut_smth, sel, neuron=[25, 50, 100], set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'float64', resnet_dt: bool = False, exclude_types: List[Tuple[int, int]] = [], env_protection: float = 0.0, old_impl: bool = False, trainable: bool = True, **kwargs)

Bases: *deepmd.pt.model.descriptor.base_descriptor.BaseDescriptor*, torch.nn.Module

Base descriptor provides the interfaces of descriptor.

get_rcut() → float

Returns the cut-off radius.

get_nsel() → int

Returns the number of selected atoms in the cut-off radius.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_ntypes() → int

Returns the number of element types.

get_dim_out() → int

Returns the output dimension.

abstract get_dim_emb() → int

Returns the output dimension.

get_dim_in() → int

Returns the input dimension.

mixed_types() → bool

If true, the discriptor 1. assumes total number of atoms aligned across frames; 2. requires a neighbor list that does not distinguish different atomic types.

If false, the discriptor 1. assumes total number of atoms of each atom type aligned across frames; 2. requires a neighbor list that distinguishes different atomic types.

share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

compute_input_stats(merged: Callable[[], List[dict]] | List[dict], path: deepmd.utils.path.DPPath | None = None)

Compute the input statistics (e.g. mean and stddev) for the descriptors from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

path
[Optional[DPPath]] The path to the stat file.

get_stats() → Dict[str, deepmd.utils.env_mat_stat.StatItem]

Get the statistics of the descriptor.

__setitem__(key, value)

__getitem__(key)

reinit_exclude(exclude_types: List[Tuple[int, int]] = [])

forward(coord_ext: torch.Tensor, atype_ext: torch.Tensor, nlist: torch.Tensor, mapping: torch.Tensor | None = None)

Compute the descriptor.

Parameters

coord_ext
The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext
The extended aotm types. shape: nf x nall

nlist
The neighbor list. shape: nf x nloc x nnei

mapping
The index mapping, not required by this descriptor.

Returns

descriptor
The descriptor. shape: nf x nloc x (ng x axis_neuron)

gr
The rotationally equivariant and permutationally invariant single particle representation. shape: nf x nloc x ng x 3

g2
The rotationally invariant pair-partical representation. this descriptor returns None

h2
The rotationally equivariant pair-partical representation. this descriptor returns None

sw
The smooth switch function.

set_stat_mean_and_stddev(mean: torch.Tensor, stddev: torch.Tensor) → None

serialize() → dict
Serialize the obj to dict.

classmethod deserialize(data: dict) → DescrptSeR
Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

BD
The deserialized descriptor

classmethod update_sel(global_jdata: dict, local_jdata: dict)
Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

deepmd.pt.model.model

The model that takes the coordinates, cell and atom types as input and predicts some property. The models are automatically generated from atomic models by the deepmd.dpmodel.make_model method.

The make_model method does the reduction, auto-differentiation and communication of the atomic properties according to output variable definition deepmd.dpmodel.OutputVariableDef.

All models should be inherited from *deepmd.pt.model.model.model.BaseModel*. Models generated by make_model have already done it.

Submodules

deepmd.pt.model.model.dipole_model

Module Contents

Classes

DipoleModel	A base class to implement common methods for all the Models.

Attributes

*DPDOSModel_*

deepmd.pt.model.model.dipole_model.**DPDOSModel_**

class deepmd.pt.model.model.dipole_model.**DipoleModel**(*args, **kwargs)

Bases: *deepmd.pt.model.model.dp_model.DPModelCommon*, *DPDOSModel_*

A base class to implement common methods for all the Models.

model_type = 'dipole'

forward(coord, atype, box: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

forward_lower(extended_coord, extended_atype, nlist, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False)

deepmd.pt.model.model.dos_model

Module Contents

Classes

DOSModel	A base class to implement common methods for all the Models.

Attributes

*DPDOSModel_*	

deepmd.pt.model.model.dos_model.DPDOSModel_

class deepmd.pt.model.model.dos_model.DOSModel(*args, **kwargs)

Bases: *deepmd.pt.model.model.dp_model.DPModelCommon*, *DPDOSModel_*

A base class to implement common methods for all the Models.

model_type = 'dos'

forward(coord, atype, box: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

get_numb_dos() → int

Get the number of DOS for DOSFittingNet.

forward_lower(extended_coord, extended_atype, nlist, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False)

deepmd.pt.model.model.dp_model

Module Contents

Classes

DPModelCommon	A base class to implement common methods for all the Models.

class deepmd.pt.model.model.dp_model.DPModelCommon

A base class to implement common methods for all the Models.

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

get_fitting_net()

Get the fitting network.

get_descriptor()

Get the descriptor.

deepmd.pt.model.model.dp_zbl_model

Module Contents

Classes

DPZBLModel	Base class for all neural network modules.

Attributes

*DPZBLModel_*

deepmd.pt.model.model.dp_zbl_model.**DPZBLModel_**

class deepmd.pt.model.model.dp_zbl_model.**DPZBLModel**(*args, **kwargs)

Bases: *DPZBLModel_*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

model_type = 'ener'

forward(coord, atype, box: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

forward_lower(extended_coord, extended_atype, nlist, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False)

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

deepmd.pt.model.model.ener_model

Module Contents

Classes

EnergyModel	A base class to implement common methods for all the Models.

Attributes

*DPEnergyModel_*

deepmd.pt.model.model.ener_model.**DPEnergyModel_**

class deepmd.pt.model.model.ener_model.**EnergyModel**(*args, **kwargs)

Bases: *deepmd.pt.model.model.dp_model.DPModelCommon*, *DPEnergyModel_*

A base class to implement common methods for all the Models.

model_type = 'ener'

forward(coord, atype, box: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

forward_lower(extended_coord, extended_atype, nlist, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False, comm_dict: Dict[str, torch.Tensor] | None = None)

deepmd.pt.model.model.frozen

Module Contents

Classes

FrozenModel	Load model from a frozen model, which cannot be trained.

class deepmd.pt.model.model.frozen.FrozenModel(model_file: str, **kwargs)

Bases: *deepmd.pt.model.model.model.BaseModel*

Load model from a frozen model, which cannot be trained.

Parameters

model_file
[str] The path to the frozen model

fitting_output_def() → deepmd.dpmodel.output_def.FittingOutputDef

Get the output def of developer implemented atomic models.

get_rcut() → float

Get the cut-off radius.

get_type_map() → List[str]

Get the type map.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim).

If False, the shape is (nframes, nloc, ndim).

mixed_types() → bool

If true, the model 1. assumes total number of atoms aligned across frames; 2. uses a neighbor list that does not distinguish different atomic types.

If false, the model 1. assumes total number of atoms of each atom type aligned across frames; 2. uses a neighbor list that distinguishes different atomic types.

forward(coord, atype, box: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

get_model_def_script() → str

Get the model definition script.

serialize() → dict

Serialize the model.

Returns

dict
The serialized data

classmethod deserialize(data: dict)

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

BaseModel
The deserialized model

get_nnei() → int

Returns the total number of selected neighboring atoms in the cut-off radius.

get_nsel() → int

Returns the total number of selected neighboring atoms in the cut-off radius.

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

model_output_type() → str

Get the output type for the model.

deepmd.pt.model.model.make_hessian_model

Module Contents

Functions

make_hessian_model(T_Model)	Make a model that can compute Hessian.

deepmd.pt.model.model.make_hessian_model.**make_hessian_model**(T_Model)

Make a model that can compute Hessian.

LIMITATION: this model is not jitable due to the restrictions of torch jit script.

LIMITATION: only the hessian of forward_common is available.

Parameters

T_Model
The model. Should provide the forward_common and atomic_output_def methods

Returns

The model computes hessian.

deepmd.pt.model.model.make_model

Module Contents

Functions

make_model(T_AtomicModel)	Make a model as a derived class of an atomic model.

deepmd.pt.model.model.make_model.**make_model**(T_AtomicModel: Type[deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomic

Make a model as a derived class of an atomic model.

The model provide two interfaces.

1. the forward_common_lower, that takes extended coordinates, atyps and neighbor list, and outputs the atomic and property and derivatives (if required) on the extended region.

2. the forward_common, that takes coordinates, atypes and cell and predicts the atomic and reduced property, and derivatives (if required) on the local region.

Parameters

T_AtomicModel
The atomic model.

Returns

CM
The model.

deepmd.pt.model.model.model

Module Contents

Classes

BaseModel	Base class for all neural network modules.

class deepmd.pt.model.model.model.**BaseModel**(*args, **kwargs)

Bases: torch.nn.Module, make_base_model()

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

abstract compute_or_load_stat(sampled_func, stat_file_path: deepmd.utils.path.DPPath | None = None)

Compute or load the statistics parameters of the model, such as mean and standard deviation of descriptors or the energy bias of the fitting net. When sampled is provided, all the statistics parameters will be calculated (or re-calculated for update), and saved in the stat_file_path‘(s). When ‘sampled is not provided, it will check the existence of ‘stat_file_path‘(s) and load the calculated statistics parameters.

Parameters

sampled_func
The sampled data frames from different data systems.

stat_file_path
The path to the statistics files.

get_model_def_script() → str

Get the model definition script.

get_ntypes()

Returns the number of element types.

deepmd.pt.model.model.polar_model

Module Contents

Classes

PolarModel	A base class to implement common methods for all the Models.

Attributes

*DPDOSModel_*

deepmd.pt.model.model.polar_model.**DPDOSModel_**

class deepmd.pt.model.model.polar_model.**PolarModel**(*args, **kwargs)

Bases: *deepmd.pt.model.model.dp_model.DPModelCommon*, *DPDOSModel_*

A base class to implement common methods for all the Models.

model_type = 'polar'

forward(coord, atype, box: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

forward_lower(extended_coord, extended_atype, nlist, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False)

deepmd.pt.model.model.spin_model

Module Contents

Classes

SpinModel	A spin model wrapper, with spin input preprocess and output split.
SpinEnergyModel	A spin model for energy.

class deepmd.pt.model.model.spin_model.**SpinModel**(backbone_model, spin: deepmd.utils.spin.Spin)

Bases: `torch.nn.Module`

A spin model wrapper, with spin input preprocess and output split.

process_spin_input(coord, atype, spin)

Generate virtual coordinates and types, concat into the input.

process_spin_input_lower(extended_coord, extended_atype, extended_spin, nlist, mapping: torch.Tensor | None = None)

Add extended_spin into extended_coord to generate virtual atoms, and extend nlist and mapping. Note that the final extended_coord_updated with shape [nframes, nall + nall, 3] has the following order: - [:, :nloc]: original nloc real atoms. - [:, nloc: nloc + nloc]: virtual atoms corresponding to nloc real atoms. - [:, nloc + nloc: nloc + nall]: ghost real atoms. - [:, nloc + nall: nall + nall]: virtual atoms corresponding to ghost real atoms.

process_spin_output(atype, out_tensor, add_mag: bool = True, virtual_scale: bool = True)

Split the output both real and virtual atoms, and scale the latter. add_mag: whether to add magnetic tensor onto the real tensor.

Default: True. e.g. Ture for forces and False for atomic virials on real atoms.

virtual_scale: whether to scale the magnetic tensor with virtual scale factor.
Default: True. e.g. Ture for forces and False for atomic virials on virtual atoms.

process_spin_output_lower(extended_atype, extended_out_tensor, nloc: int, add_mag: bool = True, virtual_scale: bool = True)

Split the extended output of both real and virtual atoms with switch, and scale the latter. add_mag: whether to add magnetic tensor onto the real tensor.

Default: True. e.g. Ture for forces and False for atomic virials on real atoms.

virtual_scale: whether to scale the magnetic tensor with virtual scale factor.
Default: True. e.g. Ture for forces and False for atomic virials on virtual atoms.

static extend_nlist(extended_atype, nlist)

static concat_switch_virtual(extended_tensor, extended_tensor_virtual, nloc: int)

Concat real and virtual extended tensors, and switch all the local ones to the first nloc * 2 atoms. - [:, :nloc]: original nloc real atoms. - [:, nloc: nloc + nloc]: virtual atoms corresponding to nloc real atoms. - [:, nloc + nloc: nloc + nall]: ghost real atoms. - [:, nloc + nall: nall + nall]: virtual atoms corresponding to ghost real atoms.

static expand_aparam(aparam, nloc: int)

Expand the atom parameters for virtual atoms if necessary.

get_type_map() → List[str]

Get the type map.

get_rcut()

Get the cut-off radius.

get_dim_fparam()

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam()

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model. Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim). If False, the shape is (nframes, nloc, ndim).

model_output_type() → List[str]

Get the output type for the model.

get_model_def_script() → str

Get the model definition script.

get_nnei() → int

Returns the total number of selected neighboring atoms in the cut-off radius.

get_nsel() → int

Returns the total number of selected neighboring atoms in the cut-off radius.

has_spin() → bool

Returns whether it has spin input and output.

__getattr__(name)

Get attribute from the wrapped model.

compute_or_load_stat(sampled_func, stat_file_path: deepmd.utils.path.DPPath | None = None)

Compute or load the statistics parameters of the model, such as mean and standard deviation of descriptors or the energy bias of the fitting net. When sampled is provided, all the statistics parameters will be calculated (or re-calculated for update), and saved in the stat_file_path`(s). When `sampled is not provided, it will check the existence of `stat_file_path`(s) and load the calculated statistics parameters.

Parameters

sampled_func
The lazy sampled function to get data frames from different data systems.

stat_file_path
The dictionary of paths to the statistics files.

forward_common(coord, atype, spin, box: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

forward_common_lower(extended_coord, extended_atype, extended_spin, nlist, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False)

serialize() → dict

classmethod deserialize(data) → SpinModel

class deepmd.pt.model.model.spin_model.SpinEnergyModel(backbone_model, spin: deepmd.utils.spin.Spin)

Bases: *SpinModel*

A spin model for energy.

model_type = 'ener'

forward(coord, atype, spin, box: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

forward_lower(extended_coord, extended_atype, extended_spin, nlist, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False)

deepmd.pt.model.model.transform_output

Module Contents

Functions

atomic_virial_corr(extended_coord, atom_energy)	
task_deriv_one(atom_energy, energy, extended_coord[, ...])	
get_leading_dims(vv, vdef)	Get the dimensions of nf x nloc.
get_atom_axis(vdef)	Get the axis of atoms.
take_deriv(vv, svv, vdef, coord_ext[, do_virial, ...])	
fit_output_to_model_output(→ Dict[str, torch.Tensor])	Transform the output of the fitting network to
communicate_extended_output(→ Dict[str, torch.Tensor])	Transform the output of the model network defined on

deepmd.pt.model.model.transform_output.**atomic_virial_corr**(extended_coord: torch.Tensor, atom_energy: torch.Tensor)

deepmd.pt.model.model.transform_output.**task_deriv_one**(atom_energy: torch.Tensor, energy: torch.Tensor, extended_coord: torch.Tensor, do_virial: bool = True, do_atomic_virial: bool = False)

deepmd.pt.model.model.transform_output.**get_leading_dims**(vv: torch.Tensor, vdef: deepmd.dpmodel.OutputVariableDef)

Get the dimensions of nf x nloc.

deepmd.pt.model.model.transform_output.**get_atom_axis**(vdef: torch.Tensor)

Get the axis of atoms.

deepmd.pt.model.model.transform_output.**take_deriv**(vv: torch.Tensor, svv: torch.Tensor, vdef: deepmd.dpmodel.OutputVariableDef, coord_ext: torch.Tensor, do_virial: bool = False, do_atomic_virial: bool = False)

deepmd.pt.model.model.transform_output.**fit_output_to_model_output**(fit_ret: Dict[str, torch.Tensor], fit_output_def: deepmd.dpmodel.FittingOutputDef, coord_ext: torch.Tensor, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

Transform the output of the fitting network to the model output.

deepmd.pt.model.model.transform_output.**communicate_extended_output**(model_ret: Dict[str, torch.Tensor], model_output_def: deepmd.dpmodel.ModelOutputDef, mapping: torch.Tensor, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

Transform the output of the model network defined on local and ghost (extended) atoms to local atoms.

Package Contents

Classes

DPModelCommon	A base class to implement common methods for all the Models.
DPZBLModel	Base class for all neural network modules.
EnergyModel	A base class to implement common methods for all the Models.
FrozenModel	Load model from a frozen model, which cannot be trained.
BaseModel	Base class for all neural network modules.
SpinEnergyModel	A spin model for energy.
SpinModel	A spin model wrapper, with spin input preprocess and output split.

Functions

make_hessian_model(T_Model)	Make a model that can compute Hessian.
make_model(T_AtomicModel)	Make a model as a derived class of an atomic model.
get_model(model_params)	

class deepmd.pt.model.model.**DPModelCommon**

A base class to implement common methods for all the Models.

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

get_fitting_net()

Get the fitting network.

get_descriptor()

Get the descriptor.

class deepmd.pt.model.model.**DPZBLModel**(*args, **kwargs)

Bases: DPZBLModel_

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call to(), etc.

Note: As per the example above, an __init__() call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

model_type = 'ener'

forward(coord, atype, box: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

forward_lower(extended_coord, extended_atype, nlist, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False)

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

class deepmd.pt.model.model.EnergyModel(*args, **kwargs)

Bases: *deepmd.pt.model.model.dp_model.DPModelCommon*, DPEnergyModel_

A base class to implement common methods for all the Models.

model_type = 'ener'

forward(coord, atype, box: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

forward_lower(extended_coord, extended_atype, nlist, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False, comm_dict: Dict[str, torch.Tensor] | None = None)

class deepmd.pt.model.model.FrozenModel(model_file: str, **kwargs)

Bases: *deepmd.pt.model.model.model.BaseModel*

Load model from a frozen model, which cannot be trained.

Parameters

model_file
[str] The path to the frozen model

fitting_output_def() → deepmd.dpmodel.output_def.FittingOutputDef

Get the output def of developer implemented atomic models.

get_rcut() → float

Get the cut-off radius.

get_type_map() → List[str]

Get the type map.

get_sel() → List[int]

Returns the number of selected atoms for each type.

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

`is_aparam_nall()` → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim).

If False, the shape is (nframes, nloc, ndim).

`mixed_types()` → bool

If true, the model 1. assumes total number of atoms aligned across frames; 2. uses a neighbor list that does not distinguish different atomic types.

If false, the model 1. assumes total number of atoms of each atom type aligned across frames; 2. uses a neighbor list that distinguishes different atomic types.

`forward`(coord, atype, box: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

`get_model_def_script()` → str

Get the model definition script.

`serialize()` → dict

Serialize the model.

Returns

`dict`
The serialized data

`classmethod deserialize`(data: dict)

Deserialize the model.

Parameters

data
[`dict`] The serialized data

Returns

`BaseModel`
The deserialized model

`get_nnei()` → int

Returns the total number of selected neighboring atoms in the cut-off radius.

`get_nsel()` → int

Returns the total number of selected neighboring atoms in the cut-off radius.

`classmethod update_sel`(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[`dict`] The global data, containing the training section

local_jdata
[`dict`] The local data refer to the current class

`model_output_type()` → str

Get the output type for the model.

deepmd.pt.model.model.**make_hessian_model**(T_Model)

Make a model that can compute Hessian.

LIMITATION: this model is not jitable due to the restrictions of torch jit script.

LIMITATION: only the hessian of forward_common is available.

Parameters

T_Model
The model. Should provide the forward_common and atomic_output_def methods

Returns

The *model* **computes** hessian.

deepmd.pt.model.model.**make_model**(T_AtomicModel: Type[deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel])

Make a model as a derived class of an atomic model.

The model provide two interfaces.

1. the forward_common_lower, that takes extended coordinates, atyps and neighbor list, and outputs the atomic and property and derivatives (if required) on the extended region.

2. the forward_common, that takes coordinates, atypes and cell and predicts the atomic and reduced property, and derivatives (if required) on the local region.

Parameters

T_AtomicModel
The atomic model.

Returns

CM
The model.

class deepmd.pt.model.model.**BaseModel**(*args, **kwargs)

Bases: torch.nn.Module, make_base_model()

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call to(), etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

abstract compute_or_load_stat(sampled_func, stat_file_path: deepmd.utils.path.DPPath | None = None)

Compute or load the statistics parameters of the model, such as mean and standard deviation of descriptors or the energy bias of the fitting net. When sampled is provided, all the statistics parameters will be calculated (or re-calculated for update), and saved in the stat_file_path‘(s). When ‘sampled is not provided, it will check the existence of ‘stat_file_path‘(s) and load the calculated statistics parameters.

Parameters

sampled_func
The sampled data frames from different data systems.

stat_file_path
The path to the statistics files.

get_model_def_script() → str

Get the model definition script.

get_ntypes()

Returns the number of element types.

class deepmd.pt.model.model.SpinEnergyModel(backbone_model, spin: deepmd.utils.spin.Spin)

Bases: *SpinModel*

A spin model for energy.

model_type = 'ener'

forward(coord, atype, spin, box: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

forward_lower(extended_coord, extended_atype, extended_spin, nlist, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False)

class deepmd.pt.model.model.SpinModel(backbone_model, spin: deepmd.utils.spin.Spin)

Bases: `torch.nn.Module`

A spin model wrapper, with spin input preprocess and output split.

process_spin_input(coord, atype, spin)

Generate virtual coordinates and types, concat into the input.

process_spin_input_lower(extended_coord, extended_atype, extended_spin, nlist, mapping: torch.Tensor | None = None)

Add extended_spin into extended_coord to generate virtual atoms, and extend nlist and mapping. Note that the final extended_coord_updated with shape [nframes, nall + nall, 3] has the following

order: - [:, :nloc]: original nloc real atoms. - [:, nloc: nloc + nloc]: virtual atoms corresponding to nloc real atoms. - [:, nloc + nloc: nloc + nall]: ghost real atoms. - [:, nloc + nall: nall + nall]: virtual atoms corresponding to ghost real atoms.

process_spin_output(atype, out_tensor, add_mag: bool = True, virtual_scale: bool = True)

Split the output both real and virtual atoms, and scale the latter. add_mag: whether to add magnetic tensor onto the real tensor.

Default: True. e.g. Ture for forces and False for atomic virials on real atoms.

virtual_scale: whether to scale the magnetic tensor with virtual scale factor.
Default: True. e.g. Ture for forces and False for atomic virials on virtual atoms.

process_spin_output_lower(extended_atype, extended_out_tensor, nloc: int, add_mag: bool = True, virtual_scale: bool = True)

Split the extended output of both real and virtual atoms with switch, and scale the latter. add_mag: whether to add magnetic tensor onto the real tensor.

Default: True. e.g. Ture for forces and False for atomic virials on real atoms.

virtual_scale: whether to scale the magnetic tensor with virtual scale factor.
Default: True. e.g. Ture for forces and False for atomic virials on virtual atoms.

static extend_nlist(extended_atype, nlist)

static concat_switch_virtual(extended_tensor, extended_tensor_virtual, nloc: int)

Concat real and virtual extended tensors, and switch all the local ones to the first nloc * 2 atoms. - [:, :nloc]: original nloc real atoms. - [:, nloc: nloc + nloc]: virtual atoms corresponding to nloc real atoms. - [:, nloc + nloc: nloc + nall]: ghost real atoms. - [:, nloc + nall: nall + nall]: virtual atoms corresponding to ghost real atoms.

static expand_aparam(aparam, nloc: int)

Expand the atom parameters for virtual atoms if necessary.

get_type_map() → List[str]

Get the type map.

get_rcut()

Get the cut-off radius.

get_dim_fparam()

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam()

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model. Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

is_aparam_nall() → bool

Check whether the shape of atomic parameters is (nframes, nall, ndim). If False, the shape is (nframes, nloc, ndim).

model_output_type() → List[str]

Get the output type for the model.

get_model_def_script() → str
Get the model definition script.

get_nnei() → int
Returns the total number of selected neighboring atoms in the cut-off radius.

get_nsel() → int
Returns the total number of selected neighboring atoms in the cut-off radius.

has_spin() → bool
Returns whether it has spin input and output.

__getattr__(name)
Get attribute from the wrapped model.

compute_or_load_stat(sampled_func, stat_file_path: deepmd.utils.path.DPPath | None = None)
Compute or load the statistics parameters of the model, such as mean and standard deviation of descriptors or the energy bias of the fitting net. When sampled is provided, all the statistics parameters will be calculated (or re-calculated for update), and saved in the stat_file_path`(s). When `sampled is not provided, it will check the existence of `stat_file_path`(s) and load the calculated statistics parameters.

Parameters

sampled_func
The lazy sampled function to get data frames from different data systems.

stat_file_path
The dictionary of paths to the statistics files.

forward_common(coord, atype, spin, box: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False) → Dict[str, torch.Tensor]

forward_common_lower(extended_coord, extended_atype, extended_spin, nlist, mapping: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None, do_atomic_virial: bool = False)

serialize() → dict

classmethod deserialize(data) → SpinModel

deepmd.pt.model.model.**get_model**(model_params)

deepmd.pt.model.network

Submodules

deepmd.pt.model.network.layernorm

Module Contents

Classes

LayerNorm	Base class for all neural network modules.

Attributes

device

deepmd.pt.model.network.layernorm.**device**

class deepmd.pt.model.network.layernorm.**LayerNorm**(num_in, eps: float = 1e-05, uni_init: bool = True, bavg: float = 0.0, stddev: float = 1.0, precision: str = DEFAULT_PRECISION, trainable: bool = True)

Bases: *deepmd.pt.model.network.mlp.MLPLayer*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

dim_out() → int

forward(xx: torch.Tensor) → torch.Tensor

One Layer Norm used by DP model.

Parameters

xx

[`torch.Tensor`] The input of index.

Returns

yy: `torch.Tensor`
The output.

`serialize()` → dict

Serialize the layer to a dict.

Returns

`dict`
The serialized layer.

`classmethod deserialize`(data: dict) → LayerNorm

Deserialize the layer from a dict.

Parameters

data
[`dict`] The dict to deserialize from.

deepmd.pt.model.network.mlp

Module Contents

Classes

MLPLayer	Base class for all neural network modules.
MLP	Native representation of a neural network.
NetworkCollection	PyTorch implementation of NetworkCollection.

Functions

empty_t(shape, precision)

Attributes

device
__version__
*MLP_*
EmbeddingNet
FittingNet

deepmd.pt.model.network.mlp.**device**

deepmd.pt.model.network.mlp.**__version__ = 'unknown'**

deepmd.pt.model.network.mlp.**empty_t**(shape, precision)

class deepmd.pt.model.network.mlp.**MLPLayer**(num_in, num_out, bias: bool = True, use_timestep: bool = False, activation_function: str | None = None, resnet: bool = False, bavg: float = 0.0, stddev: float = 1.0, precision: str = DEFAULT_PRECISION)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

check_type_consistency()

dim_in() → int

dim_out() → int

forward(xx: torch.Tensor) → torch.Tensor

One MLP layer used by DP model.

Parameters

xx
[`torch.Tensor`] The input.

Returns

yy: torch.Tensor
The output.

serialize() → dict

Serialize the layer to a dict.

Returns

dict
The serialized layer.

classmethod deserialize(data: dict) → MLPLayer

Deserialize the layer from a dict.

Parameters

data
[dict] The dict to deserialize from.

deepmd.pt.model.network.mlp.MLP_

class deepmd.pt.model.network.mlp.MLP(*args, **kwargs)

Bases: *MLP_*

Native representation of a neural network.

Parameters

layers
[list[NativeLayer], optional] The layers of the network.

forward

deepmd.pt.model.network.mlp.EmbeddingNet

deepmd.pt.model.network.mlp.FittingNet

class deepmd.pt.model.network.mlp.NetworkCollection(*args, **kwargs)

Bases: *deepmd.dpmodel.utils.NetworkCollection*, torch.nn.Module

PyTorch implementation of NetworkCollection.

NETWORK_TYPE_MAP: ClassVar[Dict[str, type]]

deepmd.pt.model.network.network

Module Contents

Classes

Dropout	Base class for all neural network modules.
Identity	Base class for all neural network modules.
DropPath	Drop paths (Stochastic Depth) per sample (when applied in main path of residual blocks).
ResidualLinear	Base class for all neural network modules.
TypeFilter	Base class for all neural network modules.
SimpleLinear	Base class for all neural network modules.

continues on next page

Table 1 – continued from previous page

Linear	Applies a linear transformation to the incoming data: $y = xA^T + b$.
Transition	Base class for all neural network modules.
Embedding	A simple lookup table that stores embeddings of a fixed dictionary and size.
NonLinearHead	Base class for all neural network modules.
NonLinear	Base class for all neural network modules.
MaskLMHead	Head for masked language modeling.
ResidualDeep	Base class for all neural network modules.
TypeEmbedNet	Base class for all neural network modules.
TypeEmbedNetConsistent	Type embedding network that is consistent with other backends.
GaussianKernel	Base class for all neural network modules.
GaussianEmbedding	Base class for all neural network modules.
NeighborWiseAttention	Base class for all neural network modules.
NeighborWiseAttentionLayer	Base class for all neural network modules.
GatedSelfAttetion	Base class for all neural network modules.
LocalSelfMultiheadAttention	Base class for all neural network modules.
NodeTaskHead	Base class for all neural network modules.
EnergyHead	Base class for all neural network modules.
OuterProduct	Base class for all neural network modules.
Attention	Base class for all neural network modules.
AtomAttention	Base class for all neural network modules.
TriangleMultiplication	Base class for all neural network modules.
EvoformerEncoderLayer	Base class for all neural network modules.
Evoformer2bEncoder	Base class for all neural network modules.
Evoformer3bEncoderLayer	Base class for all neural network modules.
Evoformer3bEncoder	Base class for all neural network modules.

Functions

Tensor(*shape)	
softmax_dropout(input_x, dropout_prob[, is_training, ...])	
checkpoint_sequential(functions, input_x[, enabled])	
gaussian(x, mean, std)	

deepmd.pt.model.network.network.**Tensor**(*shape)

class deepmd.pt.model.network.network.**Dropout**(p)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(x, inplace: bool = False)

class deepmd.pt.model.network.network.Identity

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(x)

class deepmd.pt.model.network.network.**DropPath**(prob=None)

Bases: torch.nn.Module

Drop paths (Stochastic Depth) per sample (when applied in main path of residual blocks).

forward(x)

extra_repr() → str

Set the extra representation of the module.

To print customized extra information, you should re-implement this method in your own modules. Both single-line and multi-line strings are acceptable.

deepmd.pt.model.network.network.**softmax_dropout**(input_x, dropout_prob, is_training=True, mask=None, bias=None, inplace=True)

deepmd.pt.model.network.network.**checkpoint_sequential**(functions, input_x, enabled=True)

class deepmd.pt.model.network.network.**ResidualLinear**(num_in, num_out, bavg=0.0, stddev=1.0, resnet_dt=False)

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

resnet: Final[int]

forward(inputs)

Return X ?+ X*W+b.

class deepmd.pt.model.network.network.TypeFilter(offset, length, neuron, return_G=False, tebd_dim=0, use_tebd=False, tebd_mode='concat')

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

use_tebd: Final[bool]

tebd_mode: Final[str]

forward(inputs, atype_tebd: torch.Tensor | None = None, nlist_tebd: torch.Tensor | None = None)

Calculate decoded embedding for each atom.

Args: - inputs: Descriptor matrix. Its shape is [nframes*natoms[0], len_descriptor].

Returns

- torch.Tensor: *Embedding* **contributed by** me. **Its shape is** [nframes*natoms[0], 4, **self.neuron**[-1]].

class deepmd.pt.model.network.network.SimpleLinear(num_in, num_out, bavg=0.0, stddev=1.0, use_timestep=False, activate=None, bias: bool = True)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

use_timestep: Final[bool]

forward(inputs)

Return X*W+b.

class deepmd.pt.model.network.network.Linear(d_in: int, d_out: int, bias: bool = True, init: str = 'default')

Bases: `torch.nn.Linear`

Applies a linear transformation to the incoming data: $y = xA^T + b$.

This module supports TensorFloat32.

On certain ROCm devices, when using float16 inputs this module will use different precision for backward.

Parameters

- **in_features** – size of each input sample
- **out_features** – size of each output sample
- **bias** – If set to `False`, the layer will not learn an additive bias. Default: `True`

Shape:

- Input: $(*, H_{in})$ where $*$ means any number of dimensions including none and $H_{in} =$ in_features.
- Output: $(*, H_{out})$ where all but the last dimension are the same shape as the input and $H_{out} =$ out_features.

weight

the learnable weights of the module of shape (out_features, in_features). The values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{\text{in_features}}$

bias

the learnable bias of the module of shape (out_features). If *bias* is `True`, the values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{1}{\text{in_features}}$

Examples:

```
>>> m = nn.Linear(20, 30)
>>> input = torch.randn(128, 20)
>>> output = m(input)
>>> print(output.size())
torch.Size([128, 30])
```

_trunc_normal_init(scale=1.0)

_glorot_uniform_init()

_zero_init(use_bias=True)

_normal_init()

class deepmd.pt.model.network.network.Transition(d_in, n, dropout=0.0)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

_transition(x)

forward(x: torch.Tensor) → torch.Tensor

class deepmd.pt.model.network.network.**Embedding**(num_embeddings: int, embedding_dim: int, padding_idx: int | None = None, dtype=torch.float64)

Bases: `torch.nn.Embedding`

A simple lookup table that stores embeddings of a fixed dictionary and size.

This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.

Parameters

- **num_embeddings** (*int*) – size of the dictionary of embeddings
- **embedding_dim** (*int*) – the size of each embedding vector
- **padding_idx** (*int, optional*) – If specified, the entries at `padding_idx` do not contribute to the gradient; therefore, the embedding vector at `padding_idx` is not updated during training, i.e. it remains as a fixed "pad". For a newly constructed Embedding, the embedding vector at `padding_idx` will default to all zeros, but can be updated to another value to be used as the padding vector.
- **max_norm** (*float, optional*) – If given, each embedding vector with norm larger than `max_norm` is renormalized to have norm `max_norm`.
- **norm_type** (*float, optional*) – The p of the p-norm to compute for the `max_norm` option. Default 2.
- **scale_grad_by_freq** (*bool, optional*) – If given, this will scale gradients by the inverse of frequency of the words in the mini-batch. Default `False`.
- **sparse** (*bool, optional*) – If `True`, gradient w.r.t. *weight* matrix will be a sparse tensor. See Notes for more details regarding sparse gradients.

weight
the learnable weights of the module of shape (num_embeddings, embedding_dim) initialized from $\mathcal{N}(0, 1)$

Type
Tensor

Shape:

- Input: $(*)$, IntTensor or LongTensor of arbitrary shape containing the indices to extract
- Output: $(*, H)$, where * is the input shape and H = embedding_dim

Note: Keep in mind that only a limited number of optimizers support sparse gradients: currently it's `optim.SGD` (CUDA and CPU), `optim.SparseAdam` (CUDA and CPU) and `optim.Adagrad` (CPU)

Note: When `max_norm` is not `None`, *`Embedding`*'s forward method will modify the *`weight`* tensor in-place. Since tensors needed for gradient computations cannot be modified in-place, performing a differentiable operation on `Embedding.weight` before calling *`Embedding`*'s forward method requires cloning `Embedding.weight` when `max_norm` is not `None`. For example:

```
n, d, m = 3, 5, 7
embedding = nn.Embedding(n, d, max_norm=True)
W = torch.randn((m, d), requires_grad=True)
idx = torch.tensor([1, 2])
a = embedding.weight.clone() @ W.t()  # weight must be cloned for this to be differentiable
b = embedding(idx) @ W.t()  # modifies weight in-place
out = (a.unsqueeze(0) + b.unsqueeze(1))
loss = out.sigmoid().prod()
loss.backward()
```

Examples:

```
>>> # an Embedding module containing 10 tensors of size 3
>>> embedding = nn.Embedding(10, 3)
>>> # a batch of 2 samples of 4 indices each
>>> input = torch.LongTensor([[1, 2, 4, 5], [4, 3, 2, 9]])
>>> # xdoctest: +IGNORE_WANT("non-deterministic")
>>> embedding(input)
tensor([[[-0.0251, -1.6902,  0.7172],
         [-0.6431,  0.0748,  0.6969],
         [ 1.4970,  1.3448, -0.9685],
         [-0.3677, -2.7265, -0.1685]],

        [[ 1.4970,  1.3448, -0.9685],
         [ 0.4362, -0.4004,  0.9400],
         [-0.6431,  0.0748,  0.6969],
         [ 0.9124, -2.3616,  1.1151]]])

>>> # example with padding_idx
>>> embedding = nn.Embedding(10, 3, padding_idx=0)
>>> input = torch.LongTensor([[0, 2, 0, 5]])
>>> embedding(input)
tensor([[[ 0.0000,  0.0000,  0.0000],
         [ 0.1535, -2.0309,  0.9315],
         [ 0.0000,  0.0000,  0.0000],
         [-0.1655,  0.9897,  0.0635]]])

>>> # example of changing `pad` vector
>>> padding_idx = 0
>>> embedding = nn.Embedding(3, 3, padding_idx=padding_idx)
>>> embedding.weight
Parameter containing:
tensor([[ 0.0000,  0.0000,  0.0000],
        [-0.7895, -0.7089, -0.0364],
        [ 0.6778,  0.5803,  0.2678]], requires_grad=True)
>>> with torch.no_grad():
...     embedding.weight[padding_idx] = torch.ones(3)
>>> embedding.weight
Parameter containing:
tensor([[ 1.0000,  1.0000,  1.0000],
        [-0.7895, -0.7089, -0.0364],
```

(continues on next page)

(continued from previous page)

```
        [ 0.6778,  0.5803,  0.2678]], requires_grad=True)
```

_normal_init(std=0.02)

class deepmd.pt.model.network.network.**NonLinearHead**(input_dim, out_dim, activation_fn, hidden=None)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(x)

class deepmd.pt.model.network.network.**NonLinear**(input, output_size, hidden=None)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
```

(continues on next page)

(continued from previous page)

```
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(x)

zero_init()

class deepmd.pt.model.network.network.**MaskLMHead**(embed_dim, output_dim, activation_fn, weight=None)

Bases: `torch.nn.Module`

Head for masked language modeling.

forward(features, masked_tokens: torch.Tensor | None = None, **kwargs)

class deepmd.pt.model.network.network.**ResidualDeep**(type_id, embedding_width, neuron, bias_atom_e, out_dim=1, resnet_dt=False)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(inputs)

Calculate decoded embedding for each atom.

Args: - inputs: Embedding net output per atom. Its shape is [nframes*nloc, self.embedding_width].

Returns

•torch.Tensor: **Output layer with shape** [nframes*nloc, **self.neuron**[-1]].

class deepmd.pt.model.network.network.**TypeEmbedNet**(type_nums, embed_dim, bavg=0.0, stddev=1.0, precision='default')

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(atype)

Parameters
atype – Type of each input, [nframes, nloc] or [nframes, nloc, nnei].

Returns

type_embedding:

share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

class deepmd.pt.model.network.network.TypeEmbedNetConsistent(*, ntypes: int, neuron: List[int], resnet_dt: bool = False, activation_function: str = 'tanh', precision: str = 'default', trainable: bool = True, seed: int | None = None, padding: bool = False)

Bases: `torch.nn.Module`

Type embedding network that is consistent with other backends.

Parameters

ntypes
[`int`] Number of atom types

neuron
[`list`[`int`]] Number of neurons in each hidden layers of the embedding net

resnet_dt
Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

activation_function
The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

trainable
If the weights of embedding net are trainable.

seed
Random seed for initializing the network parameters.

padding
Concat the zero padding to the output, as the default embedding of empty type.

forward(device: torch.device)

Caulate type embedding network.

Returns

type_embedding: **torch.Tensor**
Type embedding network.

classmethod deserialize(data: dict)

Deserialize the model.

Parameters

data
[`dict`] The serialized data

Returns

TypeEmbedNetConsistent
The deserialized model

serialize() → dict

Serialize the model.

Returns

dict
The serialized data

deepmd.pt.model.network.network.**gaussian**(x, mean, std: float)

class deepmd.pt.model.network.network.**GaussianKernel**(K=128, num_pair=512, std_width=1.0, start=0.0, stop=9.0)

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call to(), etc.

Note: As per the example above, an __init__() call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(x, atom_pair)

class deepmd.pt.model.network.network.**GaussianEmbedding**(rcut, kernel_num, num_pair, embed_dim, pair_embed_dim, sel, ntypes, atomic_sum_gbf)

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(coord_selected, atom_feature, edge_type_2dim, edge_feature)

Calculate decoded embedding for each atom. :param coord_selected: Clustered atom coordinates with shape [nframes*nloc, natoms, 3]. :param atom_feature: Previous calculated atomic features with shape [nframes*nloc, natoms, embed_dim]. :param edge_type_2dim: Edge index for gbf calculation with shape [nframes*nloc, natoms, natoms, 2]. :param edge_feature: Previous calculated edge features with shape [nframes*nloc, natoms, natoms, pair_dim].

Returns

atom_feature: **Updated atomic features with shape** [nframes*nloc, **natoms**, **embed_dim**].
attn_bias: **Updated edge features as attention bias with shape** [nframes*nloc, **natoms**, **natoms**, **pair_dim**].
delta_pos: **Delta position for** force/vector **prediction with shape** [nframes*nloc, **natoms**, **natoms**, 3].

class deepmd.pt.model.network.network.**NeighborWiseAttention**(layer_num, nnei, embed_dim, hidden_dim, dotr=False, do_mask=False, post_ln=True, ffn=False, ffn_embed_dim=1024, activation='tanh', scaling_factor=1.0, head_num=1, normalize=True, temperature=None, smooth=True)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(input_G, nei_mask, input_r: torch.Tensor | None = None, sw: torch.Tensor | None = None)

Parameters

- **input_G** – Input G, [nframes * nloc, nnei, embed_dim].
- **nei_mask** – neighbor mask, [nframes * nloc, nnei].
- **input_r** – normalized radial, [nframes, nloc, nei, 3].

Returns

out: **Output G**, [**nframes** * **nloc**, **nnei**, **embed_dim**]

class deepmd.pt.model.network.network.**NeighborWiseAttentionLayer**(nnei, embed_dim, hidden_dim, dotr=False, do_mask=False, post_ln=True, ffn=False, ffn_embed_dim=1024, activation='tanh', scaling_factor=1.0, head_num=1, normalize=True, temperature=None, smooth=True)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

ffn: Final[bool]

forward(x, nei_mask, input_r: torch.Tensor | None = None, sw: torch.Tensor | None = None)

class deepmd.pt.model.network.network.**GatedSelfAttetion**(nnei, embed_dim, hidden_dim, dotr=False, do_mask=False, scaling_factor=1.0, head_num=1, normalize=True, temperature=None, bias=True, smooth=True)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(query, nei_mask, input_r: torch.Tensor | None = None, sw: torch.Tensor | None = None, attnw_shift: float = 20.0)

Parameters

- **query** – input G, [nframes * nloc, nnei, embed_dim].
- **nei_mask** – neighbor mask, [nframes * nloc, nnei].
- **input_r** – normalized radial, [nframes, nloc, nei, 3].

Returns

type_embedding:

class deepmd.pt.model.network.network.LocalSelfMultiheadAttention(feature_dim, attn_head, scaling_factor=1.0)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(query, attn_bias: torch.Tensor | None = None, nlist_mask: torch.Tensor | None = None, nlist: torch.Tensor | None = None, return_attn=True)

class deepmd.pt.model.network.network.**NodeTaskHead**(embed_dim: int, pair_dim: int, num_head: int)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

zero_init()

forward(query: Tensor, pair: Tensor, delta_pos: Tensor, attn_mask: Tensor = None) → Tensor

class deepmd.pt.model.network.network.**EnergyHead**(input_dim, output_dim)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(x)

class deepmd.pt.model.network.network.**OuterProduct**(d_atom, d_pair, d_hid=32)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

_opm(a, b)

forward(m: torch.Tensor, nlist: torch.Tensor, op_mask: float, op_norm: float) → torch.Tensor

class deepmd.pt.model.network.network.Attention(q_dim: int, k_dim: int, v_dim: int, head_dim: int, num_heads: int, gating: bool = False, dropout: float = 0.0)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(q: torch.Tensor, k: torch.Tensor, v: torch.Tensor, bias: torch.Tensor, mask: torch.Tensor = None) → torch.Tensor

class deepmd.pt.model.network.network.AtomAttention(q_dim: int, k_dim: int, v_dim: int, pair_dim: int, head_dim: int, num_heads: int, gating: bool = False, dropout: float = 0.0)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(q: torch.Tensor, k: torch.Tensor, v: torch.Tensor, nlist: torch.Tensor, pair: torch.Tensor, mask: torch.Tensor = None) → torch.Tensor

class deepmd.pt.model.network.network.**TriangleMultiplication**(d_pair, d_hid)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(z: torch.Tensor, mask: torch.Tensor | None = None) → torch.Tensor

class deepmd.pt.model.network.network.**EvoformerEncoderLayer**(feature_dim: int = 768, ffn_dim: int = 2048, attn_head: int = 8, activation_fn: str = 'gelu', post_ln: bool = False)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(x, attn_bias: torch.Tensor | None = None, nlist_mask: torch.Tensor | None = None, nlist: torch.Tensor | None = None, return_attn=True)

class deepmd.pt.model.network.network.Evoformer2bEncoder(nnei: int, layer_num: int = 6, attn_head: int = 8, atomic_dim: int = 1024, pair_dim: int = 100, feature_dim: int = 1024, ffn_dim: int = 2048, post_ln: bool = False, final_layer_norm: bool = True, final_head_layer_norm: bool = False, emb_layer_norm: bool = False, atomic_residual: bool = False, evo_residual: bool = False, residual_factor: float = 1.0, activation_function: str = 'gelu')

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call to(), etc.

Note: As per the example above, an __init__() call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(atomic_rep, pair_rep, nlist, nlist_type, nlist_mask)

Encoder the atomic and pair representations.

Args: - atomic_rep: Atomic representation with shape [nframes, nloc, atomic_dim]. - pair_rep: Pair representation with shape [nframes, nloc, nnei, pair_dim]. - nlist: Neighbor list with shape [nframes, nloc, nnei]. - nlist_type: Neighbor types with shape [nframes, nloc, nnei]. - nlist_mask: Neighbor mask with shape [nframes, nloc, nnei], False if blank.

Returns

- atomic_rep: **Atomic representation after encoder with shape** [**nframes**, **nloc**, **feature_dim**].

- transformed_atomic_rep: **Transformed atomic representation after encoder with shape** [**nframes**, **nloc**, **atomic_dim**].
- pair_rep: **Pair representation after encoder with shape** [**nframes**, **nloc**, **nnei**, **attn_head**].
- delta_pair_rep: **Delta pair representation after encoder with shape** [**nframes**, **nloc**, **nnei**, **attn_head**].
- norm_x: **Normalization loss of** atomic_rep.
- norm_delta_pair_rep: **Normalization loss of** delta_pair_rep.

class deepmd.pt.model.network.network.**Evoformer3bEncoderLayer**(nnei, embedding_dim: int = 768, pair_dim: int = 64, pair_hidden_dim: int = 32, ffn_embedding_dim: int = 3072, num_attention_heads: int = 8, dropout: float = 0.1, droppath_prob: float = 0.0, pair_dropout: float = 0.25, attention_dropout: float = 0.1, activation_dropout: float = 0.1, pre_ln: bool = True, tri_update: bool = True)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

update_pair(x, pair, nlist, op_mask, op_norm)

shared_dropout(x, shared_dim, dropout)

forward(x: torch.Tensor, pair: torch.Tensor, nlist: torch.Tensor = None, attn_mask: torch.Tensor | None = None, pair_mask: torch.Tensor | None = None, op_mask: float = 1.0, op_norm: float = 1.0)

Encoder the atomic and pair representations.

Args: - x: Atomic representation with shape [ncluster, natoms, embed_dim]. - pair: Pair representation with shape [ncluster, natoms, natoms, pair_dim]. - attn_mask: Attention mask with shape [ncluster, head, natoms, natoms]. - pair_mask: Neighbor mask with shape [ncluster, natoms, natoms].

class deepmd.pt.model.network.network.**Evoformer3bEncoder**(nnei, layer_num=6, attn_head=8, atomic_dim=768, pair_dim=64, pair_hidden_dim=32, ffn_embedding_dim=3072, dropout: float = 0.1, droppath_prob: float = 0.0, pair_dropout: float = 0.25, attention_dropout: float = 0.1, activation_dropout: float = 0.1, pre_ln: bool = True, tri_update: bool = True, **kwargs)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(x, pair, attn_mask=None, pair_mask=None, atom_mask=None)

Encoder the atomic and pair representations.

Parameters

- **x** – Atomic representation with shape [ncluster, natoms, atomic_dim].
- **pair** – Pair representation with shape [ncluster, natoms, natoms, pair_dim].
- **attn_mask** – Attention mask (with -inf for softmax) with shape [ncluster, head, natoms, natoms].
- **pair_mask** – Pair mask (with 1 for real atom pair and 0 for padding) with shape [ncluster, natoms, natoms].
- **atom_mask** – Atom mask (with 1 for real atom and 0 for padding) with shape [ncluster, natoms].

Returns

x: **Atomic representation with shape** [**ncluster**, **natoms**, **atomic_dim**].
pair: **Pair representation with shape** [**ncluster**, **natoms**, **natoms**, **pair_dim**].

deepmd.pt.model.task

Submodules

deepmd.pt.model.task.atten_lcc

Module Contents

Classes

FittingNetAttenLcc	Base class for all neural network modules.

class deepmd.pt.model.task.atten_lcc.**FittingNetAttenLcc**(embedding_width, bias_atom_e, pair_embed_dim, attention_heads, **kwargs)

Bases: *deepmd.pt.model.task.fitting.Fitting*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)
```

(continues on next page)

(continued from previous page)

```
    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
: **training** (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(output, pair, delta_pos, atype, nframes, nloc)

deepmd.pt.model.task.base_fitting

Module Contents

deepmd.pt.model.task.base_fitting.**BaseFitting**

deepmd.pt.model.task.denoise

Module Contents

Classes

DenoiseNet	Base class for all neural network modules.

class deepmd.pt.model.task.denoise.**DenoiseNet**(feature_dim, ntypes, attn_head=8, prefactor=[0.5, 0.5], activation_function='gelu', **kwargs)

Bases: *deepmd.pt.model.task.fitting.Fitting*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
```

(continues on next page)

(continued from previous page)

```
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

output_def()

Returns the output def of the fitting net.

forward(pair_weights, diff, nlist_mask, features, sw, masked_tokens: torch.Tensor | None = None)

Calculate the updated coord. Args: - coord: Input noisy coord with shape [nframes, nloc, 3]. - pair_weights: Input pair weights with shape [nframes, nloc, nnei, head]. - diff: Input pair relative coord list with shape [nframes, nloc, nnei, 3]. - nlist_mask: Input nlist mask with shape [nframes, nloc, nnei].

Returns

- denoised_coord: **Denoised updated coord with shape** [**nframes**, **nloc**, 3].

deepmd.pt.model.task.dipole

Module Contents

Classes

DipoleFittingNet	Construct a dipole fitting net.

Attributes

log

deepmd.pt.model.task.dipole.log

class deepmd.pt.model.task.dipole.DipoleFittingNet(ntypes: int, dim_descrpt: int, embedding_width: int, neuron: List[int] = [128, 128, 128], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, mixed_types: bool = True, rcond: float | None = None, seed: int | None = None, exclude_types: List[int] = [], r_differentiable: bool = True, c_differentiable: bool = True, **kwargs)

Bases: *deepmd.pt.model.task.fitting.GeneralFitting*

Construct a dipole fitting net.

Parameters

ntypes
[int] Element count.

dim_descrpt
[int] Embedding width per atom.

embedding_width
[int] The dimension of rotation matrix, m1.

neuron
[List[int]] Number of neurons in each hidden layers of the fitting net.

resnet_dt
[bool] Using time-step in the ResNet construction.

numb_fparam
[int] Number of frame parameters.

numb_aparam
[int] Number of atomic parameters.

activation_function
[str] Activation function.

precision
[str] Numerical precision.

mixed_types
[bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

rcond
[float, optional] The condition number for the regression of atomic energy.

seed
[int, optional] Random seed.

r_differentiable
If the variable is differentiated with respect to coordinates of atoms. Only reduciable variable are differentiable.

c_differentiable
If the variable is differentiated with respect to the cell tensor (pbc case). Only reduciable variable are differentiable.

exclude_types: List[int]

_net_out_dim()

Set the FittingNet output dim.

serialize() → dict

Serialize the fitting to dict.

classmethod deserialize(data: dict) → deepmd.pt.model.task.fitting.GeneralFitting

Deserialize the fitting.

Parameters

data
[dict] The serialized data

Returns

BF
The deserialized fitting

output_def() → deepmd.dpmodel.FittingOutputDef

Returns the output def of the fitting net.

compute_output_stats(merged: Callable[[], List[dict]] | List[dict], stat_file_path: deepmd.utils.path.DPPath | None = None)

Compute the output statistics (e.g. energy bias) for the fitting net from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

stat_file_path
[Optional[DPPath]] The path to the stat file.

forward(descriptor: torch.Tensor, atype: torch.Tensor, gr: torch.Tensor | None = None, g2: torch.Tensor | None = None, h2: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None)

deepmd.pt.model.task.dos

Module Contents

Classes

DOSFittingNet	Construct a fitting net for energy.

Attributes

dtype
device
log

deepmd.pt.model.task.dos.**dtype**

deepmd.pt.model.task.dos.**device**

deepmd.pt.model.task.dos.**log**

class deepmd.pt.model.task.dos.**DOSFittingNet**(ntypes: int, dim_descrpt: int, numb_dos: int = 300, neuron: List[int] = [128, 128, 128], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, rcond: float | None = None, bias_dos: torch.Tensor | None = None, trainable: bool | List[bool] = True, seed: int | None = None, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, exclude_types: List[int] = [], mixed_types: bool = True)

Bases: deepmd.pt.model.task.ener.InvarFitting

Construct a fitting net for energy.

Parameters

var_name
: [str] The atomic property to fit, 'energy', 'dipole', and 'polar'.

ntypes
: [int] Element count.

dim_descrpt
: [int] Embedding width per atom.

dim_out
: [int] The output dimension of the fitting net.

neuron
: [List[int]] Number of neurons in each hidden layers of the fitting net.

bias_atom_e
: [torch.Tensor, optional] Average enery per atom for each element.

resnet_dt
: [bool] Using time-step in the ResNet construction.

numb_fparam
: [int] Number of frame parameters.

numb_aparam
: [int] Number of atomic parameters.

activation_function
: [str] Activation function.

precision
[str] Numerical precision.

mixed_types
[bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

rcond
[float, optional] The condition number for the regression of atomic energy.

seed
[int, optional] Random seed.

exclude_types: List[int]
Atomic contributions of the excluded atom types are set zero.

atom_ener: List[Optional[torch.Tensor]], optional
Specifying atomic energy contribution in vacuum. The value is a list specifying the bias. the elements can be None or np.array of output shape. For example: [None, [2.]] means type 0 is not set, type 1 is set to [2.] The set_davg_zero key in the descrptor should be set.

exclude_types: List[int]

output_def() → deepmd.dpmodel.FittingOutputDef
Returns the output def of the fitting net.

classmethod deserialize(data: dict) → DOSFittingNet
Deserialize the fitting.

Parameters

data
[dict] The serialized data

Returns

BF
The deserialized fitting

serialize() → dict
Serialize the fitting to dict.

deepmd.pt.model.task.ener

Module Contents

Classes

EnergyFittingNet	Construct a fitting net for energy.
EnergyFittingNetDirect	Base class for all neural network modules.

Attributes

dtype
device
log

deepmd.pt.model.task.ener.**dtype**

deepmd.pt.model.task.ener.**device**

deepmd.pt.model.task.ener.**log**

class deepmd.pt.model.task.ener.**EnergyFittingNet**(ntypes: int, dim_descrpt: int, neuron: List[int] = [128, 128, 128], bias_atom_e: torch.Tensor | None = None, resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, mixed_types: bool = True, **kwargs)

Bases: *deepmd.pt.model.task.invar_fitting.InvarFitting*

Construct a fitting net for energy.

Parameters

var_name
: [str] The atomic property to fit, 'energy', 'dipole', and 'polar'.

ntypes
: [int] Element count.

dim_descrpt
: [int] Embedding width per atom.

dim_out
: [int] The output dimension of the fitting net.

neuron
: [List[int]] Number of neurons in each hidden layers of the fitting net.

bias_atom_e
: [torch.Tensor, optional] Average enery per atom for each element.

resnet_dt
: [bool] Using time-step in the ResNet construction.

numb_fparam
: [int] Number of frame parameters.

numb_aparam
: [int] Number of atomic parameters.

activation_function
: [str] Activation function.

precision
[str] Numerical precision.

mixed_types
[bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

rcond
[float, optional] The condition number for the regression of atomic energy.

seed
[int, optional] Random seed.

exclude_types: List[int]
Atomic contributions of the excluded atom types are set zero.

atom_ener: List[Optional[torch.Tensor]], optional
Specifying atomic energy contribution in vacuum. The value is a list specifying the bias. the elements can be None or np.array of output shape. For example: [None, [2.]] means type 0 is not set, type 1 is set to [2.] The set_davg_zero key in the descrptor should be set.

exclude_types: List[int]

classmethod deserialize(data: dict) → deepmd.pt.model.task.fitting.GeneralFitting

Deserialize the fitting.

Parameters

data
[dict] The serialized data

Returns

BF
The deserialized fitting

serialize() → dict

Serialize the fitting to dict.

class deepmd.pt.model.task.ener.EnergyFittingNetDirect(ntypes, dim_descrpt, neuron, bias_atom_e=None, out_dim=1, resnet_dt=True, use_tebd=True, return_energy=False, **kwargs)

Bases: *deepmd.pt.model.task.fitting.Fitting*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)
```

(continues on next page)

(continued from previous page)

```
    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

output_def()

Returns the output def of the fitting net.

abstract serialize() → dict

Serialize the obj to dict.

abstract deserialize() → EnergyFittingNetDirect

Deserialize the fitting.

Parameters

data
[dict] The serialized data

Returns

BF
The deserialized fitting

forward(inputs: torch.Tensor, atype: torch.Tensor, gr: torch.Tensor | None = None, g2: torch.Tensor | None = None, h2: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None) → Tuple[torch.Tensor, None]

Based on embedding net output, alculate total energy.

Args: - inputs: Embedding matrix. Its shape is [nframes, natoms[0], self.dim_descrpt]. - natoms: Tell atom count and element count. Its shape is [2+self.ntypes].

Returns

- torch.Tensor: **Total energy with shape** [**nframes**, **natoms**[0]].

deepmd.pt.model.task.fitting

Module Contents

Classes

Fitting	Base class for all neural network modules.
GeneralFitting	Construct a general fitting net.

Attributes

dtype
device
log

deepmd.pt.model.task.fitting.**dtype**

deepmd.pt.model.task.fitting.**device**

deepmd.pt.model.task.fitting.**log**

class deepmd.pt.model.task.fitting.**Fitting**(*args, **kwargs)

Bases: `torch.nn.Module`, *deepmd.pt.model.task.base_fitting.BaseFitting*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

share_params(base_class, shared_level, resume=False)
Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

class deepmd.pt.model.task.fitting.GeneralFitting(var_name: str, ntypes: int, dim_descrpt: int, neuron: List[int] = [128, 128, 128], bias_atom_e: torch.Tensor | None = None, resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, mixed_types: bool = True, rcond: float | None = None, seed: int | None = None, exclude_types: List[int] = [], trainable: bool | List[bool] = True, remove_vaccum_contribution: List[bool] | None = None, **kwargs)

Bases: *Fitting*

Construct a general fitting net.

Parameters

var_name
[str] The atomic property to fit, 'energy', 'dipole', and 'polar'.

ntypes
[int] Element count.

dim_descrpt
[int] Embedding width per atom.

dim_out
[int] The output dimension of the fitting net.

neuron
[List[int]] Number of neurons in each hidden layers of the fitting net.

bias_atom_e
[torch.Tensor, optional] Average enery per atom for each element.

resnet_dt
[bool] Using time-step in the ResNet construction.

numb_fparam
[int] Number of frame parameters.

numb_aparam
[int] Number of atomic parameters.

activation_function
: [str] Activation function.

precision
: [str] Numerical precision.

mixed_types
: [bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

rcond
: [float, optional] The condition number for the regression of atomic energy.

seed
: [int, optional] Random seed.

exclude_types: List[int]
: Atomic contributions of the excluded atom types are set zero.

trainable
: [Union[List[bool], bool]] If the parameters in the fitting net are trainable. Now this only supports setting all the parameters in the fitting net at one state. When in List[bool], the trainable will be True only if all the boolean parameters are True.

remove_vaccum_contribution: List[bool], optional
: Remove vaccum contribution before the bias is added. The list assigned each type. For mixed_types provide [True], otherwise it should be a list of the same length as ntypes signaling if or not removing the vaccum contribution for the atom types in the list.

exclude_types: List[int]

reinit_exclude(exclude_types: List[int] = [])

serialize() → dict

Serialize the fitting to dict.

classmethod deserialize(data: dict) → GeneralFitting

Deserialize the fitting.

Parameters

data
: [dict] The serialized data

Returns

BF
: The deserialized fitting

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this atomic model.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this atomic model.

get_sel_type() → List[int]

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

__setitem__(key, value)

__getitem__(key)

abstract _net_out_dim()

Set the FittingNet output dim.

_extend_f_avg_std(xx: torch.Tensor, nb: int) → torch.Tensor

_extend_a_avg_std(xx: torch.Tensor, nb: int, nloc: int) → torch.Tensor

_forward_common(descriptor: torch.Tensor, atype: torch.Tensor, gr: torch.Tensor | None = None, g2: torch.Tensor | None = None, h2: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None)

deepmd.pt.model.task.invar_fitting

Module Contents

Classes

InvarFitting	Construct a fitting net for energy.

Attributes

dtype
device
log

deepmd.pt.model.task.invar_fitting.**dtype**

deepmd.pt.model.task.invar_fitting.**device**

deepmd.pt.model.task.invar_fitting.**log**

class deepmd.pt.model.task.invar_fitting.**InvarFitting**(var_name: str, ntypes: int, dim_descrpt: int, dim_out: int, neuron: List[int] = [128, 128, 128], bias_atom_e: torch.Tensor | None = None, resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, mixed_types: bool = True, rcond: float | None = None, seed: int | None = None, exclude_types: List[int] = [], atom_ener: List[torch.Tensor | None] | None = None, **kwargs)

Bases: *deepmd.pt.model.task.fitting.GeneralFitting*

Construct a fitting net for energy.

Parameters

var_name
: [str] The atomic property to fit, 'energy', 'dipole', and 'polar'.

ntypes
: [int] Element count.

dim_descrpt
: [int] Embedding width per atom.

dim_out
: [int] The output dimension of the fitting net.

neuron
: [List[int]] Number of neurons in each hidden layers of the fitting net.

bias_atom_e
: [torch.Tensor, optional] Average enery per atom for each element.

resnet_dt
: [bool] Using time-step in the ResNet construction.

numb_fparam
: [int] Number of frame parameters.

numb_aparam
: [int] Number of atomic parameters.

activation_function
: [str] Activation function.

precision
: [str] Numerical precision.

mixed_types
: [bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

rcond
: [float, optional] The condition number for the regression of atomic energy.

seed
: [int, optional] Random seed.

exclude_types: List[int]
: Atomic contributions of the excluded atom types are set zero.

atom_ener: List[Optional[torch.Tensor]], optional
: Specifying atomic energy contribution in vacuum. The value is a list specifying the bias. the elements can be None or np.array of output shape. For example: [None, [2.]] means type 0 is not set, type 1 is set to [2.] The set_davg_zero key in the descrptor should be set.

exclude_types: List[int]

_net_out_dim()

Set the FittingNet output dim.

serialize() → dict

Serialize the fitting to dict.

classmethod deserialize(data: dict) → deepmd.pt.model.task.fitting.GeneralFitting

Deserialize the fitting.

Parameters

data
[dict] The serialized data

Returns

BF
The deserialized fitting

output_def() → deepmd.dpmodel.FittingOutputDef

Returns the output def of the fitting net.

forward(descriptor: torch.Tensor, atype: torch.Tensor, gr: torch.Tensor | None = None, g2: torch.Tensor | None = None, h2: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None)

Based on embedding net output, alculate total energy.

Args: - inputs: Embedding matrix. Its shape is [nframes, natoms[0], self.dim_descrpt]. - natoms: Tell atom count and element count. Its shape is [2+self.ntypes].

Returns

- torch.Tensor: **Total energy with shape** [**nframes**, **natoms**[0]].

deepmd.pt.model.task.polarizability

Module Contents

Classes

PolarFittingNet	Construct a polar fitting net.

Attributes

log

deepmd.pt.model.task.polarizability.log

class deepmd.pt.model.task.polarizability.PolarFittingNet(ntypes: int, dim_descrpt: int, embedding_width: int, neuron: List[int] = [128, 128, 128], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, mixed_types: bool = True, rcond: float | None = None, seed: int | None = None, exclude_types: List[int] = [], fit_diag: bool = True, scale: List[float] | float | None = None, shift_diag: bool = True, **kwargs)

Bases: *deepmd.pt.model.task.fitting.GeneralFitting*

Construct a polar fitting net.

Parameters

ntypes
[int] Element count.

dim_descrpt
[int] Embedding width per atom.

embedding_width
[int] The dimension of rotation matrix, m1.

neuron
[List[int]] Number of neurons in each hidden layers of the fitting net.

resnet_dt
[bool] Using time-step in the ResNet construction.

numb_fparam
[int] Number of frame parameters.

numb_aparam
[int] Number of atomic parameters.

activation_function
[str] Activation function.

precision
[str] Numerical precision.

mixed_types
[bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

rcond
[float, optional] The condition number for the regression of atomic energy.

seed
[int, optional] Random seed.

fit_diag
[bool] Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.

scale
: [List[float]] The output of the fitting net (polarizability matrix) for type i atom will be scaled by scale[i]

shift_diag
: [bool] Whether to shift the diagonal part of the polarizability matrix. The shift operation is carried out after scale.

exclude_types: List[int]

_net_out_dim()
Set the FittingNet output dim.

__setitem__(key, value)

__getitem__(key)

serialize() → dict
Serialize the fitting to dict.

classmethod deserialize(data: dict) → deepmd.pt.model.task.fitting.GeneralFitting
Deserialize the fitting.

Parameters

data
: [dict] The serialized data

Returns

BF
: The deserialized fitting

output_def() → deepmd.dpmodel.FittingOutputDef
Returns the output def of the fitting net.

forward(descriptor: torch.Tensor, atype: torch.Tensor, gr: torch.Tensor | None = None, g2: torch.Tensor | None = None, h2: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None)

deepmd.pt.model.task.task

deepmd.pt.model.task.type_predict

Module Contents

Classes

TypePredictNet	Base class for all neural network modules.

class deepmd.pt.model.task.type_predict.TypePredictNet(feature_dim, ntypes, activation_function='gelu', **kwargs)

Bases: *deepmd.pt.model.task.Fitting*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(features, masked_tokens: torch.Tensor | None = None)

Calculate the predicted logits. Args: - features: Input features with shape [nframes, nloc, feature_dim]. - masked_tokens: Input masked tokens with shape [nframes, nloc].

Returns

- logits: **Predicted probs with shape** [**nframes**, **nloc**, **ntypes**].

Package Contents

Classes

FittingNetAttenLcc	Base class for all neural network modules.
DenoiseNet	Base class for all neural network modules.
DipoleFittingNet	Construct a dipole fitting net.
EnergyFittingNet	Construct a fitting net for energy.
EnergyFittingNetDirect	Base class for all neural network modules.
Fitting	Base class for all neural network modules.
PolarFittingNet	Construct a polar fitting net.
TypePredictNet	Base class for all neural network modules.

Attributes

BaseFitting

class deepmd.pt.model.task.**FittingNetAttenLcc**(embedding_width, bias_atom_e, pair_embed_dim, attention_heads, **kwargs)

Bases: *deepmd.pt.model.task.fitting.Fitting*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(output, pair, delta_pos, atype, nframes, nloc)

deepmd.pt.model.task.**BaseFitting**

class deepmd.pt.model.task.**DenoiseNet**(feature_dim, ntypes, attn_head=8, prefactor=[0.5, 0.5], activation_function='gelu', **kwargs)

Bases: *deepmd.pt.model.task.fitting.Fitting*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

output_def()

Returns the output def of the fitting net.

forward(pair_weights, diff, nlist_mask, features, sw, masked_tokens: torch.Tensor | None = None)

Calculate the updated coord. Args: - coord: Input noisy coord with shape [nframes, nloc, 3]. - pair_weights: Input pair weights with shape [nframes, nloc, nnei, head]. - diff: Input pair relative coord list with shape [nframes, nloc, nnei, 3]. - nlist_mask: Input nlist mask with shape [nframes, nloc, nnei].

Returns

- denoised_coord: **Denoised updated coord with shape** [**nframes**, **nloc**, 3].

class deepmd.pt.model.task.DipoleFittingNet(ntypes: int, dim_descrpt: int, embedding_width: int, neuron: List[int] = [128, 128, 128], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, mixed_types: bool = True, rcond: float | None = None, seed: int | None = None, exclude_types: List[int] = [], r_differentiable: bool = True, c_differentiable: bool = True, **kwargs)

Bases: *deepmd.pt.model.task.fitting.GeneralFitting*

Construct a dipole fitting net.

Parameters

ntypes
[int] Element count.

dim_descrpt
[int] Embedding width per atom.

embedding_width
: [int] The dimension of rotation matrix, m1.

neuron
: [List[int]] Number of neurons in each hidden layers of the fitting net.

resnet_dt
: [bool] Using time-step in the ResNet construction.

numb_fparam
: [int] Number of frame parameters.

numb_aparam
: [int] Number of atomic parameters.

activation_function
: [str] Activation function.

precision
: [str] Numerical precision.

mixed_types
: [bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

rcond
: [float, optional] The condition number for the regression of atomic energy.

seed
: [int, optional] Random seed.

r_differentiable
: If the variable is differentiated with respect to coordinates of atoms. Only reduciable variable are differentiable.

c_differentiable
: If the variable is differentiated with respect to the cell tensor (pbc case). Only reduciable variable are differentiable.

exclude_types: List[int]

_net_out_dim()
Set the FittingNet output dim.

serialize() → dict
Serialize the fitting to dict.

classmethod deserialize(data: dict) → deepmd.pt.model.task.fitting.GeneralFitting
Deserialize the fitting.

Parameters

data
: [dict] The serialized data

Returns

BF
: The deserialized fitting

output_def() → deepmd.dpmodel.FittingOutputDef
Returns the output def of the fitting net.

compute_output_stats(merged: Callable[[], List[dict]] | List[dict], stat_file_path: deepmd.utils.path.DPPath | None = None)

Compute the output statistics (e.g. energy bias) for the fitting net from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

stat_file_path
[Optional[DPPath]] The path to the stat file.

forward(descriptor: torch.Tensor, atype: torch.Tensor, gr: torch.Tensor | None = None, g2: torch.Tensor | None = None, h2: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None)

class deepmd.pt.model.task.**EnergyFittingNet**(ntypes: int, dim_descrpt: int, neuron: List[int] = [128, 128, 128], bias_atom_e: torch.Tensor | None = None, resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, mixed_types: bool = True, **kwargs)

Bases: *deepmd.pt.model.task.invar_fitting.InvarFitting*

Construct a fitting net for energy.

Parameters

var_name
[str] The atomic property to fit, 'energy', 'dipole', and 'polar'.

ntypes
[int] Element count.

dim_descrpt
[int] Embedding width per atom.

dim_out
[int] The output dimension of the fitting net.

neuron
[List[int]] Number of neurons in each hidden layers of the fitting net.

bias_atom_e
[torch.Tensor, optional] Average enery per atom for each element.

resnet_dt
[bool] Using time-step in the ResNet construction.

numb_fparam
[int] Number of frame parameters.

numb_aparam
: [int] Number of atomic parameters.

activation_function
: [str] Activation function.

precision
: [str] Numerical precision.

mixed_types
: [bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

rcond
: [float, optional] The condition number for the regression of atomic energy.

seed
: [int, optional] Random seed.

exclude_types: List[int]
: Atomic contributions of the excluded atom types are set zero.

atom_ener: List[Optional[torch.Tensor]], optional
: Specifying atomic energy contribution in vacuum. The value is a list specifying the bias. the elements can be None or np.array of output shape. For example: [None, [2.]] means type 0 is not set, type 1 is set to [2.] The set_davg_zero key in the descrptor should be set.

exclude_types: List[int]

classmethod deserialize(data: dict) → deepmd.pt.model.task.fitting.GeneralFitting

Deserialize the fitting.

Parameters

data
: [dict] The serialized data

Returns

BF
: The deserialized fitting

serialize() → dict

Serialize the fitting to dict.

class deepmd.pt.model.task.EnergyFittingNetDirect(ntypes, dim_descrpt, neuron, bias_atom_e=None, out_dim=1, resnet_dt=True, use_tebd=True, return_energy=False, **kwargs)

Bases: *deepmd.pt.model.task.fitting.Fitting*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F
```

(continues on next page)

(continued from previous page)

```
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

output_def()

Returns the output def of the fitting net.

abstract serialize() → dict

Serialize the obj to dict.

abstract deserialize() → EnergyFittingNetDirect

Deserialize the fitting.

Parameters

data
[dict] The serialized data

Returns

BF
The deserialized fitting

forward(inputs: torch.Tensor, atype: torch.Tensor, gr: torch.Tensor | None = None, g2: torch.Tensor | None = None, h2: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None) → Tuple[torch.Tensor, None]

Based on embedding net output, alculate total energy.

Args: - inputs: Embedding matrix. Its shape is [nframes, natoms[0], self.dim_descrpt]. - natoms: Tell atom count and element count. Its shape is [2+self.ntypes].

Returns

- torch.Tensor: **Total energy with shape** [**nframes**, **natoms**[0]].

class deepmd.pt.model.task.Fitting(*args, **kwargs)

Bases: `torch.nn.Module`, *deepmd.pt.model.task.base_fitting.BaseFitting*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

share_params(base_class, shared_level, resume=False)

Share the parameters of self to the base_class with shared_level during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

class deepmd.pt.model.task.PolarFittingNet(ntypes: int, dim_descrpt: int, embedding_width: int, neuron: List[int] = [128, 128, 128], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, activation_function: str = 'tanh', precision: str = DEFAULT_PRECISION, mixed_types: bool = True, rcond: float | None = None, seed: int | None = None, exclude_types: List[int] = [], fit_diag: bool = True, scale: List[float] | float | None = None, shift_diag: bool = True, **kwargs)

Bases: *deepmd.pt.model.task.fitting.GeneralFitting*

Construct a polar fitting net.

Parameters

ntypes
[int] Element count.

dim_descrpt
[int] Embedding width per atom.

embedding_width
[int] The dimension of rotation matrix, m1.

neuron
[List[int]] Number of neurons in each hidden layers of the fitting net.

resnet_dt
[bool] Using time-step in the ResNet construction.

numb_fparam
[int] Number of frame parameters.

numb_aparam
[int] Number of atomic parameters.

activation_function
[str] Activation function.

precision
[str] Numerical precision.

mixed_types
[bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

rcond
[float, optional] The condition number for the regression of atomic energy.

seed
[int, optional] Random seed.

fit_diag
[bool] Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.

scale
[List[float]] The output of the fitting net (polarizability matrix) for type i atom will be scaled by scale[i]

shift_diag
[bool] Whether to shift the diagonal part of the polarizability matrix. The shift operation is carried out after scale.

exclude_types: List[int]

_net_out_dim()
Set the FittingNet output dim.

__setitem__(key, value)

__getitem__(key)

serialize() → dict
Serialize the fitting to dict.

classmethod deserialize(data: dict) → deepmd.pt.model.task.fitting.GeneralFitting
Deserialize the fitting.

Parameters

data
[dict] The serialized data

Returns

BF
The deserialized fitting

output_def() → deepmd.dpmodel.FittingOutputDef
Returns the output def of the fitting net.

forward(descriptor: torch.Tensor, atype: torch.Tensor, gr: torch.Tensor | None = None, g2: torch.Tensor | None = None, h2: torch.Tensor | None = None, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None)

class deepmd.pt.model.task.TypePredictNet(feature_dim, ntypes, activation_function='gelu', **kwargs)

Bases: *deepmd.pt.model.task.Fitting*

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables
training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(features, masked_tokens: torch.Tensor | None = None)
Calculate the predicted logits. Args: - features: Input features with shape [nframes, nloc, feature_dim]. - masked_tokens: Input masked tokens with shape [nframes, nloc].

Returns

- logits: **Predicted probs with shape** [**nframes**, **nloc**, **ntypes**].

deepmd.pt.optimizer

Submodules

deepmd.pt.optimizer.KFWrapper

Module Contents

Classes

KFOptimizerWrapper	

class deepmd.pt.optimizer.KFWrapper.**KFOptimizerWrapper**(model: torch.nn.Module, optimizer: torch.optim.optimizer.Optimizer, atoms_selected: int, atoms_per_group: int, is_distributed: bool = False)

update_energy(inputs: dict, Etot_label: torch.Tensor, update_prefactor: float = 1) → None

update_force(inputs: dict, Force_label: torch.Tensor, update_prefactor: float = 1) → None

update_denoise_coord(inputs: dict, clean_coord: torch.Tensor, update_prefactor: float = 1, mask_loss_coord: bool = True, coord_mask: torch.Tensor = None) → None

__sample(atoms_selected: int, atoms_per_group: int, natoms: int) → numpy.ndarray

deepmd.pt.optimizer.LKF

Module Contents

Classes

LKFOptimizer	Base class for all optimizers.

Functions

distribute_indices(total_length, num_workers)	

deepmd.pt.optimizer.LKF.**distribute_indices**(total_length, num_workers)

class deepmd.pt.optimizer.LKF.LKFOptimizer(params, kalman_lambda=0.98, kalman_nue=0.9987, block_size=5120)

Bases: torch.optim.optimizer.Optimizer

Base class for all optimizers.

Warning: Parameters need to be specified as collections that have a deterministic ordering that is consistent between runs. Examples of objects that don't satisfy those properties are sets and iterators over values of dictionaries.

Parameters

- **params** (*iterable*) – an iterable of torch.Tensor s or dict s. Specifies what Tensors should be optimized.
- **defaults** – (dict): a dict containing default values of optimization options (used when a parameter group doesn't specify them).

__init_P()

__get_blocksize()

__get_nue()

__split_weights(weight)

__update(H, error, weights)

set_grad_prefactor(grad_prefactor)

step(error)

Performs a single optimization step (parameter update).

Parameters

closure (*Callable*) – A closure that reevaluates the model and returns the loss. Optional for most optimizers.

Note: Unless otherwise specified, this function should not modify the .grad field of the parameters.

get_device_id(index)

Package Contents

Classes

KFOptimizerWrapper	
LKFOptimizer	Base class for all optimizers.

class deepmd.pt.optimizer.KFOptimizerWrapper(model: torch.nn.Module, optimizer: torch.optim.optimizer.Optimizer, atoms_selected: int, atoms_per_group: int, is_distributed: bool = False)

update_energy(inputs: dict, Etot_label: torch.Tensor, update_prefactor: float = 1) → None

update_force(inputs: dict, Force_label: torch.Tensor, update_prefactor: float = 1) → None

update_denoise_coord(inputs: dict, clean_coord: torch.Tensor, update_prefactor: float = 1, mask_loss_coord: bool = True, coord_mask: torch.Tensor = None) → None

__sample(atoms_selected: int, atoms_per_group: int, natoms: int) → numpy.ndarray

class deepmd.pt.optimizer.LKFOptimizer(params, kalman_lambda=0.98, kalman_nue=0.9987, block_size=5120)

Bases: torch.optim.optimizer.Optimizer

Base class for all optimizers.

Warning: Parameters need to be specified as collections that have a deterministic ordering that is consistent between runs. Examples of objects that don't satisfy those properties are sets and iterators over values of dictionaries.

Parameters

- **params** (*iterable*) – an iterable of torch.Tensor s or dict s. Specifies what Tensors should be optimized.
- **defaults** – (dict): a dict containing default values of optimization options (used when a parameter group doesn't specify them).

__init_P()

__get_blocksize()

__get_nue()

__split_weights(weight)

__update(H, error, weights)

set_grad_prefactor(grad_prefactor)

step(error)

Performs a single optimization step (parameter update).

Parameters

closure (*Callable*) – A closure that reevaluates the model and returns the loss. Optional for most optimizers.

Note: Unless otherwise specified, this function should not modify the .grad field of the parameters.

get_device_id(index)

deepmd.pt.train

Submodules

deepmd.pt.train.training

Module Contents

Classes

Trainer

Functions

_model_change_out_bias(_model, new_type_map, ...)

Attributes

log

deepmd.pt.train.training.log

class deepmd.pt.train.training.**Trainer**(config: Dict[str, Any], training_data, stat_file_path=None, validation_data=None, init_model=None, restart_model=None, finetune_model=None, force_load=False, shared_links=None, finetune_links=None, init_frz_model=None)

run()

save_model(save_path, lr=0.0, step=0)

get_data(is_train=True, task_key='Default')

print_header(fout, train_results, valid_results)

print_on_training(fout, step_id, cur_lr, train_results, valid_results)

deepmd.pt.train.training.**_model_change_out_bias**(_model, new_type_map, _sample_func, _model_params)

deepmd.pt.train.wrapper

Module Contents

Classes

ModelWrapper	Base class for all neural network modules.

Attributes

log

deepmd.pt.train.wrapper.log

class deepmd.pt.train.wrapper.**ModelWrapper**(model: torch.nn.Module | Dict, loss: torch.nn.Module | Dict = None, model_params=None, shared_links=None)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

share_params(shared_links, resume=False)

Share the parameters of classes following rules defined in shared_links during multitask training. If not start from checkpoint (resume is False), some seperated parameters (e.g. mean and stddev) will be re-calculated across different classes.

forward(coord, atype, spin: torch.Tensor | None = None, box: torch.Tensor | None = None, cur_lr: torch.Tensor | None = None, label: torch.Tensor | None = None, task_key: torch.Tensor | None = None, inference_only=False, do_atomic_virial=False, fparam: torch.Tensor | None = None, aparam: torch.Tensor | None = None)

set_extra_state(state: Dict)

Set extra state contained in the loaded state_dict.

This function is called from `load_state_dict()` to handle any extra state found within the state_dict. Implement this function and a corresponding *get_extra_state()* for your module if you need to store extra state within its state_dict.

Parameters
state (*dict*) – Extra state from the state_dict

get_extra_state() → Dict

Return any extra state to include in the module's state_dict.

Implement this and a corresponding *set_extra_state()* for your module if you need to store extra state. This function is called when building the module's state_dict().

Note that extra state should be picklable to ensure working serialization of the state_dict. We only provide provide backwards compatibility guarantees for serializing Tensors; other objects may break backwards compatibility if their serialized pickled form changes.

Returns
Any extra state to store in the module's state_dict

Return type
object

deepmd.pt.utils

Submodules

deepmd.pt.utils.ase_calc

Module Contents

Classes

DPCalculator	Implementation of ASE deepmd calculator.

class deepmd.pt.utils.ase_calc.DPCalculator(model: str | pathlib.Path, label: str = 'DP', type_dict: Dict[str, int] | None = None, neighbor_list=None, **kwargs)

Bases: ase.calculators.calculator.Calculator

Implementation of ASE deepmd calculator.

Implemented propertie are energy, forces and stress

Parameters

model
: [Union[str, Path]] path to the model

label
: [str, optional] calculator label, by default "DP"

type_dict
: [Dict[str, int], optional] mapping of element types and their numbers, best left None and the calculator will infer this information from model, by default None

neighbor_list
: [ase.neighborlist.NeighborList, optional] The neighbor list object. If None, then build the native neighbor list.

Examples

Compute potential energy

```
>>> from ase import Atoms
>>> from deepmd.tf.calculator import DP
>>> water = Atoms('H2O',
>>>               positions=[(0.7601, 1.9270, 1),
>>>                          (1.9575, 1, 1),
>>>                          (1., 1., 1.)],
>>>               cell=[100, 100, 100],
>>>               calculator=DP(model="frozen_model.pb"))
>>> print(water.get_potential_energy())
>>> print(water.get_forces())
```

Run BFGS structure optimization

```
>>> from ase.optimize import BFGS
>>> dyn = BFGS(water)
>>> dyn.run(fmax=1e-6)
>>> print(water.get_positions())
```

name = 'DP'

implemented_properties: ClassVar[List[str]] = ['energy', 'free_energy', 'forces', 'virial', 'stress']

calculate(atoms: ase.Atoms | None = None, properties: List[str] = ['energy', 'forces', 'virial'], system_changes: List[str] = all_changes)

Run calculation with deepmd model.

Parameters

atoms
: [Optional[Atoms], optional] atoms object to run the calculation on, by default None

properties
: [List[str], optional] unused, only for function signature compatibility, by default ["energy", "forces", "stress"]

system_changes
[List[str], optional] unused, only for function signature compatibility, by default all_changes

deepmd.pt.utils.auto_batch_size

Module Contents

Classes

AutoBatchSize	Auto batch size.

class deepmd.pt.utils.auto_batch_size.**AutoBatchSize**(initial_batch_size: int = 1024, factor: float = 2.0)

Bases: *deepmd.utils.batch_size.AutoBatchSize*

Auto batch size.

Parameters

initial_batch_size
[int, default: 1024] initial batch size (number of total atoms) when DP_INFER_BATCH_SIZE is not set

factor
[float, default: 2.] increased factor

is_gpu_available() → bool

Check if GPU is available.

Returns

bool
True if GPU is available

is_oom_error(e: Exception) → bool

Check if the exception is an OOM error.

Parameters

e
[Exception] Exception

execute_all(callable: Callable, total_size: int, natoms: int, *args, **kwargs) → Tuple[numpy.ndarray | torch.Tensor]

Excuate a method with all given data.

Parameters

callable
[Callable] The method should accept *args and **kwargs as input and return the similiar array.

total_size
[int] Total size

natoms
: [int] The number of atoms

*args
: Variable length argument list.

**kwargs
: If 2D np.ndarray or torch.Tensor, assume the first axis is batch; otherwise do nothing.

deepmd.pt.utils.cache

Module Contents

Functions

lru_cache([maxsize, typed, copy, deepcopy])	

deepmd.pt.utils.cache.**lru_cache**(maxsize=16, typed=False, copy=False, deepcopy=False)

deepmd.pt.utils.dataloader

Module Contents

Classes

DpLoaderSet	A dataset for storing DataLoaders to multiple Systems.
BackgroundConsumer	A class that represents a thread of control.
BufferedIterator	

Functions

setup_seed(seed)	
collate_batch(batch)	
get_weighted_sampler(training_data, prob_style[, sys_prob])	

Attributes

log
_sentinel
QUEUESIZE

deepmd.pt.utils.dataloader.**log**

deepmd.pt.utils.dataloader.**setup_seed**(seed)

class deepmd.pt.utils.dataloader.**DpLoaderSet**(systems, batch_size, type_map, seed=10, shuffle=True)

Bases: torch.utils.data.Dataset

A dataset for storing DataLoaders to multiple Systems.

Parameters

sys_path
Path to the data system

batch_size
Max frame count in a batch.

type_map
Gives the name of different atom types

seed
Random seed for dataloader

shuffle
If the data are shuffled (Only effective in serial mode. Always shuffle in distributed data parallelism)

set_noise(noise_settings)

__len__()

__getitem__(idx)

add_data_requirement(data_requirement: List[deepmd.utils.data.DataRequirementItem])
Add data requirement for each system in multiple systems.

print_summary(name: str, prob: List[float])

deepmd.pt.utils.dataloader.**_sentinel**

deepmd.pt.utils.dataloader.**QUEUESIZE = 32**

class deepmd.pt.utils.dataloader.**BackgroundConsumer**(queue, source, max_len)

Bases: threading.Thread

A class that represents a thread of control.

This class can be safely subclassed in a limited fashion. There are two ways to specify the activity: by passing a callable object to the constructor, or by overriding the run() method in a subclass.

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

class deepmd.pt.utils.dataloader.**BufferedIterator**(iterable)

_create_consumer()

__iter__()

__len__()

__next__()

deepmd.pt.utils.dataloader.**collate_batch**(batch)

deepmd.pt.utils.dataloader.**get_weighted_sampler**(training_data, prob_style, sys_prob=False)

deepmd.pt.utils.dataset

Module Contents

Classes

DeepmdDataSetForLoader	An abstract class representing a Dataset.

class deepmd.pt.utils.dataset.**DeepmdDataSetForLoader**(system: str, type_map: List[str] | None = None)

Bases: torch.utils.data.Dataset

An abstract class representing a Dataset.

All datasets that represent a map from keys to data samples should subclass it. All subclasses should overwrite *__getitem__()*, supporting fetching a data sample for a given key. Subclasses could also optionally overwrite *__len__()*, which is expected to return the size of the dataset by many Sampler implementations and the default options of DataLoader. Subclasses could also optionally implement __getitems__(), for speedup batched samples loading. This method accepts list of indices of samples of batch and returns list of samples.

Note: DataLoader by default constructs an index sampler that yields integral indices. To make it work with a map-style dataset with non-integral indices/keys, a custom sampler must be provided.

__len__()

__getitem__(index)

Get a frame from the selected system.

add_data_requirement(data_requirement: List[deepmd.utils.data.DataRequirementItem])

Add data requirement for this data system.

deepmd.pt.utils.dp_random

Module Contents

Functions

choice(a[, size, replace, p])	Generates a random sample from a given 1-D array.
random([size])	Return random floats in the half-open interval [0.0, 1.0).
seed([val])	Seed the generator.
shuffle(x)	Modify a sequence in-place by shuffling its contents.

deepmd.pt.utils.dp_random.**choice**(a: numpy.ndarray | int, size: int | Tuple[int, Ellipsis] | None = None, replace: bool = True, p: numpy.ndarray | None = None)

Generates a random sample from a given 1-D array.

Parameters

a
: [1-D array_like or `int`] If an ndarray, a random sample is generated from its elements. If an int, the random sample is generated as if it were np.arange(a)

size
: [`int` or `tuple of ints`, `optional`] Output shape. If the given shape is, e.g., (m, n, k), then m * n * k samples are drawn. Default is None, in which case a single value is returned.

replace
: [bool, `optional`] Whether the sample is with or without replacement. Default is True, meaning that a value of a can be selected multiple times.

p
: [1-D array_like, `optional`] The probabilities associated with each entry in a. If not given, the sample assumes a uniform distribution over all entries in a.

Returns

`np.ndarray`
: arrays with results and their shapes

deepmd.pt.utils.dp_random.**random**(size=None)

Return random floats in the half-open interval [0.0, 1.0).

Parameters

size
: Output shape.

Returns

`np.ndarray`
: Arrays with results and their shapes.

deepmd.pt.utils.dp_random.**seed**(val: int | None = None)

Seed the generator.

Parameters

val
[int] Seed.

deepmd.pt.utils.dp_random.**shuffle**(x: numpy.ndarray)

Modify a sequence in-place by shuffling its contents.

Parameters

x
[np.ndarray] The array or list to be shuffled.

deepmd.pt.utils.env

Module Contents

deepmd.pt.utils.env.GLOBAL_NP_FLOAT_PRECISION

deepmd.pt.utils.env.SAMPLER_RECORD

deepmd.pt.utils.env.NUM_WORKERS

deepmd.pt.utils.env.LOCAL_RANK

deepmd.pt.utils.env.LOCAL_RANK

deepmd.pt.utils.env.DEVICE

deepmd.pt.utils.env.JIT = False

deepmd.pt.utils.env.CACHE_PER_SYS = 5

deepmd.pt.utils.env.ENERGY_BIAS_TRAINABLE = True

deepmd.pt.utils.env.PRECISION_DICT

deepmd.pt.utils.env.GLOBAL_PT_FLOAT_PRECISION

deepmd.pt.utils.env.GLOBAL_PT_ENER_FLOAT_PRECISION

deepmd.pt.utils.env.RESERVED_PRECISON_DICT

deepmd.pt.utils.env.DEFAULT_PRECISION = 'float64'

deepmd.pt.utils.env_mat_stat

Module Contents

Classes

EnvMatStat	A base class to store and calculate the statistics of the environment matrix.
EnvMatStatSe	Environmental matrix statistics for the se_a/se_r environemntal matrix.

class deepmd.pt.utils.env_mat_stat.EnvMatStat

Bases: *deepmd.utils.env_mat_stat.EnvMatStat*

A base class to store and calculate the statistics of the environment matrix.

compute_stat(env_mat: Dict[str, torch.Tensor]) → Dict[str, deepmd.utils.env_mat_stat.StatItem]

Compute the statistics of the environment matrix for a single system.

Parameters

env_mat
[torch.Tensor] The environment matrix.

Returns

Dict[str, StatItem]
The statistics of the environment matrix.

class deepmd.pt.utils.env_mat_stat.EnvMatStatSe(descriptor: deepmd.pt.model.descriptor.DescriptorBlock)

Bases: *EnvMatStat*

Environmental matrix statistics for the se_a/se_r environemntal matrix.

Parameters

descriptor
[DescriptorBlock] The descriptor of the model.

iter(data: List[Dict[str, torch.Tensor | List[Tuple[int, int]]]]) → Iterator[Dict[str, deepmd.utils.env_mat_stat.StatItem]]

Get the iterator of the environment matrix.

Parameters

data
[List[Dict[str, Union[torch.Tensor, List[Tuple[int, int]]]]]] The data.

Yields

Dict[str, StatItem]
The statistics of the environment matrix.

get_hash() → str

Get the hash of the environment matrix.

Returns

str
The hash of the environment matrix.

__call__()

deepmd.pt.utils.exclude_mask

Module Contents

Classes

AtomExcludeMask	Computes the type exclusion mask for atoms.
PairExcludeMask	Computes the type exclusion mask for atom pairs.

class deepmd.pt.utils.exclude_mask.**AtomExcludeMask**(ntypes: int, exclude_types: List[int] = [])

Bases: torch.nn.Module

Computes the type exclusion mask for atoms.

reinit(ntypes: int, exclude_types: List[int] = [])

get_exclude_types()

get_type_mask()

forward(atype: torch.Tensor) → torch.Tensor

Compute type exclusion mask for atoms.

Parameters

atype
The extended atom types. shape: nf x natom

Returns

mask
The type exclusion mask for atoms. shape: nf x natom Element [ff,ii] being 0 if type(ii) is excluded, otherwise being 1.

class deepmd.pt.utils.exclude_mask.**PairExcludeMask**(ntypes: int, exclude_types: List[Tuple[int, int]] = [])

Bases: torch.nn.Module

Computes the type exclusion mask for atom pairs.

reinit(ntypes: int, exclude_types: List[Tuple[int, int]] = [])

get_exclude_types()

forward(nlist: torch.Tensor, atype_ext: torch.Tensor) → torch.Tensor

Compute type exclusion mask.

Parameters

nlist
The neighbor list. shape: nf x nloc x nnei

atype_ext
The extended aotm types. shape: nf x nall

Returns

mask
The type exclusion mask of shape: nf x nloc x nnei. Element [ff,ii,jj] being 0 if type(ii), type(nlist[ff,ii,jj]) is excluded, otherwise being 1.

deepmd.pt.utils.finetune

Module Contents

Functions

change_finetune_model_params_single(...[, ...])	
change_finetune_model_params(finetune_model, model_config)	Load model_params according to the pretrained one.

Attributes

log

deepmd.pt.utils.finetune.**log**

deepmd.pt.utils.finetune.**change_finetune_model_params_single**(_single_param_target, _model_param_pretrained, from_multitask=False, model_branch='Default', model_branch_from='')

deepmd.pt.utils.finetune.**change_finetune_model_params**(finetune_model, model_config, model_branch='')

Load model_params according to the pretrained one. This function modifies the fine-tuning input in different modes as follows: 1. Single-task fine-tuning from a single-task pretrained model:

- Updates the model parameters based on the pretrained model.

2.Single-task fine-tuning from a multi-task pretrained model:

- Updates the model parameters based on the selected branch in the pretrained model.
- The chosen branch can be defined from the command-line or finetune_head input parameter.
- If not defined, model parameters in the fitting network will be randomly initialized.

3.Multi-task fine-tuning from a single-task pretrained model:

- Updates model parameters in each branch based on the single branch ('Default') in the pretrained model.
- If finetune_head is not set to 'Default', model parameters in the fitting network of the branch will be randomly initialized.

4.Multi-task fine-tuning from a multi-task pretrained model:

- Updates model parameters in each branch based on the selected branch in the pretrained model.
- The chosen branches can be defined from the finetune_head input parameter of each model.
- If finetune_head is not defined and the model_key is the same as in the pretrained model, it will resume from the model_key branch without fine-tuning.
- If finetune_head is not defined and a new model_key is used, model parameters in the fitting network of the branch will be randomly initialized.

Parameters

finetune_model
: The pretrained model.

model_config
: The fine-tuning input parameters.

model_branch
: The model branch chosen in command-line mode, only for single-task fine-tuning.

Returns

model_config:
: Updated model parameters.

finetune_links:
: Fine-tuning rules in a dict format, with model_branch: model_branch_from pairs. If model_key is not in this dict, it will do just resuming instead of fine-tuning.

deepmd.pt.utils.learning_rate

Module Contents

Classes

LearningRateExp

class deepmd.pt.utils.learning_rate.**LearningRateExp**(start_lr, stop_lr, decay_steps, stop_steps, decay_rate=None, **kwargs)

value(step)

Get the learning rate at the given step.

deepmd.pt.utils.multi_task

Module Contents

Functions

preprocess_shared_params(model_config)	Preprocess the model params for multitask model, and generate the links dict for further sharing.
get_class_name(item_key, item_params)	

deepmd.pt.utils.multi_task.**preprocess_shared_params**(model_config)

Preprocess the model params for multitask model, and generate the links dict for further sharing.

Parameters
model_config – Model params of multitask model.

Returns

model_config: **Preprocessed model params of multitask** model.
Those string names are replaced with real params in shared_dict of model params.

shared_links: **Dict of link infos for further** sharing.
Each item, whose key must be in shared_dict, is a dict with following keys: - "type": The real class type of this item. - "links": List of shared settings, each sub-item is a dict with following keys:

- "model_key": Model key in the model_dict to share this item.
- "shared_type": Type of this shard item.
- "shared_level": Shared level (int) of this item in this model.
  Lower for more params to share, 0 means to share all params in this item.

This list are sorted by "shared_level".

For example, if one has model_config **like** this:
"model": {

"shared_dict": {
"my_type_map": ["foo", "bar"], "my_des1": {

"type": "se_e2_a", "neuron": [10, 20, 40] },

}, "model_dict": {

"model_1": {
"type_map": "my_type_map", "descriptor": "my_des1", "fitting_net": {

"neuron": [100, 100, 100]

}

}, "model_2": {

"type_map": "my_type_map", "descriptor": "my_des1", "fitting_net": {

"neuron": [100, 100, 100]

}

} “model_3”: {

“type_map”: “my_type_map”, “descriptor”: “my_des1:1”, “fitting_net”: {

“neuron”: [100, 100, 100]

}

}

}

}
The above config will init three model branches named model_1 **and** model_2 **and**
model_3,
in which:

- model_2 and model_3 will have the same type_map as that in model_1.
- model_2 will share all the parameters of descriptor with model_1,

while model_3 will share part of parameters of descriptor with model_1 on human-defined share-level 1 (default is 0, meaning share all the parameters). - model_1, model_2 and model_3 have three different `fitting_net`s.

The returned model_config **will automatically fulfill the** `input` model_config **as if** there’s **no** sharing,
and the shared_links **will keep** `all` **the sharing information with** looking:

{
‘my_des1’: {
‘type’: ‘DescrptSeA’, ‘links’: [

{‘model_key’: ‘model_1’, ‘shared_type’: ‘descriptor’, ‘shared_level’: 0},
{‘model_key’: ‘model_2’, ‘shared_type’: ‘descriptor’, ‘shared_level’: 0},
{‘model_key’: ‘model_3’, ‘shared_type’: ‘descriptor’, ‘shared_level’: 1}]

}

}

deepmd.pt.utils.multi_task.**get_class_name**(item_key, item_params)

deepmd.pt.utils.neighbor_stat

Module Contents

Classes

NeighborStatOP	Class for getting neighbor statics data information.
NeighborStat	Neighbor statistics using pure NumPy.

class deepmd.pt.utils.neighbor_stat.**NeighborStatOP**(ntypes: int, rcut: float, mixed_types: bool)

Bases: `torch.nn.Module`

Class for getting neighbor statics data information.

Parameters

ntypes
The num of atom types

rcut
The cut-off radius

mixed_types
[bool, optional] If True, treat neighbors of all types as a single type.

forward(coord: torch.Tensor, atype: torch.Tensor, cell: torch.Tensor | None) → Tuple[torch.Tensor, torch.Tensor]

Calculate the neareest neighbor distance between atoms, maximum nbor size of atoms and the output data range of the environment matrix.

Parameters

coord
The coordinates of atoms.

atype
The atom types.

cell
The cell.

Returns

torch.Tensor
The minimal squared distance between two atoms, in the shape of (nframes,)

torch.Tensor
The maximal number of neighbors

class deepmd.pt.utils.neighbor_stat.NeighborStat(ntypes: int, rcut: float, mixed_type: bool = False)

Bases: *deepmd.utils.neighbor_stat.NeighborStat*

Neighbor statistics using pure NumPy.

Parameters

ntypes
[int] The num of atom types

rcut
[float] The cut-off radius

mixed_type
[bool, optional, default=False] Treat all types as a single type.

iterator(data: deepmd.utils.data_system.DeepmdDataSystem) → Iterator[Tuple[numpy.ndarray, float, str]]

Abstract method for producing data.

Yields

np.ndarray
The maximal number of neighbors

float
The squared minimal distance between two atoms

str
The directory of the data system

_execute(coord: numpy.ndarray, atype: numpy.ndarray, cell: numpy.ndarray | None)

Execute the operation.

Parameters

coord
The coordinates of atoms.

atype
The atom types.

cell
The cell.

deepmd.pt.utils.nlist

Module Contents

Functions

extend_input_and_build_neighbor_list(coord, atype, ...)	
build_neighbor_list(→ torch.Tensor)	Build neightbor list for a single frame. keeps nsel neighbors.
nlist_distinguish_types(nlist, atype, sel)	Given a nlist that does not distinguish atom types, return a nlist that
get_multiple_nlist_key(→ str)	
build_multiple_neighbor_list(→ Dict[str, torch.Tensor])	Input one neighbor list, and produce multiple neighbor lists with
extend_coord_with_ghosts(coord, atype, cell, rcut[, ...])	Extend the coordinates of the atoms by appending peridoc images.

deepmd.pt.utils.nlist.**extend_input_and_build_neighbor_list**(coord, atype, rcut: float, sel: List[int], mixed_types: bool = False, box: torch.Tensor | None = None)

deepmd.pt.utils.nlist.**build_neighbor_list**(coord: torch.Tensor, atype: torch.Tensor, nloc: int, rcut: float, sel: int | List[int], distinguish_types: bool = True) → torch.Tensor

Build neightbor list for a single frame. keeps nsel neighbors.

Parameters

coord
[torch.Tensor] exptended coordinates of shape [batch_size, nall x 3]

atype
[torch.Tensor] extended atomic types of shape [batch_size, nall] if type < 0 the atom is treat as virtual atoms.

nloc
: [int] number of local atoms.

rcut
: [float] cut-off radius

sel
: [int or List[int]] maximal number of neighbors (of each type). if distinguish_types==True, nsel should be list and the length of nsel should be equal to number of types.

distinguish_types
: [bool] distinguish different types.

Returns

neighbor_list
: [torch.Tensor] Neighbor list of shape [batch_size, nloc, nsel], the neighbors are stored in an ascending order. If the number of neighbors is less than nsel, the positions are masked with -1. The neighbor list of an atom looks like |------ nsel ------| xx xx xx xx -1 -1 -1 if distinguish_types==True and we have two types |---- nsel[0] -----| |---- nsel[1] -----| xx xx xx xx -1 -1 -1 xx xx xx -1 -1 -1 -1 For virtual atoms all neighboring positions are filled with -1.

deepmd.pt.utils.nlist.**nlist_distinguish_types**(nlist: torch.Tensor, atype: torch.Tensor, sel: List[int])

Given a nlist that does not distinguish atom types, return a nlist that distinguish atom types.

deepmd.pt.utils.nlist.**get_multiple_nlist_key**(rcut: float, nsel: int) → str

deepmd.pt.utils.nlist.**build_multiple_neighbor_list**(coord: torch.Tensor, nlist: torch.Tensor, rcuts: List[float], nsels: List[int]) → Dict[str, torch.Tensor]

Input one neighbor list, and produce multiple neighbor lists with different cutoff radius and numbers of selection out of it. The required rcuts and nsels should be smaller or equal to the input nlist.

Parameters

coord
: [torch.Tensor] exptended coordinates of shape [batch_size, nall x 3]

nlist
: [torch.Tensor] Neighbor list of shape [batch_size, nloc, nsel], the neighbors should be stored in an ascending order.

rcuts
: [List[float]] list of cut-off radius in ascending order.

nsels
: [List[int]] maximal number of neighbors in ascending order.

Returns

nlist_dict
: [Dict[str, torch.Tensor]] A dict of nlists, key given by get_multiple_nlist_key(rc, nsel) value being the corresponding nlist.

deepmd.pt.utils.nlist.**extend_coord_with_ghosts**(coord: torch.Tensor, atype: torch.Tensor, cell: torch.Tensor | None, rcut: float, cell_cpu: torch.Tensor | None = None)

Extend the coordinates of the atoms by appending peridoc images. The number of images is large enough to ensure all the neighbors within rcut are appended.

Parameters

coord
: [`torch.Tensor`] original coordinates of shape [-1, nloc*3].

atype
: [`torch.Tensor`] atom type of shape [-1, nloc].

cell
: [`torch.Tensor`] simulation cell tensor of shape [-1, 9].

rcut
: [`float`] the cutoff radius

cell_cpu
: [`torch.Tensor`] cell on cpu for performance

Returns

extended_coord: **`torch.Tensor`**
: extended coordinates of shape [-1, nall*3].

extended_atype: **`torch.Tensor`**
: extended atom type of shape [-1, nall].

index_mapping: **`torch.Tensor`**
: maping extended index to the local index

deepmd.pt.utils.plugin

Base of plugin systems.

Module Contents

Classes

Plugin	A class to register and restore plugins.
PluginVariant	A class to remove type from input arguments.
VariantABCMeta	Metaclass for defining Abstract Base Classes (ABCs).
VariantMeta	

class deepmd.pt.utils.plugin.**Plugin**

A class to register and restore plugins.

Examples

```
>>> plugin = Plugin()
>>> @plugin.register("xx")
    def xxx():
        pass
>>> print(plugin.plugins["xx"])
```

Attributes

plugins
[Dict[str, object]] plugins

__add__(other) → Plugin

register(key: str) → Callable[[object], object]
Register a plugin.

Parameters

key
[str] key of the plugin

Returns

Callable[[object], object]
decorator

get_plugin(key) → object
Visit a plugin by key.

Parameters

key
[str] key of the plugin

Returns

object
the plugin

class deepmd.pt.utils.plugin.PluginVariant
A class to remove type from input arguments.

class deepmd.pt.utils.plugin.VariantABCMeta
Bases: *VariantMeta*, abc.ABCMeta

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

class deepmd.pt.utils.plugin.VariantMeta

__call__(*args, **kwargs)
Remove type and keys that starts with underline.

deepmd.pt.utils.preprocess

Module Contents

Classes

Region3D	

Functions

normalize_coord(coord, region, nloc)	Move outer atoms into region by mirror.
compute_serial_cid(cell_offset, ncell)	Tell the sequential cell ID in its 3D space.
compute_pbc_shift(cell_offset, ncell)	Tell shift count to move the atom into region.
build_inside_clist(coord, region, ncell)	Build cell list on atoms inside region.
append_neighbors(coord, region, atype, rcut)	Make ghost atoms who are valid neighbors.
build_neighbor_list(nloc, coord, atype, rcut, sec, mapping)	For each atom inside region, build its neighbor list.
compute_smooth_weight(distance, rmin, rmax)	Compute smooth weight for descriptor elements.
make_env_mat(coord, atype, region, rcut, sec[, pbc, ...])	Based on atom coordinates, return environment matrix.

Attributes

log	

deepmd.pt.utils.preprocess.**log**

class deepmd.pt.utils.preprocess.**Region3D**(boxt)

phys2inter(coord)

Convert physical coordinates to internal ones.

inter2phys(coord)

Convert internal coordinates to physical ones.

get_face_distance()

Return face distinces to each surface of YZ, ZX, XY.

deepmd.pt.utils.preprocess.**normalize_coord**(coord, region: Region3D, nloc: int)

Move outer atoms into region by mirror.

Args: - coord: shape is [nloc*3]

deepmd.pt.utils.preprocess.**compute_serial_cid**(cell_offset, ncell)

Tell the sequential cell ID in its 3D space.

Args: - cell_offset: shape is [3] - ncell: shape is [3]

deepmd.pt.utils.preprocess.**compute_pbc_shift**(cell_offset, ncell)

Tell shift count to move the atom into region.

deepmd.pt.utils.preprocess.**build_inside_clist**(coord, region: Region3D, ncell)

Build cell list on atoms inside region.

Args: - coord: shape is [nloc*3] - ncell: shape is [3]

deepmd.pt.utils.preprocess.**append_neighbors**(coord, region: Region3D, atype, rcut: float)

Make ghost atoms who are valid neighbors.

Args: - coord: shape is [nloc*3] - atype: shape is [nloc]

deepmd.pt.utils.preprocess.**build_neighbor_list**(nloc: int, coord, atype, rcut: float, sec, mapping, type_split=True, min_check=False)

For each atom inside region, build its neighbor list.

Args: - coord: shape is [nall*3] - atype: shape is [nall]

deepmd.pt.utils.preprocess.**compute_smooth_weight**(distance, rmin: float, rmax: float)

Compute smooth weight for descriptor elements.

deepmd.pt.utils.preprocess.**make_env_mat**(coord, atype, region, rcut: float | list, sec, pbc=True, type_split=True, min_check=False)

Based on atom coordinates, return environment matrix.

Returns

nlist: **nlist**, [**nloc**, **nnei**]
merged_coord_shift: **shift on nall atoms**, [**nall**, 3]
merged_mapping: **mapping from nall index to nloc index**, [**nall**]

deepmd.pt.utils.region

Module Contents

Functions

phys2inter(→ torch.Tensor)	Convert physical coordinates to internal(direct) coordinates.
inter2phys(→ torch.Tensor)	Convert internal(direct) coordinates to physical coordinates.
to_face_distance(→ torch.Tensor)	Compute the to-face-distance of the simulation cell.
_to_face_distance(cell)	
b_to_face_distance(cell)	
normalize_coord(→ torch.Tensor)	Apply PBC according to the atomic coordinates.

deepmd.pt.utils.region.**phys2inter**(coord: torch.Tensor, cell: torch.Tensor) → torch.Tensor

Convert physical coordinates to internal(direct) coordinates.

Parameters

coord
[torch.Tensor] physical coordinates of shape [*, na, 3].

cell
[torch.Tensor] simulation cell tensor of shape [*, 3, 3].

Returns

inter_coord: **torch.Tensor**
the internal coordinates

deepmd.pt.utils.region.**inter2phys**(coord: torch.Tensor, cell: torch.Tensor) → torch.Tensor

Convert internal(direct) coordinates to physical coordinates.

Parameters

coord
[torch.Tensor] internal coordinates of shape [*, na, 3].

cell
[torch.Tensor] simulation cell tensor of shape [*, 3, 3].

Returns

phys_coord: **torch.Tensor**
the physical coordinates

deepmd.pt.utils.region.**to_face_distance**(cell: torch.Tensor) → torch.Tensor

Compute the to-face-distance of the simulation cell.

Parameters

cell
[torch.Tensor] simulation cell tensor of shape [*, 3, 3].

Returns

dist: **torch.Tensor**
the to face distances of shape [*, 3]

deepmd.pt.utils.region.**_to_face_distance**(cell)

deepmd.pt.utils.region.**b_to_face_distance**(cell)

deepmd.pt.utils.region.**normalize_coord**(coord: torch.Tensor, cell: torch.Tensor) → torch.Tensor

Apply PBC according to the atomic coordinates.

Parameters

coord
[torch.Tensor] orignal coordinates of shape [*, na, 3].

Returns

wrapped_coord: **torch.Tensor**
wrapped coordinates of shape [*, na, 3].

deepmd.pt.utils.serialization

Module Contents

Functions

serialize_from_file(→ dict)	Serialize the model file to a dictionary.
deserialize_to_file(→ None)	Deserialize the dictionary to a model file.

deepmd.pt.utils.serialization.**serialize_from_file**(model_file: str) → dict

Serialize the model file to a dictionary.

Parameters

model_file
[str] The model file to be serialized.

Returns

dict
The serialized model data.

deepmd.pt.utils.serialization.**deserialize_to_file**(model_file: str, data: dict) → None

Deserialize the dictionary to a model file.

Parameters

model_file
[str] The model file to be saved.

data
[dict] The dictionary to be deserialized.

deepmd.pt.utils.stat

Module Contents

Functions

make_stat_input(datasets, dataloaders, nbatches)	Pack data for statistics.
_restore_from_file(→ Optional[dict])	
_save_to_file(stat_file_path, bias_out, std_out)	
_post_process_stat(out_bias, out_std)	Post process the statistics.
_compute_model_predict(sampled, keys, model_forward)	
_make_preset_out_bias(→ Optional[numpy.array])	Make preset out bias.
_fill_stat_with_global(atomic_stat, global_stat)	This function is used to fill atomic stat with global stat.
compute_output_stats(merged, ntypes[, keys, ...])	Compute the output statistics (e.g. energy bias) for the fitting net from packed data.
compute_output_stats_global(sampled, ntypes, keys[, ...])	This function only handle stat computation from reduced global labels.
compute_output_stats_atomic(sampled, ntypes, keys[, ...])	

Attributes

log

deepmd.pt.utils.stat.**log**

deepmd.pt.utils.stat.**make_stat_input**(datasets, dataloaders, nbatches)

Pack data for statistics.

Args: - dataset: A list of dataset to analyze. - nbatches: Batch count for collecting stats.

Returns

- a list of dicts, each of which contains data from a system

deepmd.pt.utils.stat.**_restore_from_file**(stat_file_path: deepmd.utils.path.DPPath, keys: List[str] = ['energy']) → dict | None

deepmd.pt.utils.stat.**_save_to_file**(stat_file_path: deepmd.utils.path.DPPath, bias_out: dict, std_out: dict)

deepmd.pt.utils.stat.**_post_process_stat**(out_bias, out_std)

Post process the statistics.

For global statistics, we do not have the std for each type of atoms, thus fake the output std by ones for all the types.

deepmd.pt.utils.stat._compute_model_predict(sampled: Callable[[], List[dict]] | List[dict], keys: List[str], model_forward: Callable[Ellipsis, torch.Tensor])

deepmd.pt.utils.stat._make_preset_out_bias(ntypes: int, ibias: List[numpy.array | None]) → numpy.array | None

Make preset out bias.

output:
a np array of shape [ntypes, *(odim0, odim1, ...)] is any item is not None None if all items are None.

deepmd.pt.utils.stat._fill_stat_with_global(atomic_stat: numpy.ndarray | None, global_stat: numpy.ndarray)

This function is used to fill atomic stat with global stat.

Parameters

atomic_stat
[Union[np.ndarray, None]] The atomic stat.

global_stat
[np.ndarray] The global stat.

if the atomic stat is None, use global stat.
if the atomic stat is not None, but has nan values (missing atypes), fill with global stat.

deepmd.pt.utils.stat.compute_output_stats(merged: Callable[[], List[dict]] | List[dict], ntypes: int, keys: str | List[str] = ['energy'], stat_file_path: deepmd.utils.path.DPPath | None = None, rcond: float | None = None, preset_bias: Dict[str, List[torch.Tensor | None]] | None = None, model_forward: Callable[Ellipsis, torch.Tensor] | None = None)

Compute the output statistics (e.g. energy bias) for the fitting net from packed data.

Parameters

merged
[Union[Callable[[], List[dict]], List[dict]]]

- List[dict]: A list of data samples from various data systems.
  Each element, merged[i], is a data dictionary containing keys: torch.Tensor originating from the i-th data system.
- Callable[[], List[dict]]: A lazy function that returns data samples in the above format
  only when needed. Since the sampling process can be slow and memory-intensive, the lazy function helps by only sampling once.

ntypes
[int] The number of atom types.

stat_file_path
[DPPath, optional] The path to the stat file.

rcond
[float, optional] The condition number for the regression of atomic energy.

preset_bias
[Dict[str, List[Optional[torch.Tensor]]], optional] Specifying atomic energy contribution in vacuum. Given by key:value pairs. The value is a list specifying the

bias. the elements can be None or np.array of output shape. For example: [None, [2.]] means type 0 is not set, type 1 is set to [2.] The set_davg_zero key in the descrptor should be set.

model_forward
[Callable[..., torch.Tensor], optional] The wrapped forward function of atomic model. If not None, the model will be utilized to generate the original energy prediction, which will be subtracted from the energy label of the data. The difference will then be used to calculate the delta complement energy bias for each type.

deepmd.pt.utils.stat.**compute_output_stats_global**(sampled: List[dict], ntypes: int, keys: List[str], rcond: float | None = None, preset_bias: Dict[str, List[torch.Tensor | None]] | None = None, model_pred: Dict[str, numpy.ndarray] | None = None)

This function only handle stat computation from reduced global labels.

deepmd.pt.utils.stat.**compute_output_stats_atomic**(sampled: List[dict], ntypes: int, keys: List[str], model_pred: Dict[str, numpy.ndarray] | None = None)

deepmd.pt.utils.update_sel

Module Contents

Classes

UpdateSel	Update the sel field in the descriptor.

class deepmd.pt.utils.update_sel.**UpdateSel**

Bases: *deepmd.utils.update_sel.BaseUpdateSel*

Update the sel field in the descriptor.

property neighbor_stat: Type[*deepmd.pt.utils.neighbor_stat.NeighborStat*]

hook(min_nbor_dist, max_nbor_size)

deepmd.pt.utils.utils

Module Contents

Classes

ActivationFn	Base class for all neural network modules.

Functions

to_numpy_array(...)	
to_torch_tensor(...)	
dict_to_device(sample_dict)	

class deepmd.pt.utils.utils.**ActivationFn**(activation: str | None)

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(x: torch.Tensor) → torch.Tensor

Returns the tensor after applying activation function corresponding to activation.

deepmd.pt.utils.utils.**to_numpy_array**(xx: torch.Tensor) → numpy.ndarray

deepmd.pt.utils.utils.**to_numpy_array**(xx: None) → None

deepmd.pt.utils.utils.**to_torch_tensor**(xx: numpy.ndarray) → torch.Tensor

deepmd.pt.utils.utils.**to_torch_tensor**(xx: None) → None

deepmd.pt.utils.utils.**dict_to_device**(sample_dict)

Package Contents

Classes

AtomExcludeMask	Computes the type exclusion mask for atoms.
PairExcludeMask	Computes the type exclusion mask for atom pairs.

class deepmd.pt.utils.**AtomExcludeMask**(ntypes: int, exclude_types: List[int] = [])

Bases: torch.nn.Module

Computes the type exclusion mask for atoms.

reinit(ntypes: int, exclude_types: List[int] = [])

get_exclude_types()

get_type_mask()

forward(atype: torch.Tensor) → torch.Tensor

Compute type exclusion mask for atoms.

Parameters

atype
The extended atom types. shape: nf x natom

Returns

mask
The type exclusion mask for atoms. shape: nf x natom Element [ff,ii] being 0 if type(ii) is excluded, otherwise being 1.

class deepmd.pt.utils.**PairExcludeMask**(ntypes: int, exclude_types: List[Tuple[int, int]] = [])

Bases: torch.nn.Module

Computes the type exclusion mask for atom pairs.

reinit(ntypes: int, exclude_types: List[Tuple[int, int]] = [])

get_exclude_types()

forward(nlist: torch.Tensor, atype_ext: torch.Tensor) → torch.Tensor

Compute type exclusion mask.

Parameters

nlist
The neighbor list. shape: nf x nloc x nnei

atype_ext
The extended aotm types. shape: nf x nall

Returns

mask
The type exclusion mask of shape: nf x nloc x nnei. Element [ff,ii,jj] being 0 if type(ii), type(nlist[ff,ii,jj]) is excluded, otherwise being 1.

Submodules

deepmd.pt.cxx_op

Module Contents

deepmd.pt.cxx_op.ENABLE_CUSTOMIZED_OP

Package Contents

deepmd.pt.ENABLE_CUSTOMIZED_OP

20.1.7 deepmd.tf

Root of the deepmd package, exposes all public classes and submodules.

Subpackages

deepmd.tf.cluster

Module that reads node resources, auto detects if running local or on SLURM.

Submodules

deepmd.tf.cluster.local

Get local GPU resources.

Module Contents

Functions

get_gpus()			Get available IDs of GPU cards at local.
get_resource(→ tional[List[int]]])	Tuple[str,	List[str],	Op- Get local resources: nodename, nodelist, and gpus.

deepmd.tf.cluster.local.get_gpus()

Get available IDs of GPU cards at local. These IDs are valid when used as the TensorFlow device ID.

Returns

Optional[List[int]]
List of available GPU IDs. Otherwise, None.

deepmd.tf.cluster.local.get_resource() → Tuple[str, List[str], List[int] | None]

Get local resources: nodename, nodelist, and gpus.

Returns

Tuple[str, List[str], Optional[List[int]]]
nodename, nodelist, and gpus

Package Contents

Functions

get_resource(→ Tuple[str, List[str], Optional[List[int]]])	Get local or slurm resources: nodename, nodelist, and gpus.

deepmd.tf.cluster.get_resource() → Tuple[str, List[str], List[int] | None]

Get local or slurm resources: nodename, nodelist, and gpus.

Returns

Tuple[str, List[str], Optional[List[int]]]
nodename, nodelist, and gpus

deepmd.tf.descriptor

Submodules

deepmd.tf.descriptor.descriptor

Module Contents

Classes

Descriptor	The abstract class for descriptors. All specific descriptors should

class deepmd.tf.descriptor.descriptor.Descriptor

Bases: *deepmd.tf.utils.PluginVariant*, make_plugin_registry('descriptor')

The abstract class for descriptors. All specific descriptors should be based on this class.

The descriptor $\mathcal{D}$ describes the environment of an atom, which should be a function of coordinates and types of its neighbour atoms.

Notes

Only methods and attributes defined in this class are generally public, that can be called by other classes.

Examples

```
>>> descript = Descriptor(type="se_e2_a", rcut=6.0, rcut_smth=0.5, sel=[50])
>>> type(descript)
<class 'deepmd.tf.descriptor.se_a.DescrptSeA'>
```

property explicit_ntypes: bool

Explicit ntypes with type embedding.

abstract get_rcut() → float

Returns the cut-off radius.

Returns

float
the cut-off radius

Notes

This method must be implemented, as it's called by other classes.

abstract get_ntypes() → int

Returns the number of atom types.

Returns

int
the number of atom types

Notes

This method must be implemented, as it's called by other classes.

abstract get_dim_out() → int

Returns the output dimension of this descriptor.

Returns

int
the output dimension of this descriptor

Notes

This method must be implemented, as it's called by other classes.

abstract get_dim_rot_mat_1() → int

Returns the first dimension of the rotation matrix. The rotation is of shape dim_1 x 3.

Returns

int
the first dimension of the rotation matrix

abstract get_nlist() → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, List[int], List[int]]

Returns neighbor information.

Returns

nlist
[tf.Tensor] Neighbor list

rij
[tf.Tensor] The relative distance between the neighbor and the center atom.

sel_a
[list[int]] The number of neighbors with full information

sel_r
[list[int]] The number of neighbors with only radial information

abstract compute_input_stats(data_coord: List[numpy.ndarray], data_box: List[numpy.ndarray], data_atype: List[numpy.ndarray], natoms_vec: List[numpy.ndarray], mesh: List[numpy.ndarray], input_dict: Dict[str, List[numpy.ndarray]], **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
[list[np.ndarray]] The coordinates. Can be generated by *deepmd.tf.model.model_stat.make_stat_input()*

data_box
[list[np.ndarray]] The box. Can be generated by *deepmd.tf.model.model_stat.make_stat_input()*

data_atype
[list[np.ndarray]] The atom types. Can be generated by *deepmd.tf.model.model_stat.make_stat_input()*

natoms_vec
[list[np.ndarray]] The vector for the number of atoms of the system and different types of atoms. Can be generated by *deepmd.tf.model.model_stat.make_stat_input()*

mesh
[list[np.ndarray]] The mesh for neighbor searching. Can be generated by *deepmd.tf.model.model_stat.make_stat_input()*

input_dict
[dict[str, list[np.ndarray]]] Dictionary for additional input

**kwargs
Additional keyword arguments which may contain mixed_type and real_natoms_vec.

Notes

This method must be implemented, as it's called by other classes.

abstract build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: Dict[str, Any], reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
[tf.Tensor] The coordinate of atoms

atype_
[tf.Tensor] The type of atoms

natoms
[tf.Tensor] The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of frames

mesh
[tf.Tensor] For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
[dict[str, *Any*]] Dictionary for additional inputs

reuse
[bool, optional] The weights in the networks should be reused when get the variable.

suffix
[str, optional] Name suffix to identify this descriptor

Returns

descriptor: tf.Tensor
The output descriptor

Notes

This method must be implemented, as it's called by other classes.

abstract enable_compression(min_nbor_dist: float, graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, table_extrapolate: float = 5.0, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
[`float`] The nearest distance between atoms

graph
[`tf.Graph`] The graph of the model

graph_def
[`tf.GraphDef`] The graph definition of the model

table_extrapolate
[`float`, default: 5.] The scale of model extrapolation

table_stride_1
[`float`, default: 0.01] The uniform stride of the first table

table_stride_2
[`float`, default: 0.1] The uniform stride of the second table

check_frequency
[`int`, default: -1] The overflow check frequency

suffix
[`str`, `optional`] The suffix of the scope

Notes

This method is called by others when the descriptor supported compression.

abstract enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

Notes

This method is called by others when the descriptor supported compression.

abstract prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
[tf.Tensor] The atomic energy

natoms
[tf.Tensor] The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
[tf.Tensor] The force on atoms

virial
[tf.Tensor] The total virial

atom_virial
[tf.Tensor] The atomic virial

abstract init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the embedding net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str, optional] The suffix of the scope

Notes

This method is called by others when the descriptor supported initialization from the given variables.

abstract get_tensor_names(suffix: str = '') → Tuple[str]

Get names of tensors.

Parameters

suffix
[str] The suffix of the scope

Returns

Tuple[str]
Names of tensors

abstract pass_tensors_from_frz_model(*tensors: deepmd.tf.env.tf.Tensor) → None

Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.

Parameters

*tensors
[tf.Tensor] passed tensors

Notes

The number of parameters in the method must be equal to the numbers of returns in *get_tensor_names()*.

build_type_exclude_mask(exclude_types: Set[Tuple[int, int]], ntypes: int, sel: List[int], ndescrpt: int, atype: deepmd.tf.env.tf.Tensor, shape0: deepmd.tf.env.tf.Tensor) → deepmd.tf.env.tf.Tensor

Build the type exclude mask for the descriptor.

Parameters

exclude_types
: [List[Tuple[int, int]]] The list of excluded types, e.g. [(0, 1), (1, 0)] means the interaction between type 0 and type 1 is excluded.

ntypes
: [int] The number of types.

sel
: [List[int]] The list of the number of selected neighbors for each type.

ndescrpt
: [int] The number of descriptors for each atom.

atype
: [tf.Tensor] The type of atoms, with the size of shape0.

shape0
: [tf.Tensor] The shape of the first dimension of the inputs, which is equal to nsamples * natoms.

Returns

tf.Tensor
: The type exclude mask, with the shape of (shape0, ndescrpt), and the precision of GLOBAL_TF_FLOAT_PRECISION. The mask has the value of 1 if the interaction between two types is not excluded, and 0 otherwise.

Notes

To exclude the interaction between two types, the derivative of energy with respect to distances (or angles) between two atoms should be zero[Rdfe82bee38f2-1]_, i.e.

$$\forall i \in \text{type } 1, j \in \text{type } 2, \frac{\partial E}{\partial r_{ij}} = 0$$

When embedding networks between every two types are built, we can just remove that network. But when type_one_side is enabled, a network may be built for multiple pairs of types. In this case, we need to build a mask to exclude the interaction between two types.

The mask assumes the descriptors are sorted by neighbro type with the fixed number of given sel and each neighbor has the same number of descriptors (for example 4).

References

[1]

abstract classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

classmethod deserialize(data: dict, suffix: str = '') → Descriptor

Deserialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Parameters

data
[dict] The serialized data

suffix
[str, optional] Name suffix to identify this descriptor

Returns

Descriptor
The deserialized descriptor

abstract serialize(suffix: str = '') → dict

Serialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Returns

dict
The serialized data

suffix
[str, optional] Name suffix to identify this descriptor

deepmd.tf.descriptor.hybrid

Module Contents

Classes

DescrptHybrid	Concate a list of descriptors to form a new descriptor.

class deepmd.tf.descriptor.hybrid.DescrptHybrid(list: List[deepmd.tf.descriptor.descriptor.Descriptor | Dict[str, Any]], multi_task: bool = False, ntypes: int | None = None, spin: deepmd.tf.utils.spin.Spin | None = None, **kwargs)

Bases: *deepmd.tf.descriptor.descriptor.Descriptor*

Concate a list of descriptors to form a new descriptor.

Parameters

list
[list][List[Union[Descriptor, Dict[str, *Any*]]]] Build a descriptor from the concatenation of the list of descriptors. The descriptor can be either an object or a dictionary.

property explicit_ntypes: bool
Explicit ntypes with type embedding.

get_rcut() → float
Returns the cut-off radius.

get_ntypes() → int
Returns the number of atom types.

get_dim_out() → int
Returns the output dimension of this descriptor.

get_nlist() → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, List[int], List[int]]
Get the neighbor information of the descriptor, returns the nlist of the descriptor with the largest cut-off radius.

Returns

nlist
Neighbor list

rij
The relative distance between the neighbor and the center atom.

sel_a
The number of neighbors with full information

sel_r
The number of neighbors with only radial information

get_nlist_i(ii: int) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, List[int], List[int]]
Get the neighbor information of the ii-th descriptor.

Parameters

ii
[int] The index of the descriptor

Returns

nlist
Neighbor list

rij
The relative distance between the neighbor and the center atom.

sel_a
The number of neighbors with full information

sel_r
The number of neighbors with only radial information

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, mixed_type: bool = False, real_natoms_vec: list | None = None, **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
Dictionary for additional input

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

real_natoms_vec
If mixed_type is True, it takes in the real natoms_vec for each frame.

**kwargs
Additional keyword arguments.

merge_input_stats(stat_dict)

Merge the statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict

The dict of statisitcs computed from compute_input_stats, including:

sumr
The sum of radial statisitcs.

suma
The sum of relative coord statisitcs.

sumn
The sum of neighbor numbers.

sumr2
: The sum of square of radial statisitcs.

suma2
: The sum of square of relative coord statisitcs.

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
: The coordinate of atoms

atype_
: The type of atoms

natoms
: The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
: [tf.Tensor] The box of the system

mesh
: For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
: Dictionary for additional inputs

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

descriptor
: The output descriptor

prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
: The atomic energy

natoms
: The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
The total virial

atom_virial
The atomic virial

enable_compression(min_nbor_dist: float, graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, table_extrapolate: float = 5.0, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
[float] The nearest distance between atoms

graph
[tf.Graph] The graph of the model

graph_def
[tf.GraphDef] The graph_def of the model

table_extrapolate
[float, default: 5.] The scale of model extrapolation

table_stride_1
[float, default: 0.01] The uniform stride of the first table

table_stride_2
[float, default: 0.1] The uniform stride of the second table

check_frequency
[int, default: -1] The overflow check frequency

suffix
[str, optional] The suffix of the scope

enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the embedding net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str, optional] The suffix of the scope

get_tensor_names(suffix: str = '') → Tuple[str]

Get names of tensors.

Parameters

suffix
[str] The suffix of the scope

Returns

Tuple[str]
Names of tensors

pass_tensors_from_frz_model(*tensors: deepmd.tf.env.tf.Tensor) → None

Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.

Parameters

*tensors
[tf.Tensor] passed tensors

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

serialize(suffix: str = '') → dict

Serialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Returns

dict
The serialized data

suffix
[str, optional] Name suffix to identify this descriptor

classmethod deserialize(data: dict, suffix: str = '') → DescrptHybrid

Deserialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Parameters

data
[dict] The serialized data

suffix
[str, optional] Name suffix to identify this descriptor

Returns

Descriptor
The deserialized descriptor

deepmd.tf.descriptor.loc_frame

Module Contents

Classes

DescrptLocFrame	Defines a local frame at each atom, and the compute the descriptor as local

class deepmd.tf.descriptor.loc_frame.DescrptLocFrame(rcut: float, sel_a: List[int], sel_r: List[int], axis_rule: List[int], **kwargs)

Bases: *deepmd.tf.descriptor.descriptor.Descriptor*

Defines a local frame at each atom, and the compute the descriptor as local coordinates under this frame.

Parameters

rcut
: The cut-off radius

sel_a
: [list[int]] The length of the list should be the same as the number of atom types in the system. sel_a[i] gives the selected number of type-i neighbors. The full relative coordinates of the neighbors are used by the descriptor.

sel_r
: [list[int]] The length of the list should be the same as the number of atom types in the system. sel_r[i] gives the selected number of type-i neighbors. Only relative distance of the neighbors are used by the descriptor. sel_a[i] + sel_r[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius.

axis_rule: list[int]
: The length should be 6 times of the number of types. - axis_rule[i*6+0]: class of the atom defining the first axis of type-i atom. 0 for neighbors with full coordinates and 1 for neighbors only with relative distance. - axis_rule[i*6+1]: type of the atom defining the first axis of type-i atom. - axis_rule[i*6+2]: index of the axis atom defining the first axis. Note that the neighbors with the same class and type are sorted according to their relative distance. - axis_rule[i*6+3]: class of the atom defining the second axis of type-i atom. 0 for neighbors with full coordinates and 1 for neighbors only with relative distance. - axis_rule[i*6+4]: type of the atom defining the second axis of type-i atom. - axis_rule[i*6+5]: index of the axis atom defining the second axis. Note that the neighbors with the same class and type are sorted according to their relative distance.

get_rcut() → float
: Returns the cut-off radius.

get_ntypes() → int
: Returns the number of atom types.

get_dim_out() → int
: Returns the output dimension of this descriptor.

get_nlist() → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, List[int], List[int]]

Returns

nlist
Neighbor list

rij
The relative distance between the neighbor and the center atom.

sel_a
The number of neighbors with full information

sel_r
The number of neighbors with only radial information

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
Dictionary for additional input

**kwargs
Additional keyword arguments.

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

get_rot_mat() → deepmd.tf.env.tf.Tensor

Get rotational matrix.

prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
The atomic energy

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
The total virial

atom_virial
The atomic virial

_compute_dstats_sys_nonsmth(data_coord, data_box, data_atype, natoms_vec, mesh)

_compute_std(sumv2, sumv, sumn)

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the embedding net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str, optional] The suffix of the scope

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

deepmd.tf.descriptor.se

Module Contents

Classes

DescrptSe	A base class for smooth version of descriptors.

class deepmd.tf.descriptor.se.DescrptSe

Bases: *deepmd.tf.descriptor.descriptor.Descriptor*

A base class for smooth version of descriptors.

Notes

All of these descriptors have an environmental matrix and an embedding network (*deepmd.tf.utils.network.embedding_net()*), so they can share some similiar methods without defining them twice.

Attributes

embedding_net_variables
[dict] initial embedding network variables

descrpt_reshape
[tf.Tensor] the reshaped descriptor

descrpt_deriv
[tf.Tensor] the descriptor derivative

rij
[tf.Tensor] distances between two atoms

nlist
[tf.Tensor] the neighbor list

property precision: deepmd.tf.env.tf.DType

Precision of filter network.

_identity_tensors(suffix: str = '') → None

Identify tensors which are expected to be stored and restored.

Parameters

suffix
[str] The suffix of the scope

Notes

These tensors will be indentitied:
self.descrpt_reshape : o_rmat self.descrpt_deriv : o_rmat_deriv self.rij : o_rij self.nlist : o_nlist

Thus, this method should be called during building the descriptor and after these tensors are initialized.

get_tensor_names(suffix: str = '') → Tuple[str]

Get names of tensors.

Parameters

suffix
[str] The suffix of the scope

Returns

Tuple[str]
Names of tensors

pass_tensors_from_frz_model(descrpt_reshape: deepmd.tf.env.tf.Tensor, descrpt_deriv: deepmd.tf.env.tf.Tensor, rij: deepmd.tf.env.tf.Tensor, nlist: deepmd.tf.env.tf.Tensor)

Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.

Parameters

descrpt_reshape
The passed descrpt_reshape tensor

descrpt_deriv
The passed descrpt_deriv tensor

rij
The passed rij tensor

nlist
The passed nlist tensor

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the embedding net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str, optional] The suffix of the scope

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

serialize_network(ntypes: int, ndim: int, in_dim: int, neuron: List[int], activation_function: str, resnet_dt: bool, variables: dict, excluded_types: Set[Tuple[int, int]] = set(), suffix: str = '') → dict

Serialize network.

Parameters

ntypes
[int] The number of types

ndim
[int] The dimension of elements

in_dim
[int] The input dimension

neuron
[List[int]] The neuron list

activation_function
[str] The activation function

resnet_dt
[bool] Whether to use resnet

variables
[dict] The input variables

excluded_types
[Set[Tuple[int, int]], optional] The excluded types

suffix
[str, optional] The suffix of the scope

Returns

dict
The converted network data

classmethod deserialize_network(data: dict, suffix: str = '') → dict

Deserialize network.

Parameters

data
[dict] The input network data

suffix
[str, optional] The suffix of the scope

Returns

variables
[dict] The input variables

deepmd.tf.descriptor.se_a

Module Contents

Classes

DescrptSeA	DeepPot-SE constructed from all information (both angular and radial) of

class deepmd.tf.descriptor.se_a.DescrptSeA(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, exclude_types: List[List[int]] = [], set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, multi_task: bool = False, spin: deepmd.tf.utils.spin.Spin | None = None, tebd_input_mode: str = 'concat', env_protection: float = 0.0, **kwargs)

Bases: *deepmd.tf.descriptor.se.DescrptSe*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor $\mathcal{D}^i \in \mathcal{R}^{M_1 \times M_2}$ is given by [1]

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}^i_<$$

where $\mathcal{R}^i \in \mathbb{R}^{N \times 4}$ is the coordinate matrix, and each row of $\mathcal{R}^i$ can be constructed as follows

$$(\mathcal{R}^i)_j = [\ \begin{matrix} s(r_{ji}) \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{matrix}\ ]$$

where $\mathrm{R}_{ji} = \mathrm{R}_j - \mathrm{R}_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \|\mathrm{R}_{ji}\|$ is its norm. The switching function $s(r)$ is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r}\{(\frac{r-r_s}{r_c-r_s})^3(-6(\frac{r-r_s}{r_c-r_s})^2 + 15\frac{r-r_s}{r_c-r_s} - 10) + 1\}, & r_s \le r < r_c \\ 0, & r \ge r_c \end{cases}$$

Each row of the embedding matrix $\mathcal{G}^i \in \mathbb{R}^{N \times M_1}$ consists of outputs of a embedding network $\mathcal{N}$ of $s(r_{ji})$:

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}^i_< \in \mathbb{R}^{N \times M_2}$ takes first M_2 columns of $\mathcal{G}^i$. The equation of embedding network $\mathcal{N}$ can be found at *deepmd.tf.utils.network.embedding_net()*.

Parameters

rcut
: The cut-off radius r_c

rcut_smth
: From where the environment matrix should be smoothed r_s

sel
: [list[int]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_typesˆ2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

multi_task
: If the model has multi fitting nets to train.

env_protection: float
: Protection parameter to prevent division by zero errors during environment matrix calculations.

References

[1]

property explicit_ntypes: bool

Explicit ntypes with type embedding.

get_rcut() → float

Returns the cut-off radius.

get_ntypes() → int

Returns the number of atom types.

get_dim_out() → int

Returns the output dimension of this descriptor.

get_dim_rot_mat_1() → int

Returns the first dimension of the rotation matrix. The rotation is of shape dim_1 x 3.

get_nlist() → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, List[int], List[int]]

Returns neighbor information.

Returns

nlist
Neighbor list

rij
The relative distance between the neighbor and the center atom.

sel_a
The number of neighbors with full information

sel_r
The number of neighbors with only radial information

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
Dictionary for additional input

**kwargs
Additional keyword arguments.

merge_input_stats(stat_dict)

Merge the statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict

The dict of statisitcs computed from compute_input_stats, including:

sumr
The sum of radial statisitcs.

suma
The sum of relative coord statisitcs.

sumn
The sum of neighbor numbers.

sumr2
The sum of square of radial statisitcs.

suma2
The sum of square of relative coord statisitcs.

enable_compression(min_nbor_dist: float, graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, table_extrapolate: float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
The nearest distance between atoms

graph
[tf.Graph] The graph of the model

graph_def
[tf.GraphDef] The graph_def of the model

table_extrapolate
The scale of model extrapolation

table_stride_1
The uniform stride of the first table

table_stride_2
The uniform stride of the second table

check_frequency
The overflow check frequency

suffix
[str, optional] The suffix of the scope

enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

get_rot_mat() → deepmd.tf.env.tf.Tensor

Get rotational matrix.

prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
The atomic energy

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
The total virial

atom_virial
The atomic virial

_pass_filter(inputs, atype, natoms, input_dict, reuse=None, suffix='', trainable=True)

_compute_dstats_sys_smth(data_coord, data_box, data_atype, natoms_vec, mesh)

_compute_std(sumv2, sumv, sumn)

_concat_type_embedding(xyz_scatter, nframes, natoms, type_embedding)

Concatenate type_embedding of neighbors and xyz_scatter. If not self.type_one_side, concatenate type_embedding of center atoms as well.

Parameters

xyz_scatter:
shape is [nframes*natoms[0]*self.nnei, 1]

nframes:
shape is []

natoms:
shape is [1+1+self.ntypes]

type_embedding:
shape is [self.ntypes, Y] where Y=jdata['type_embedding']['neuron'][-1]

Returns

embedding:
environment of each atom represented by embedding.

_filter_lower(type_i, type_input, start_index, incrs_index, inputs, nframes, natoms, type_embedding=None, is_exclude=False, activation_fn=None, bavg=0.0, stddev=1.0, trainable=True, suffix='')

Input env matrix, returns R.G.

_filter(inputs, type_input, natoms, type_embedding=None, activation_fn=tf.nn.tanh, stddev=1.0, bavg=0.0, name='linear', reuse=None, trainable=True)

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the embedding net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str, optional] The suffix of the scope

classmethod deserialize(data: dict, suffix: str = '')

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

Model
The deserialized model

serialize(suffix: str = '') → dict

Serialize the model.

Parameters

suffix
[str, optional] The suffix of the scope

Returns

dict
The serialized data

deepmd.tf.descriptor.se_a_ebd

Module Contents

Classes

DescrptSeAEbd	DeepPot-SE descriptor with type embedding approach.

class deepmd.tf.descriptor.se_a_ebd.DescrptSeAEbd(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, type_nchanl: int = 2, type_nlayer: int = 1, numb_aparam: int = 0, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', exclude_types: List[List[int]] = [], **kwargs)

Bases: *deepmd.tf.descriptor.se_a.DescrptSeA*

DeepPot-SE descriptor with type embedding approach.

Parameters

rcut
The cut-off radius

rcut_smth
From where the environment matrix should be smoothed

sel
: [list[int]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net

axis_neuron
: Number of the axis neuron (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

type_nchanl
: Number of channels for type representation

type_nlayer
: Number of hidden layers for the type embedding net (skip connected).

numb_aparam
: Number of atomic parameters. If >0 it will be embedded with atom types.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are {0}

precision
: The precision of the embedding net parameters. Supported options are {1}

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
: The coordinate of atoms

atype_
: The type of atoms

natoms
: The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

_type_embed(atype, ndim=1, reuse=None, suffix='', trainable=True)

_embedding_net(inputs, natoms, filter_neuron, activation_fn=tf.nn.tanh, stddev=1.0, bavg=0.0, name='linear', reuse=None, seed=None, trainable=True)

inputs: nf x na x (nei x 4) outputs: nf x na x nei x output_size.

_type_embedding_net_two_sides(mat_g, atype, natoms, name='', reuse=None, seed=None, trainable=True)

_type_embedding_net_one_side(mat_g, atype, natoms, name='', reuse=None, seed=None, trainable=True)

_type_embedding_net_one_side_aparam(mat_g, atype, natoms, aparam, name='', reuse=None, seed=None, trainable=True)

_pass_filter(inputs, atype, natoms, input_dict, reuse=None, suffix='', trainable=True)

_ebd_filter(inputs, atype, natoms, input_dict, activation_fn=tf.nn.tanh, stddev=1.0, bavg=0.0, name='linear', reuse=None, seed=None, trainable=True)

deepmd.tf.descriptor.se_a_ebd_v2

Module Contents

Classes

DescrptSeAEbdV2	A compressible se_a_ebd model.

Attributes

log

deepmd.tf.descriptor.se_a_ebd_v2.log

class deepmd.tf.descriptor.se_a_ebd_v2.DescrptSeAEbdV2(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, exclude_types: List[List[int]] = [], set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, multi_task: bool = False, spin: deepmd.tf.utils.spin.Spin | None = None, **kwargs)

Bases: *deepmd.tf.descriptor.se_a.DescrptSeA*

A compressible se_a_ebd model.

This model is a warpper for DescriptorSeA, which set tebd_input_mode='strip'.

deepmd.tf.descriptor.se_a_ef

Module Contents

Classes

DescrptSeAEf	Smooth edition descriptor with Ef.
DescrptSeAEfLower	Helper class for implementing DescrptSeAEf.

class deepmd.tf.descriptor.se_a_ef.DescrptSeAEf(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, exclude_types: List[List[int]] = [], set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', uniform_seed=False, **kwargs)

Bases: *deepmd.tf.descriptor.se.DescrptSe*

Smooth edition descriptor with Ef.

Parameters

rcut
The cut-off radius

rcut_smth
: From where the environment matrix should be smoothed

sel
: [list[int]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net

axis_neuron
: Number of the axis neuron (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

get_rcut() → float
: Returns the cut-off radius.

get_ntypes() → int
: Returns the number of atom types.

get_dim_out() → int
: Returns the output dimension of this descriptor.

get_dim_rot_mat_1() → int
: Returns the first dimension of the rotation matrix. The rotation is of shape dim_1 x 3.

get_rot_mat() → deepmd.tf.env.tf.Tensor
: Get rotational matrix.

get_nlist() → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, List[int], List[int]]

Returns neighbor information.

Returns

nlist
Neighbor list

rij
The relative distance between the neighbor and the center atom.

sel_a
The number of neighbors with full information

sel_r
The number of neighbors with only radial information

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
Dictionary for additional input

**kwargs
Additional keyword arguments.

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number

of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs. Should have 'efield'.

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
The atomic energy

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
The total virial

atom_virial
The atomic virial

class deepmd.tf.descriptor.se_a_ef.DescrptSeAEfLower(op, rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, exclude_types: List[List[int]] = [], set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False)

Bases: *deepmd.tf.descriptor.se_a.DescrptSeA*

Helper class for implementing DescrptSeAEf.

compute_input_stats(data_coord, data_box, data_atype, natoms_vec, mesh, input_dict, **kwargs)

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
Dictionary for additional input

**kwargs
Additional keyword arguments.

_normalize_3d(a)

build(coord_, atype_, natoms, box_, mesh, input_dict, suffix='', reuse=None)

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

_compute_dstats_sys_smth(data_coord, data_box, data_atype, natoms_vec, mesh, data_efield)

deepmd.tf.descriptor.se_a_mask

Module Contents

Classes

DescrptSeAMask	DeepPot-SE constructed from all information (both angular and radial) of

class deepmd.tf.descriptor.se_a_mask.DescrptSeAMask(sel: List[int], neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, type_one_side: bool = False, exclude_types: List[List[int]] = [], seed: int | None = None, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, tebd_input_mode: str = 'concat', **kwargs)

Bases: *deepmd.tf.descriptor.se_a.DescrptSeA*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor $\mathcal{D}^i \in \mathcal{R}^{M_1 \times M_2}$ is given by [1]

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}^i_<$$

where $\mathcal{R}^i \in \mathbb{R}^{N \times 4}$ is the coordinate matrix, and each row of $\mathcal{R}^i$ can be constructed as follows

$$(\mathcal{R}^i)_j = [\ \begin{matrix} s(r_{ji}) \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{matrix}\ ]$$

where $\mathrm{R}_{ji} = \mathrm{R}_j - \mathrm{R}_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \|\mathrm{R}_{ji}\|$ is its norm. The switching function $s(r)$ is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r}\{(\frac{r-r_s}{r_c-r_s})^3(-6(\frac{r-r_s}{r_c-r_s})^2 + 15\frac{r-r_s}{r_c-r_s} - 10) + 1\}, & r_s \le r < r_c \\ 0, & r \ge r_c \end{cases}$$

Each row of the embedding matrix $\mathcal{G}^i \in \mathbb{R}^{N \times M_1}$ consists of outputs of a embedding network $\mathcal{N}$ of $s(r_{ji})$:

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}^i_< \in \mathbb{R}^{N \times M_2}$ takes first M_2 columns of $\mathcal{G}^i$. The equation of embedding network $\mathcal{N}$ can be found at *deepmd.tf.utils.network.embedding_net()*. Specially for descriptor se_a_mask is a concise implementation of se_a. The difference is that se_a_mask only considered a non-pbc system. And accept a mask matrix to indicate the atom i in frame j is a real atom or not. (1 means real atom, 0 means ghost atom) Thus se_a_mask can accept a variable number of atoms in a frame.

Parameters

sel
: [list[int]] sel[i] specifies the maxmum number of type i atoms in the neighbor list.

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

activation_function
: The activation function in the embedding net. Supported options are {0}

precision
: The precision of the embedding net parameters. Supported options are {1}

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

References

[1]

get_rcut() → float

Returns the cutoff radius.

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
: The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
: The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
: The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
: The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
: The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
: Dictionary for additional input

**kwargs
: Additional keyword arguments.

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: Dict[str, Any], reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
: The coordinate of atoms

atype_
: The type of atoms

natoms
: The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
: [tf.Tensor] The box of the system

mesh
: For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
: Dictionary for additional inputs

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

descriptor
: The output descriptor

prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
The atomic energy

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
None for se_a_mask op

atom_virial
None for se_a_mask op

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

deepmd.tf.descriptor.se_atten

Module Contents

Classes

DescrptSeAtten	Smooth version descriptor with attention.
DescrptDPA1Compat	Consistent version of the model for testing with other backend references.

Attributes

log

deepmd.tf.descriptor.se_atten.log

class deepmd.tf.descriptor.se_atten.DescrptSeAtten(rcut: float, rcut_smth: float, sel: List[int] | int, ntypes: int, neuron: List[int] = [25, 50, 100], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, set_davg_zero: bool = True, exclude_types: List[List[int]] = [], activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, attn: int = 128, attn_layer: int = 2, attn_dotr: bool = True, attn_mask: bool = False, multi_task: bool = False, smooth_type_embedding: bool = False, tebd_input_mode: str = 'concat', scaling_factor=1.0, normalize=True, temperature=None, trainable_ln: bool = True, ln_eps: float | None = 0.001, concat_output_tebd: bool = True, env_protection: float = 0.0, stripped_type_embedding: bool | None = None, **kwargs)

Bases: *deepmd.tf.descriptor.se_a.DescrptSeA*

Smooth version descriptor with attention.

Parameters

rcut: float
The cut-off radius r_c

rcut_smth: float
From where the environment matrix should be smoothed r_s

sel: list[int], int
list[int]: sel[i] specifies the maxmum number of type i atoms in the cut-off radius int: the total maxmum number of atoms in the cut-off radius

neuron: list[int]
Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron: int
Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt: bool
Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable: bool
If the weights of embedding net are trainable.

seed: int, Optional
Random seed for initializing the network parameters.

type_one_side: bool
If 'False', type embeddings of both neighbor and central atoms are considered. If 'True', only type embeddings of neighbor atoms are considered. Default is 'False'.

exclude_types
[List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero: bool
Set the shift of embedding net input to zero.

activation_function: str
The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision: str
The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

uniform_seed: bool
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

attn: int
The length of hidden vector during scale-dot attention computation.

attn_layer: int
The number of layers in attention mechanism.

attn_dotr: bool
Whether to dot the relative coordinates on the attention weights as a gated scheme.

attn_mask: bool
Whether to mask the diagonal in the attention weights.

ln_eps: float, Optional
The epsilon value for layer normalization.

multi_task: bool
If the model has multi fitting nets to train.

tebd_input_mode: str
The input mode of the type embedding. Supported modes are [“concat”, “strip”]. - “concat”: Concatenate the type embedding with the smoothed radial information as the union input for the embedding network. - “strip”: Use a separated embedding network for the type embedding and combine the output with the radial embedding network output. Default value will be strip in se_atten_v2 descriptor.

smooth_type_embedding: bool
Whether to use smooth process in attention weights calculation. And when using stripped type embedding, whether to dot smooth factor on the network output of type embedding to keep the network smooth, instead of setting set_davg_zero to be True. Default value will be True in se_atten_v2 descriptor.

stripped_type_embedding: bool, Optional
(Deprecated, kept only for compatibility.) Whether to strip the type embedding into a separate embedding network. Setting this parameter to True is equivalent to setting tebd_input_mode to ‘strip’. Setting it to False is equivalent to setting tebd_input_mode to ‘concat’. The default value is None, which means the tebd_input_mode setting will be used instead.

Raises

ValueError
if ntypes is 0.

property explicit_ntypes: bool

Explicit ntypes with type embedding.

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, mixed_type: bool = False, real_natoms_vec: list | None = None, **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. If mixed_type is True, this para is blank. See real_natoms_vec.

mesh
The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
Dictionary for additional input

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

real_natoms_vec
If mixed_type is True, it takes in the real natoms_vec for each frame.

**kwargs
Additional keyword arguments.

enable_compression(min_nbor_dist: float, graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, table_extrapolate: float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
The nearest distance between atoms

graph
[tf.Graph] The graph of the model

graph_def
[tf.GraphDef] The graph_def of the model

table_extrapolate
The scale of model extrapolation

table_stride_1
The uniform stride of the first table

table_stride_2
The uniform stride of the second table

check_frequency
The overflow check frequency

suffix
[str, optional] The suffix of the scope

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

_pass_filter(inputs, atype, natoms, input_dict, reuse=None, suffix='', trainable=True)

_compute_dstats_sys_smth(data_coord, data_box, data_atype, natoms_vec, mesh, mixed_type=False, real_natoms_vec=None)

_lookup_type_embedding(xyz_scatter, natype, type_embedding)

Concatenate type_embedding of neighbors and xyz_scatter. If not self.type_one_side, concatenate type_embedding of center atoms as well.

Parameters

xyz_scatter:
shape is [nframes*natoms[0]*self.nnei, 1]

natype:
neighbor atom type

type_embedding:
shape is [self.ntypes, Y] where Y=jdata['type_embedding']['neuron'][-1]

Returns

embedding:
environment of each atom represented by embedding.

_scaled_dot_attn(Q, K, V, temperature, input_r, dotr=False, do_mask=False, layer=0, save_weights=True)

_attention_layers(input_xyz, layer_num, shape_i, outputs_size, input_r, dotr=False, do_mask=False, trainable=True, suffix='')

_filter_lower(type_i, type_input, start_index, incrs_index, inputs, type_embedding=None, atype=None, is_exclude=False, activation_fn=None, bavg=0.0, stddev=1.0, trainable=True, suffix='', name='filter_', reuse=None)

Input env matrix, returns R.G.

_filter(inputs, type_input, natoms, type_embedding=None, atype=None, activation_fn=tf.nn.tanh, stddev=1.0, bavg=0.0, suffix='', name='linear', reuse=None, trainable=True)

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the embedding net variables with the given dict.

Parameters

graph
[`tf.Graph`] The input frozen model graph

graph_def
[`tf.GraphDef`] The input frozen model graph_def

suffix
[`str`, `optional`] The suffix of the scope

build_type_exclude_mask_mixed(exclude_types: Set[Tuple[int, int]], ntypes: int, sel: List[int], ndescrpt: int, atype: deepmd.tf.env.tf.Tensor, shape0: deepmd.tf.env.tf.Tensor, nei_type_vec: deepmd.tf.env.tf.Tensor) → deepmd.tf.env.tf.Tensor

Build the type exclude mask for the attention descriptor.

Parameters

exclude_types
[`List[Tuple[int, int]]`] The list of excluded types, e.g. [(0, 1), (1, 0)] means the interaction between type 0 and type 1 is excluded.

ntypes
[`int`] The number of types.

sel
[`List[int]`] The list of the number of selected neighbors for each type.

ndescrpt
[`int`] The number of descriptors for each atom.

atype
[tf.Tensor] The type of atoms, with the size of shape0.

shape0
[tf.Tensor] The shape of the first dimension of the inputs, which is equal to nsamples * natoms.

nei_type_vec
[tf.Tensor] The type of neighbors, with the size of (shape0, nnei).

Returns

tf.Tensor
The type exclude mask, with the shape of (shape0, ndescrpt), and the precision of GLOBAL_TF_FLOAT_PRECISION. The mask has the value of 1 if the interaction between two types is not excluded, and 0 otherwise.

See also:

deepmd.tf.descriptor.descriptor.Descriptor.build_type_exclude_mask

Notes

This method has the similiar way to build the type exclude mask as *deepmd.tf.descriptor.descriptor.Descriptor.build_type_exclude_mask()*. The mathmatical expression has been explained in that method. The difference is that the attention descriptor has provided the type of the neighbors (idx_j) that is not in order, so we use it from an extra input.

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

serialize_attention_layers(nlayer: int, nnei: int, embed_dim: int, hidden_dim: int, dotr: bool, do_mask: bool, trainable_ln: bool, ln_eps: float, variables: dict, bias: bool = True, suffix: str = '') → dict

serialize_network_strip(ntypes: int, ndim: int, in_dim: int, neuron: List[int], activation_function: str, resnet_dt: bool, variables: dict, suffix: str = '', type_one_side: bool = False) → dict

Serialize network.

Parameters

ntypes
[int] The number of types

ndim
[int] The dimension of elements

in_dim
[int] The input dimension

neuron
[List[int]] The neuron list

activation_function
[str] The activation function

resnet_dt
[bool] Whether to use resnet

variables
[dict] The input variables

suffix
[str, optional] The suffix of the scope

type_one_side
[bool, optional] If 'False', type embeddings of both neighbor and central atoms are considered. If 'True', only type embeddings of neighbor atoms are considered. Default is 'False'.

Returns

dict
The converted network data

classmethod deserialize_attention_layers(data: dict, suffix: str = '') → dict

Deserialize attention layers.

Parameters

data
[dict] The input attention layer data

suffix
[str, optional] The suffix of the scope

Returns

variables
[dict] The input variables

classmethod deserialize_network_strip(data: dict, suffix: str = '', type_one_side: bool = False) → dict

Deserialize network.

Parameters

data
[dict] The input network data

suffix
[str, optional] The suffix of the scope

type_one_side
[bool, optional] If 'False', type embeddings of both neighbor and central atoms are considered. If 'True', only type embeddings of neighbor atoms are considered. Default is 'False'.

Returns

variables
[dict] The input variables

classmethod deserialize(data: dict, suffix: str = '')

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

Model
The deserialized model

serialize(suffix: str = '') → dict

Serialize the model.

Parameters

suffix
[str, optional] The suffix of the scope

Returns

dict
The serialized data

class deepmd.tf.descriptor.se_atten.DescrptDPA1Compat(rcut: float, rcut_smth: float, sel: List[int] | int, ntypes: int, neuron: List[int] = [25, 50, 100], axis_neuron: int = 8, tebd_dim: int = 8, tebd_input_mode: str = 'concat', resnet_dt: bool = False, trainable: bool = True, type_one_side: bool = True, attn: int = 128, attn_layer: int = 2, attn_dotr: bool = True, attn_mask: bool = False, exclude_types: List[List[int]] = [], env_protection: float = 0.0, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', scaling_factor=1.0, normalize: bool = True, temperature: float | None = None, trainable_ln: bool = True, ln_eps: float | None = 0.001, smooth_type_embedding: bool = True, concat_output_tebd: bool = True, spin: Any | None = None, seed: int | None = None, uniform_seed: bool = False)

Bases: *DescrptSeAtten*

Consistent version of the model for testing with other backend references.

This model includes the type_embedding as attributes and other additional parameters.

Parameters

rcut: float
The cut-off radius r_c

rcut_smth: float
From where the environment matrix should be smoothed r_s

sel: list[int], int
: list[int]: sel[i] specifies the maxmum number of type i atoms in the cut-off radius int: the total maxmum number of atoms in the cut-off radius

ntypes: int
: Number of element types

neuron: list[int]
: Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron: int
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

tebd_dim: int
: Dimension of the type embedding

tebd_input_mode: str
: The input mode of the type embedding. Supported modes are [“concat”, “strip”]. - “concat”: Concatenate the type embedding with the smoothed radial information as the union input for the embedding network. - “strip”: Use a separated embedding network for the type embedding and combine the output with the radial embedding network output.

resnet_dt: bool
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable: bool
: If the weights of this descriptors are trainable.

trainable_ln: bool
: Whether to use trainable shift and scale weights in layer normalization.

ln_eps: float, Optional
: The epsilon value for layer normalization.

type_one_side: bool
: If ‘False’, type embeddings of both neighbor and central atoms are considered. If ‘True’, only type embeddings of neighbor atoms are considered. Default is ‘False’.

attn: int
: Hidden dimension of the attention vectors

attn_layer: int
: Number of attention layers

attn_dotr: bool
: If dot the angular gate to the attention weights

attn_mask: bool
: (Only support False to keep consistent with other backend references.) If mask the diagonal of attention weights

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

env_protection: float
: Protection parameter to prevent division by zero errors during environment matrix calculations.

set_davg_zero: bool
Set the shift of embedding net input to zero.

activation_function: str
The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision: str
The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

scaling_factor: float
(Only to keep consistent with other backend references.) (Not used in this version.) The scaling factor of normalization in calculations of attention weights. If temperature is None, the scaling of attention weights is (N_dim * scaling_factor)**0.5

normalize: bool
(Only support True to keep consistent with other backend references.) (Not used in this version.) Whether to normalize the hidden vectors in attention weights calculation.

temperature: float
(Only support 1.0 to keep consistent with other backend references.) (Not used in this version.) If not None, the scaling of attention weights is temperature itself.

smooth_type_embedding: bool
(Only support False to keep consistent with other backend references.) Whether to use smooth process in attention weights calculation.

concat_output_tebd: bool
Whether to concat type embedding at the output of the descriptor.

spin
(Only support None to keep consistent with old implementation.) The old implementation of deepspin.

`build`(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = ″) → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[`tf.Tensor`] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the embedding net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str, optional] The suffix of the scope

update_attention_layers_serialize(data: dict)

Update the serialized data to be consistent with other backend references.

classmethod deserialize(data: dict, suffix: str = '')

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

Model
The deserialized model

serialize(suffix: str = '') → dict

Serialize the model.

Parameters

suffix
[str, optional] The suffix of the scope

Returns

dict
The serialized data

deepmd.tf.descriptor.se_atten_v2

Module Contents

Classes

DescrptSeAttenV2	Smooth version 2.0 descriptor with attention.

Attributes

log

deepmd.tf.descriptor.se_atten_v2.log

class deepmd.tf.descriptor.se_atten_v2.**DescrptSeAttenV2**(rcut: float, rcut_smth: float, sel: int, ntypes: int, neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, set_davg_zero: bool = False, exclude_types: List[List[int]] = [], activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, attn: int = 128, attn_layer: int = 2, attn_dotr: bool = True, attn_mask: bool = False, multi_task: bool = False, **kwargs)

Bases: *deepmd.tf.descriptor.se_atten.DescrptSeAtten*

Smooth version 2.0 descriptor with attention.

Parameters

rcut
: The cut-off radius r_c

rcut_smth
: From where the environment matrix should be smoothed r_s

sel
: [int] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

attn
: The length of hidden vector during scale-dot attention computation.

attn_layer
: The number of layers in attention mechanism.

attn_dotr
: Whether to dot the relative coordinates on the attention weights as a gated scheme.

attn_mask
: Whether to mask the diagonal in the attention weights.

multi_task
: If the model has multi fitting nets to train.

`deepmd.tf.descriptor.se_r`

Module Contents

Classes

DescrptSeR	DeepPot-SE constructed from radial information of atomic configurations.

class deepmd.tf.descriptor.se_r.DescrptSeR(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, exclude_types: List[List[int]] = [], set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, multi_task: bool = False, spin: deepmd.tf.utils.spin.Spin | None = None, env_protection: float = 0.0, **kwargs)

Bases: *deepmd.tf.descriptor.se.DescrptSe*

DeepPot-SE constructed from radial information of atomic configurations.

The embedding takes the distance between atoms as input.

Parameters

rcut
: The cut-off radius

rcut_smth
: From where the environment matrix should be smoothed

sel
: [list[int]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

activation_function
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

get_rcut()

Returns the cut-off radius.

get_ntypes()

Returns the number of atom types.

get_dim_out()

Returns the output dimension of this descriptor.

get_nlist()

Returns neighbor information.

Returns

nlist
Neighbor list

rij
The relative distance between the neighbor and the center atom.

sel_a
The number of neighbors with full information

sel_r
The number of neighbors with only radial information

compute_input_stats(data_coord, data_box, data_atype, natoms_vec, mesh, input_dict, **kwargs)

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
Dictionary for additional input

**kwargs
Additional keyword arguments.

merge_input_stats(stat_dict)

Merge the statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict

The dict of statisitcs computed from compute_input_stats, including:

sumr
The sum of radial statisitcs.

sumn
The sum of neighbor numbers.

sumr2
The sum of square of radial statisitcs.

enable_compression(min_nbor_dist: float, graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, table_extrapolate: float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
The nearest distance between atoms

graph
[`tf.Graph`] The graph of the model

graph_def
[`tf.GraphDef`] The graph_def of the model

table_extrapolate
The scale of model extrapolation

table_stride_1
The uniform stride of the first table

table_stride_2
The uniform stride of the second table

check_frequency
The overflow check frequency

suffix
[`str`, `optional`] The suffix of the scope

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[`tf.Tensor`] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
The atomic energy

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
The total virial

atom_virial
The atomic virial

_pass_filter(inputs, atype, natoms, reuse=None, suffix='', trainable=True)

_compute_dstats_sys_se_r(data_coord, data_box, data_atype, natoms_vec, mesh)

_compute_std(sumv2, sumv, sumn)

_filter_r(inputs, type_input, natoms, activation_fn=tf.nn.tanh, stddev=1.0, bavg=0.0, name='linear', reuse=None, trainable=True)

classmethod deserialize(data: dict, suffix: str = '')

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

Model
The deserialized model

serialize(suffix: str = '') → dict

Serialize the model.

Parameters

suffix
[str, optional] The suffix of the scope

Returns

dict
The serialized data

deepmd.tf.descriptor.se_t

Module Contents

Classes

DescrptSeT	DeepPot-SE constructed from all information (both angular and radial) of atomic

class deepmd.tf.descriptor.se_t.DescrptSeT(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, multi_task: bool = False, **kwargs)

Bases: *deepmd.tf.descriptor.se.DescrptSe*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations.

The embedding takes angles between two neighboring atoms as input.

Parameters

rcut
The cut-off radius

rcut_smth
From where the environment matrix should be smoothed

sel
[list[int]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
[list[int]] Number of neurons in each hidden layers of the embedding net

resnet_dt
Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
If the weights of embedding net are trainable.

seed
Random seed for initializing the network parameters.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

get_rcut() → float

Returns the cut-off radius.

get_ntypes() → int

Returns the number of atom types.

get_dim_out() → int

Returns the output dimension of this descriptor.

get_nlist() → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, List[int], List[int]]

Returns neighbor information.

Returns

nlist
: Neighbor list

rij
: The relative distance between the neighbor and the center atom.

sel_a
: The number of neighbors with full information

sel_r
: The number of neighbors with only radial information

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
: The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
: The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
: The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
: The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
: The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
: Dictionary for additional input

**kwargs
: Additional keyword arguments.

merge_input_stats(stat_dict)

Merge the statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict
: The dict of statisitcs computed from compute_input_stats, including:

sumr
: The sum of radial statisitcs.

suma
: The sum of relative coord statisitcs.

sumn
: The sum of neighbor numbers.

sumr2
: The sum of square of radial statisitcs.

suma2
: The sum of square of relative coord statisitcs.

enable_compression(min_nbor_dist: float, graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, table_extrapolate: float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
: The nearest distance between atoms

graph
: [tf.Graph] The graph of the model

graph_def
: [tf.GraphDef] The graph_def of the model

table_extrapolate
: The scale of model extrapolation

table_stride_1
: The uniform stride of the first table

table_stride_2
: The uniform stride of the second table

check_frequency
: The overflow check frequency

suffix
[str, optional] The suffix of the scope

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
The atomic energy

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
The total virial

atom_virial
The atomic virial

_pass_filter(inputs, atype, natoms, input_dict, reuse=None, suffix='', trainable=True)

_compute_dstats_sys_smth(data_coord, data_box, data_atype, natoms_vec, mesh)

_compute_std(sumv2, sumv, sumn)

_filter(inputs, type_input, natoms, activation_fn=tf.nn.tanh, stddev=1.0, bavg=0.0, name='linear', reuse=None, trainable=True)

Package Contents

Classes

Descriptor	The abstract class for descriptors. All specific descriptors should
DescrptHybrid	Concate a list of descriptors to form a new descriptor.
DescrptLocFrame	Defines a local frame at each atom, and the compute the descriptor as local
DescrptSeA	DeepPot-SE constructed from all information (both angular and radial) of
DescrptSeAEbd	DeepPot-SE descriptor with type embedding approach.
DescrptSeAEbdV2	A compressible se_a_ebd model.
DescrptSeAEf	Smooth edition descriptor with Ef.
DescrptSeAEfLower	Helper class for implementing DescrptSeAEf.
DescrptSeAMask	DeepPot-SE constructed from all information (both angular and radial) of
DescrptSeAtten	Smooth version descriptor with attention.
DescrptSeAttenV2	Smooth version 2.0 descriptor with attention.
DescrptSeR	DeepPot-SE constructed from radial information of atomic configurations.
DescrptSeT	DeepPot-SE constructed from all information (both angular and radial) of atomic

class deepmd.tf.descriptor.Descriptor

Bases: *deepmd.tf.utils.PluginVariant*, make_plugin_registry('descriptor')

The abstract class for descriptors. All specific descriptors should be based on this class.

The descriptor $\mathcal{D}$ describes the environment of an atom, which should be a function of coordinates and types of its neighbour atoms.

Notes

Only methods and attributes defined in this class are generally public, that can be called by other classes.

Examples

```
>>> descript = Descriptor(type="se_e2_a", rcut=6.0, rcut_smth=0.5, sel=[50])
>>> type(descript)
<class 'deepmd.tf.descriptor.se_a.DescrptSeA'>
```

property explicit_ntypes: bool

Explicit ntypes with type embedding.

abstract get_rcut() → float

Returns the cut-off radius.

Returns

float
the cut-off radius

Notes

This method must be implemented, as it's called by other classes.

abstract get_ntypes() → int

Returns the number of atom types.

Returns

int
the number of atom types

Notes

This method must be implemented, as it's called by other classes.

abstract get_dim_out() → int

Returns the output dimension of this descriptor.

Returns

int
the output dimension of this descriptor

Notes

This method must be implemented, as it's called by other classes.

abstract get_dim_rot_mat_1() → int

Returns the first dimension of the rotation matrix. The rotation is of shape dim_1 x 3.

Returns

int
the first dimension of the rotation matrix

abstract get_nlist() → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, List[int], List[int]]

Returns neighbor information.

Returns

nlist
[tf.Tensor] Neighbor list

rij
[tf.Tensor] The relative distance between the neighbor and the center atom.

sel_a
[list[int]] The number of neighbors with full information

sel_r
[list[int]] The number of neighbors with only radial information

abstract compute_input_stats(data_coord: List[numpy.ndarray], data_box: List[numpy.ndarray], data_atype: List[numpy.ndarray], natoms_vec: List[numpy.ndarray], mesh: List[numpy.ndarray], input_dict: Dict[str, List[numpy.ndarray]], **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
[list[np.ndarray]] The coordinates. Can be generated by *deepmd.tf.model.model_stat.make_stat_input()*

data_box
[list[np.ndarray]] The box. Can be generated by *deepmd.tf.model.model_stat.make_stat_input()*

data_atype
[list[np.ndarray]] The atom types. Can be generated by *deepmd.tf.model.model_stat.make_stat_input()*

natoms_vec
[list[np.ndarray]] The vector for the number of atoms of the system and different types of atoms. Can be generated by *deepmd.tf.model.model_stat.make_stat_input()*

mesh
[list[np.ndarray]] The mesh for neighbor searching. Can be generated by *deepmd.tf.model.model_stat.make_stat_input()*

input_dict
[dict[str, list[np.ndarray]]] Dictionary for additional input

**kwargs
Additional keyword arguments which may contain mixed_type and real_natoms_vec.

Notes

This method must be implemented, as it's called by other classes.

abstract build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: Dict[str, Any], reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
[tf.Tensor] The coordinate of atoms

atype_
[tf.Tensor] The type of atoms

natoms
[tf.Tensor] The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of frames

mesh
[tf.Tensor] For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
[dict[str, *Any*]] Dictionary for additional inputs

reuse
[bool, optional] The weights in the networks should be reused when get the variable.

suffix
[str, optional] Name suffix to identify this descriptor

Returns

descriptor: tf.Tensor
The output descriptor

Notes

This method must be implemented, as it's called by other classes.

abstract enable_compression(min_nbor_dist: float, graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, table_extrapolate: float = 5.0, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
[`float`] The nearest distance between atoms

graph
[`tf.Graph`] The graph of the model

graph_def
[`tf.GraphDef`] The graph definition of the model

table_extrapolate
[`float`, default: 5.] The scale of model extrapolation

table_stride_1
[`float`, default: 0.01] The uniform stride of the first table

table_stride_2
[`float`, default: 0.1] The uniform stride of the second table

check_frequency
[`int`, default: -1] The overflow check frequency

suffix
[`str`, `optional`] The suffix of the scope

Notes

This method is called by others when the descriptor supported compression.

abstract enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

Notes

This method is called by others when the descriptor supported compression.

abstract prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
[tf.Tensor] The atomic energy

natoms
[tf.Tensor] The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
[tf.Tensor] The force on atoms

virial
[tf.Tensor] The total virial

atom_virial
[tf.Tensor] The atomic virial

abstract init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the embedding net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str, optional] The suffix of the scope

Notes

This method is called by others when the descriptor supported initialization from the given variables.

abstract get_tensor_names(suffix: str = '') → Tuple[str]

Get names of tensors.

Parameters

suffix
[str] The suffix of the scope

Returns

Tuple[str]
Names of tensors

abstract pass_tensors_from_frz_model(*tensors: deepmd.tf.env.tf.Tensor) → None

Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.

Parameters

*tensors
[tf.Tensor] passed tensors

Notes

The number of parameters in the method must be equal to the numbers of returns in *get_tensor_names()*.

build_type_exclude_mask(exclude_types: Set[Tuple[int, int]], ntypes: int, sel: List[int], ndescrpt: int, atype: deepmd.tf.env.tf.Tensor, shape0: deepmd.tf.env.tf.Tensor) → deepmd.tf.env.tf.Tensor

Build the type exclude mask for the descriptor.

Parameters

exclude_types
: [List[Tuple[int, int]]] The list of excluded types, e.g. [(0, 1), (1, 0)] means the interaction between type 0 and type 1 is excluded.

ntypes
: [int] The number of types.

sel
: [List[int]] The list of the number of selected neighbors for each type.

ndescrpt
: [int] The number of descriptors for each atom.

atype
: [tf.Tensor] The type of atoms, with the size of shape0.

shape0
: [tf.Tensor] The shape of the first dimension of the inputs, which is equal to nsamples * natoms.

Returns

tf.Tensor
: The type exclude mask, with the shape of (shape0, ndescrpt), and the precision of GLOBAL_TF_FLOAT_PRECISION. The mask has the value of 1 if the interaction between two types is not excluded, and 0 otherwise.

Notes

To exclude the interaction between two types, the derivative of energy with respect to distances (or angles) between two atoms should be zero[Rdfe82bee38f2-1]_, i.e.

$$\forall i \in \text{type } 1, j \in \text{type } 2, \frac{\partial E}{\partial r_{ij}} = 0$$

When embedding networks between every two types are built, we can just remove that network. But when type_one_side is enabled, a network may be built for multiple pairs of types. In this case, we need to build a mask to exclude the interaction between two types.

The mask assumes the descriptors are sorted by neighbro type with the fixed number of given sel and each neighbor has the same number of descriptors (for example 4).

References

[1]

abstract classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
: [dict] The global data, containing the training section

local_jdata
: [dict] The local data refer to the current class

classmethod deserialize(data: dict, suffix: str = '') → Descriptor

Deserialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Parameters

data
: [dict] The serialized data

suffix
: [str, optional] Name suffix to identify this descriptor

Returns

Descriptor
: The deserialized descriptor

abstract serialize(suffix: str = '') → dict

Serialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Returns

dict
: The serialized data

suffix
: [str, optional] Name suffix to identify this descriptor

class deepmd.tf.descriptor.DescrptHybrid(list: List[deepmd.tf.descriptor.descriptor.Descriptor | Dict[str, Any]], multi_task: bool = False, ntypes: int | None = None, spin: deepmd.tf.utils.spin.Spin | None = None, **kwargs)

Bases: *deepmd.tf.descriptor.descriptor.Descriptor*

Concate a list of descriptors to form a new descriptor.

Parameters

list
: [list][List[Union[*Descriptor*, Dict[str, *Any*]]]] Build a descriptor from the concatenation of the list of descriptors. The descriptor can be either an object or a dictionary.

property explicit_ntypes: bool

Explicit ntypes with type embedding.

`get_rcut()` → float

Returns the cut-off radius.

`get_ntypes()` → int

Returns the number of atom types.

`get_dim_out()` → int

Returns the output dimension of this descriptor.

`get_nlist()` → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, List[int], List[int]]

Get the neighbor information of the descriptor, returns the nlist of the descriptor with the largest cut-off radius.

Returns

nlist
Neighbor list

rij
The relative distance between the neighbor and the center atom.

sel_a
The number of neighbors with full information

sel_r
The number of neighbors with only radial information

`get_nlist_i`(ii: int) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, List[int], List[int]]

Get the neighbor information of the ii-th descriptor.

Parameters

ii
[int] The index of the descriptor

Returns

nlist
Neighbor list

rij
The relative distance between the neighbor and the center atom.

sel_a
The number of neighbors with full information

sel_r
The number of neighbors with only radial information

`compute_input_stats`(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, mixed_type: bool = False, real_natoms_vec: list | None = None, **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
Dictionary for additional input

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

real_natoms_vec
If mixed_type is True, it takes in the real natoms_vec for each frame.

**kwargs
Additional keyword arguments.

merge_input_stats(stat_dict)

Merge the statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict

The dict of statisitcs computed from compute_input_stats, including:

sumr
The sum of radial statisitcs.

suma
The sum of relative coord statisitcs.

sumn
The sum of neighbor numbers.

sumr2
The sum of square of radial statisitcs.

suma2
The sum of square of relative coord statisitcs.

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
The atomic energy

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
The total virial

atom_virial
The atomic virial

enable_compression(min_nbor_dist: float, graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, table_extrapolate: float = 5.0, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
[float] The nearest distance between atoms

graph
[tf.Graph] The graph of the model

graph_def
[tf.GraphDef] The graph_def of the model

table_extrapolate
[float, default: 5.] The scale of model extrapolation

table_stride_1
[float, default: 0.01] The uniform stride of the first table

table_stride_2
[float, default: 0.1] The uniform stride of the second table

check_frequency
[int, default: -1] The overflow check frequency

suffix
[str, optional] The suffix of the scope

enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the embedding net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str, optional] The suffix of the scope

get_tensor_names(suffix: str = '') → Tuple[str]

Get names of tensors.

Parameters

suffix
[str] The suffix of the scope

Returns

Tuple[str]
Names of tensors

pass_tensors_from_frz_model(*tensors: deepmd.tf.env.tf.Tensor) → None

Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.

Parameters

*tensors
[tf.Tensor] passed tensors

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

serialize(suffix: str = '') → dict

Serialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Returns

dict
The serialized data

suffix
[str, optional] Name suffix to identify this descriptor

classmethod deserialize(data: dict, suffix: str = '') → DescrptHybrid

Deserialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Parameters

data
[dict] The serialized data

suffix
[str, optional] Name suffix to identify this descriptor

Returns

Descriptor
The deserialized descriptor

class deepmd.tf.descriptor.DescrptLocFrame(rcut: float, sel_a: List[int], sel_r: List[int], axis_rule: List[int], **kwargs)

Bases: *deepmd.tf.descriptor.descriptor.Descriptor*

Defines a local frame at each atom, and the compute the descriptor as local coordinates under this frame.

Parameters

rcut
The cut-off radius

sel_a
[list[int]] The length of the list should be the same as the number of atom types in the system. sel_a[i] gives the selected number of type-i neighbors. The full relative coordinates of the neighbors are used by the descriptor.

sel_r
[list[int]] The length of the list should be the same as the number of atom types in the system. sel_r[i] gives the selected number of type-i neighbors. Only relative distance of the neighbors are used by the descriptor. sel_a[i] + sel_r[i] is recommended

to be larger than the maximally possible number of type-i neighbors in the cut-off radius.

axis_rule: list[int]
The length should be 6 times of the number of types. - axis_rule[i*6+0]: class of the atom defining the first axis of type-i atom. 0 for neighbors with full coordinates and 1 for neighbors only with relative distance. - axis_rule[i*6+1]: type of the atom defining the first axis of type-i atom. - axis_rule[i*6+2]: index of the axis atom defining the first axis. Note that the neighbors with the same class and type are sorted according to their relative distance. - axis_rule[i*6+3]: class of the atom defining the second axis of type-i atom. 0 for neighbors with full coordinates and 1 for neighbors only with relative distance. - axis_rule[i*6+4]: type of the atom defining the second axis of type-i atom. - axis_rule[i*6+5]: index of the axis atom defining the second axis. Note that the neighbors with the same class and type are sorted according to their relative distance.

get_rcut() → float

Returns the cut-off radius.

get_ntypes() → int

Returns the number of atom types.

get_dim_out() → int

Returns the output dimension of this descriptor.

get_nlist() → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, List[int], List[int]]

Returns

nlist
Neighbor list

rij
The relative distance between the neighbor and the center atom.

sel_a
The number of neighbors with full information

sel_r
The number of neighbors with only radial information

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
: The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
: Dictionary for additional input

**kwargs
: Additional keyword arguments.

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
: The coordinate of atoms

atype_
: The type of atoms

natoms
: The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
: [tf.Tensor] The box of the system

mesh
: For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
: Dictionary for additional inputs

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

descriptor
: The output descriptor

get_rot_mat() → deepmd.tf.env.tf.Tensor

Get rotational matrix.

prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
: The atomic energy

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
The total virial

atom_virial
The atomic virial

_compute_dstats_sys_nonsmth(data_coord, data_box, data_atype, natoms_vec, mesh)

_compute_std(sumv2, sumv, sumn)

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the embedding net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str, optional] The suffix of the scope

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

class deepmd.tf.descriptor.DescrptSeA(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, exclude_types: List[List[int]] = [], set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, multi_task: bool = False, spin: deepmd.tf.utils.spin.Spin | None = None, tebd_input_mode: str = 'concat', env_protection: float = 0.0, **kwargs)

Bases: *deepmd.tf.descriptor.se.DescrptSe*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor $\mathcal{D}^i \in \mathcal{R}^{M_1 \times M_2}$ is given by [1]

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}^i_<$$

where $\mathcal{R}^i \in \mathbb{R}^{N \times 4}$ is the coordinate matrix, and each row of $\mathcal{R}^i$ can be constructed as follows

$$(\mathcal{R}^i)_j = [\begin{array}{c} s(r_{ji}) \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{array}]$$

where $\mathrm{R}_{ji} = \mathrm{R}_j - \mathrm{R}_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \|\mathrm{R}_{ji}\|$ is its norm. The switching function $s(r)$ is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r}\{(\frac{r-r_s}{r_c-r_s})^3(-6(\frac{r-r_s}{r_c-r_s})^2 + 15\frac{r-r_s}{r_c-r_s} - 10) + 1\}, & r_s \le r < r_c \\ 0, & r \ge r_c \end{cases}$$

Each row of the embedding matrix $\mathcal{G}^i \in \mathbb{R}^{N \times M_1}$ consists of outputs of a embedding network $\mathcal{N}$ of $s(r_{ji})$:

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}^i_< \in \mathbb{R}^{N \times M_2}$ takes first M_2 columns of $\mathcal{G}^i$. The equation of embedding network $\mathcal{N}$ can be found at *deepmd.tf.utils.network.embedding_net()*.

Parameters

rcut
: The cut-off radius r_c

rcut_smth
: From where the environment matrix should be smoothed r_s

sel
: [list[int]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

multi_task
: If the model has multi fitting nets to train.

env_protection: float
: Protection parameter to prevent division by zero errors during environment matrix calculations.

References

[1]

property explicit_ntypes: bool
: Explicit ntypes with type embedding.

get_rcut() → float
: Returns the cut-off radius.

get_ntypes() → int
: Returns the number of atom types.

get_dim_out() → int
: Returns the output dimension of this descriptor.

get_dim_rot_mat_1() → int
: Returns the first dimension of the rotation matrix. The rotation is of shape dim_1 x 3.

get_nlist() → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, List[int], List[int]]
: Returns neighbor information.

  Returns

  nlist
  : Neighbor list

  rij
  : The relative distance between the neighbor and the center atom.

  sel_a
  : The number of neighbors with full information

  sel_r
  : The number of neighbors with only radial information

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
Dictionary for additional input

**kwargs
Additional keyword arguments.

merge_input_stats(stat_dict)

Merge the statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict
The dict of statisitcs computed from compute_input_stats, including:

sumr
The sum of radial statisitcs.

suma
The sum of relative coord statisitcs.

sumn
The sum of neighbor numbers.

sumr2
The sum of square of radial statisitcs.

suma2
The sum of square of relative coord statisitcs.

enable_compression(min_nbor_dist: float, graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, table_extrapolate: float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
The nearest distance between atoms

graph
[tf.Graph] The graph of the model

graph_def
[tf.GraphDef] The graph_def of the model

table_extrapolate
The scale of model extrapolation

table_stride_1
The uniform stride of the first table

table_stride_2
The uniform stride of the second table

check_frequency
The overflow check frequency

suffix
[str, optional] The suffix of the scope

enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

get_rot_mat() → deepmd.tf.env.tf.Tensor

Get rotational matrix.

prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
The atomic energy

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
The total virial

atom_virial
The atomic virial

_pass_filter(inputs, atype, natoms, input_dict, reuse=None, suffix='', trainable=True)

_compute_dstats_sys_smth(data_coord, data_box, data_atype, natoms_vec, mesh)

_compute_std(sumv2, sumv, sumn)

_concat_type_embedding(xyz_scatter, nframes, natoms, type_embedding)

Concatenate type_embedding of neighbors and xyz_scatter. If not self.type_one_side, concatenate type_embedding of center atoms as well.

Parameters

xyz_scatter:
shape is [nframes*natoms[0]*self.nnei, 1]

nframes:
shape is []

natoms:
shape is [1+1+self.ntypes]

type_embedding:
shape is [self.ntypes, Y] where Y=jdata['type_embedding']['neuron'][-1]

Returns

embedding:
environment of each atom represented by embedding.

_filter_lower(type_i, type_input, start_index, incrs_index, inputs, nframes, natoms, type_embedding=None, is_exclude=False, activation_fn=None, bavg=0.0, stddev=1.0, trainable=True, suffix='')

Input env matrix, returns R.G.

_filter(inputs, type_input, natoms, type_embedding=None, activation_fn=tf.nn.tanh, stddev=1.0, bavg=0.0, name='linear', reuse=None, trainable=True)

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the embedding net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str, optional] The suffix of the scope

classmethod deserialize(data: dict, suffix: str = '')

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

Model
The deserialized model

serialize(suffix: str = '') → dict

Serialize the model.

Parameters

suffix
[str, optional] The suffix of the scope

Returns

dict
The serialized data

class deepmd.tf.descriptor.DescrptSeAEbd(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, type_nchanl: int = 2, type_nlayer: int = 1, numb_aparam: int = 0, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', exclude_types: List[List[int]] = [], **kwargs)

Bases: *deepmd.tf.descriptor.se_a.DescrptSeA*

DeepPot-SE descriptor with type embedding approach.

Parameters

rcut
: The cut-off radius

rcut_smth
: From where the environment matrix should be smoothed

sel
: [list[int]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net

axis_neuron
: Number of the axis neuron (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

type_nchanl
: Number of channels for type representation

type_nlayer
: Number of hidden layers for the type embedding net (skip connected).

numb_aparam
: Number of atomic parameters. If >0 it will be embedded with atom types.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are {0}

precision
: The precision of the embedding net parameters. Supported options are {1}

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

_type_embed(atype, ndim=1, reuse=None, suffix='', trainable=True)

_embedding_net(inputs, natoms, filter_neuron, activation_fn=tf.nn.tanh, stddev=1.0, bavg=0.0, name='linear', reuse=None, seed=None, trainable=True)

inputs: nf x na x (nei x 4) outputs: nf x na x nei x output_size.

_type_embedding_net_two_sides(mat_g, atype, natoms, name='', reuse=None, seed=None, trainable=True)

_type_embedding_net_one_side(mat_g, atype, natoms, name='', reuse=None, seed=None, trainable=True)

_type_embedding_net_one_side_aparam(mat_g, atype, natoms, aparam, name='', reuse=None, seed=None, trainable=True)

_pass_filter(inputs, atype, natoms, input_dict, reuse=None, suffix='', trainable=True)

_ebd_filter(inputs, atype, natoms, input_dict, activation_fn=tf.nn.tanh, stddev=1.0, bavg=0.0, name='linear', reuse=None, seed=None, trainable=True)

class deepmd.tf.descriptor.**DescrptSeAEbdV2**(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, exclude_types: List[List[int]] = [], set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, multi_task: bool = False, spin: deepmd.tf.utils.spin.Spin | None = None, **kwargs)

Bases: *deepmd.tf.descriptor.se_a.DescrptSeA*

A compressible se_a_ebd model.

This model is a warpper for DescriptorSeA, which set tebd_input_mode='strip'.

class deepmd.tf.descriptor.DescrptSeAEf(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, exclude_types: List[List[int]] = [], set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', uniform_seed=False, **kwargs)

Bases: *deepmd.tf.descriptor.se.DescrptSe*

Smooth edition descriptor with Ef.

Parameters

rcut
: The cut-off radius

rcut_smth
: From where the environment matrix should be smoothed

sel
: [`list`[`int`]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
: [`list`[`int`]] Number of neurons in each hidden layers of the embedding net

axis_neuron
: Number of the axis neuron (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [`List`[`List`[`int`]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

uniform_seed
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

get_rcut() → float

Returns the cut-off radius.

get_ntypes() → int

Returns the number of atom types.

get_dim_out() → int

Returns the output dimension of this descriptor.

get_dim_rot_mat_1() → int

Returns the first dimension of the rotation matrix. The rotation is of shape dim_1 x 3.

get_rot_mat() → deepmd.tf.env.tf.Tensor

Get rotational matrix.

get_nlist() → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, List[int], List[int]]

Returns neighbor information.

Returns

nlist
Neighbor list

rij
The relative distance between the neighbor and the center atom.

sel_a
The number of neighbors with full information

sel_r
The number of neighbors with only radial information

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
Dictionary for additional input

**kwargs
Additional keyword arguments.

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs. Should have 'efield'.

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
The atomic energy

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
The total virial

atom_virial
The atomic virial

class deepmd.tf.descriptor.DescrptSeAEfLower(op, rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, exclude_types: List[List[int]] = [], set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False)

Bases: *deepmd.tf.descriptor.se_a.DescrptSeA*

Helper class for implementing DescrptSeAEf.

compute_input_stats(data_coord, data_box, data_atype, natoms_vec, mesh, input_dict, **kwargs)

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
Dictionary for additional input

**kwargs
Additional keyword arguments.

_normalize_3d(a)

build(coord_, atype_, natoms, box_, mesh, input_dict, suffix='', reuse=None)

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number

of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

_compute_dstats_sys_smth(data_coord, data_box, data_atype, natoms_vec, mesh, data_efield)

class deepmd.tf.descriptor.DescrptSeAMask(sel: List[int], neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, type_one_side: bool = False, exclude_types: List[List[int]] = [], seed: int | None = None, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, tebd_input_mode: str = 'concat', **kwargs)

Bases: *deepmd.tf.descriptor.se_a.DescrptSeA*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor $\mathcal{D}^i \in \mathcal{R}^{M_1 \times M_2}$ is given by [1]

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}^i_<$$

where $\mathcal{R}^i \in \mathbb{R}^{N \times 4}$ is the coordinate matrix, and each row of $\mathcal{R}^i$ can be constructed as follows

$$(\mathcal{R}^i)_j = [\begin{array}{c} s(r_{ji}) \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{array}]$$

where $\mathrm{R}_{ji} = \mathrm{R}_j - \mathrm{R}_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \|\mathrm{R}_{ji}\|$ is its norm. The switching function $s(r)$ is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r}\{(\frac{r-r_s}{r_c-r_s})^3(-6(\frac{r-r_s}{r_c-r_s})^2 + 15\frac{r-r_s}{r_c-r_s} - 10) + 1\}, & r_s \le r < r_c \\ 0, & r \ge r_c \end{cases}$$

Each row of the embedding matrix $\mathcal{G}^i \in \mathbb{R}^{N \times M_1}$ consists of outputs of a embedding network $\mathcal{N}$ of $s(r_{ji})$:

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}^i_< \in \mathbb{R}^{N \times M_2}$ takes first M_2 columns of $\mathcal{G}^i$. The equation of embedding network $\mathcal{N}$ can be found at *deepmd.tf.utils.network.embedding_net()*. Specially for descriptor se_a_mask is a concise implementation of se_a. The difference is that se_a_mask only considered a non-pbc system. And accept a mask matrix to indicate the atom i in frame j is a real atom or not. (1 means real atom, 0 means ghost atom) Thus se_a_mask can accept a variable number of atoms in a frame.

Parameters

sel
: [list[int]] sel[i] specifies the maxmum number of type i atoms in the neighbor list.

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

activation_function
: The activation function in the embedding net. Supported options are {0}

precision
: The precision of the embedding net parameters. Supported options are {1}

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

References

[1]

get_rcut() → float

Returns the cutoff radius.

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
: The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
: The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
: The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
: The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
: The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
: Dictionary for additional input

**kwargs
: Additional keyword arguments.

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: Dict[str, Any], reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
: The coordinate of atoms

atype_
: The type of atoms

natoms
: The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
: [tf.Tensor] The box of the system

mesh
: For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
: Dictionary for additional inputs

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

descriptor
: The output descriptor

prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
The atomic energy

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
None for se_a_mask op

atom_virial
None for se_a_mask op

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

class deepmd.tf.descriptor.DescrptSeAtten(rcut: float, rcut_smth: float, sel: List[int] | int, ntypes: int, neuron: List[int] = [25, 50, 100], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, set_davg_zero: bool = True, exclude_types: List[List[int]] = [], activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, attn: int = 128, attn_layer: int = 2, attn_dotr: bool = True, attn_mask: bool = False, multi_task: bool = False, smooth_type_embedding: bool = False, tebd_input_mode: str = 'concat', scaling_factor=1.0, normalize=True, temperature=None, trainable_ln: bool = True, ln_eps: float | None = 0.001, concat_output_tebd: bool = True, env_protection: float = 0.0, stripped_type_embedding: bool | None = None, **kwargs)

Bases: *deepmd.tf.descriptor.se_a.DescrptSeA*

Smooth version descriptor with attention.

Parameters

rcut: float
The cut-off radius r_c

rcut_smth: float
: From where the environment matrix should be smoothed r_s

sel: list[int], int
: list[int]: sel[i] specifies the maxmum number of type i atoms in the cut-off radius int: the total maxmum number of atoms in the cut-off radius

neuron: list[int]
: Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron: int
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt: bool
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable: bool
: If the weights of embedding net are trainable.

seed: int, Optional
: Random seed for initializing the network parameters.

type_one_side: bool
: If 'False', type embeddings of both neighbor and central atoms are considered. If 'True', only type embeddings of neighbor atoms are considered. Default is 'False'.

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero: bool
: Set the shift of embedding net input to zero.

activation_function: str
: The activation function in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision: str
: The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

uniform_seed: bool
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

attn: int
: The length of hidden vector during scale-dot attention computation.

attn_layer: int
: The number of layers in attention mechanism.

attn_dotr: bool
: Whether to dot the relative coordinates on the attention weights as a gated scheme.

attn_mask: bool
: Whether to mask the diagonal in the attention weights.

ln_eps: float, Optional
: The epsilon value for layer normalization.

multi_task: bool
: If the model has multi fitting nets to train.

tebd_input_mode: str
The input mode of the type embedding. Supported modes are [“concat”, “strip”]. - “concat”: Concatenate the type embedding with the smoothed radial information as the union input for the embedding network. - “strip”: Use a separated embedding network for the type embedding and combine the output with the radial embedding network output. Default value will be strip in se_atten_v2 descriptor.

smooth_type_embedding: bool
Whether to use smooth process in attention weights calculation. And when using stripped type embedding, whether to dot smooth factor on the network output of type embedding to keep the network smooth, instead of setting set_davg_zero to be True. Default value will be True in se_atten_v2 descriptor.

stripped_type_embedding: bool, Optional
(Deprecated, kept only for compatibility.) Whether to strip the type embedding into a separate embedding network. Setting this parameter to True is equivalent to setting tebd_input_mode to ‘strip’. Setting it to False is equivalent to setting tebd_input_mode to ‘concat’. The default value is None, which means the tebd_input_mode setting will be used instead.

Raises

ValueError
if ntypes is 0.

property explicit_ntypes: bool

Explicit ntypes with type embedding.

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, mixed_type: bool = False, real_natoms_vec: list | None = None, **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. If mixed_type is True, this para is blank. See real_natoms_vec.

mesh
The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
Dictionary for additional input

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

real_natoms_vec
If mixed_type is True, it takes in the real natoms_vec for each frame.

**kwargs
Additional keyword arguments.

enable_compression(min_nbor_dist: float, graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, table_extrapolate: float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
The nearest distance between atoms

graph
[tf.Graph] The graph of the model

graph_def
[tf.GraphDef] The graph_def of the model

table_extrapolate
The scale of model extrapolation

table_stride_1
The uniform stride of the first table

table_stride_2
The uniform stride of the second table

check_frequency
The overflow check frequency

suffix
[str, optional] The suffix of the scope

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

_pass_filter(inputs, atype, natoms, input_dict, reuse=None, suffix='', trainable=True)

_compute_dstats_sys_smth(data_coord, data_box, data_atype, natoms_vec, mesh, mixed_type=False, real_natoms_vec=None)

_lookup_type_embedding(xyz_scatter, natype, type_embedding)

Concatenate type_embedding of neighbors and xyz_scatter. If not self.type_one_side, concatenate type_embedding of center atoms as well.

Parameters

xyz_scatter:
shape is [nframes*natoms[0]*self.nnei, 1]

natype:
neighbor atom type

type_embedding:
shape is [self.ntypes, Y] where Y=jdata['type_embedding']['neuron'][-1]

Returns

embedding:
environment of each atom represented by embedding.

_scaled_dot_attn(Q, K, V, temperature, input_r, dotr=False, do_mask=False, layer=0, save_weights=True)

_attention_layers(input_xyz, layer_num, shape_i, outputs_size, input_r, dotr=False, do_mask=False, trainable=True, suffix='')

_filter_lower(type_i, type_input, start_index, incrs_index, inputs, type_embedding=None, atype=None, is_exclude=False, activation_fn=None, bavg=0.0, stddev=1.0, trainable=True, suffix='', name='filter_', reuse=None)

Input env matrix, returns R.G.

_filter(inputs, type_input, natoms, type_embedding=None, atype=None, activation_fn=tf.nn.tanh, stddev=1.0, bavg=0.0, suffix='', name='linear', reuse=None, trainable=True)

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the embedding net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str, optional] The suffix of the scope

build_type_exclude_mask_mixed(exclude_types: Set[Tuple[int, int]], ntypes: int, sel: List[int], ndescrpt: int, atype: deepmd.tf.env.tf.Tensor, shape0: deepmd.tf.env.tf.Tensor, nei_type_vec: deepmd.tf.env.tf.Tensor) → deepmd.tf.env.tf.Tensor

Build the type exclude mask for the attention descriptor.

Parameters

exclude_types
[List[Tuple[int, int]]] The list of excluded types, e.g. [(0, 1), (1, 0)] means the interaction between type 0 and type 1 is excluded.

ntypes
[int] The number of types.

sel
[List[int]] The list of the number of selected neighbors for each type.

ndescrpt
[int] The number of descriptors for each atom.

atype
[tf.Tensor] The type of atoms, with the size of shape0.

shape0
[tf.Tensor] The shape of the first dimension of the inputs, which is equal to nsamples * natoms.

nei_type_vec
[tf.Tensor] The type of neighbors, with the size of (shape0, nnei).

Returns

tf.Tensor
The type exclude mask, with the shape of (shape0, ndescrpt), and the precision of GLOBAL_TF_FLOAT_PRECISION. The mask has the value of 1 if the interaction between two types is not excluded, and 0 otherwise.

See also:

deepmd.tf.descriptor.descriptor.Descriptor.build_type_exclude_mask

Notes

This method has the similiar way to build the type exclude mask as *deepmd.tf.descriptor.descriptor.Descriptor.build_type_exclude_mask()*. The mathmatical expression has been explained in that method. The difference is that the attention descriptor has provided the type of the neighbors (idx_j) that is not in order, so we use it from an extra input.

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

serialize_attention_layers(nlayer: int, nnei: int, embed_dim: int, hidden_dim: int, dotr: bool, do_mask: bool, trainable_ln: bool, ln_eps: float, variables: dict, bias: bool = True, suffix: str = '') → dict

serialize_network_strip(ntypes: int, ndim: int, in_dim: int, neuron: List[int], activation_function: str, resnet_dt: bool, variables: dict, suffix: str = '', type_one_side: bool = False) → dict

Serialize network.

Parameters

ntypes
[int] The number of types

ndim
[int] The dimension of elements

in_dim
[int] The input dimension

neuron
[List[int]] The neuron list

activation_function
[str] The activation function

resnet_dt
[bool] Whether to use resnet

variables
[dict] The input variables

suffix
[str, optional] The suffix of the scope

type_one_side
[bool, optional] If 'False', type embeddings of both neighbor and central atoms are considered. If 'True', only type embeddings of neighbor atoms are considered. Default is 'False'.

Returns

dict
The converted network data

classmethod deserialize_attention_layers(data: dict, suffix: str = '') → dict

Deserialize attention layers.

Parameters

data
[dict] The input attention layer data

suffix
[str, optional] The suffix of the scope

Returns

variables
[dict] The input variables

classmethod deserialize_network_strip(data: dict, suffix: str = '', type_one_side: bool = False) → dict

Deserialize network.

Parameters

data
[dict] The input network data

suffix
[str, optional] The suffix of the scope

type_one_side
[bool, optional] If 'False', type embeddings of both neighbor and central atoms are considered. If 'True', only type embeddings of neighbor atoms are considered. Default is 'False'.

Returns

variables
[dict] The input variables

classmethod deserialize(data: dict, suffix: str = '')

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

Model
The deserialized model

serialize(suffix: str = '') → dict

Serialize the model.

Parameters

suffix
[str, optional] The suffix of the scope

Returns

dict
The serialized data

class deepmd.tf.descriptor.DescrptSeAttenV2(rcut: float, rcut_smth: float, sel: int, ntypes: int, neuron: List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, set_davg_zero: bool = False, exclude_types: List[List[int]] = [], activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, attn: int = 128, attn_layer: int = 2, attn_dotr: bool = True, attn_mask: bool = False, multi_task: bool = False, **kwargs)

Bases: *deepmd.tf.descriptor.se_atten.DescrptSeAtten*

Smooth version 2.0 descriptor with attention.

Parameters

rcut
: The cut-off radius r_c

rcut_smth
: From where the environment matrix should be smoothed r_s

sel
: [int] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

attn
: The length of hidden vector during scale-dot attention computation.

attn_layer
: The number of layers in attention mechanism.

attn_dotr
: Whether to dot the relative coordinates on the attention weights as a gated scheme.

attn_mask
: Whether to mask the diagonal in the attention weights.

multi_task
: If the model has multi fitting nets to train.

class deepmd.tf.descriptor.DescrptSeR(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, type_one_side: bool = True, exclude_types: List[List[int]] = [], set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, multi_task: bool = False, spin: deepmd.tf.utils.spin.Spin | None = None, env_protection: float = 0.0, **kwargs)

Bases: *deepmd.tf.descriptor.se.DescrptSe*

DeepPot-SE constructed from radial information of atomic configurations.

The embedding takes the distance between atoms as input.

Parameters

rcut
: The cut-off radius

rcut_smth
: From where the environment matrix should be smoothed

sel
: [list[int]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

activation_function
: The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

get_rcut()

Returns the cut-off radius.

get_ntypes()

Returns the number of atom types.

get_dim_out()

Returns the output dimension of this descriptor.

get_nlist()

Returns neighbor information.

Returns

nlist
Neighbor list

rij
The relative distance between the neighbor and the center atom.

sel_a
The number of neighbors with full information

sel_r
The number of neighbors with only radial information

compute_input_stats(data_coord, data_box, data_atype, natoms_vec, mesh, input_dict, **kwargs)

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
Dictionary for additional input

**kwargs
Additional keyword arguments.

merge_input_stats(stat_dict)

Merge the statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict

The dict of statisitcs computed from compute_input_stats, including:

sumr
: The sum of radial statisitcs.

sumn
: The sum of neighbor numbers.

sumr2
: The sum of square of radial statisitcs.

enable_compression(min_nbor_dist: float, graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, table_extrapolate: float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
: The nearest distance between atoms

graph
: [`tf.Graph`] The graph of the model

graph_def
: [`tf.GraphDef`] The graph_def of the model

table_extrapolate
: The scale of model extrapolation

table_stride_1
: The uniform stride of the first table

table_stride_2
: The uniform stride of the second table

check_frequency
: The overflow check frequency

suffix
: [`str`, `optional`] The suffix of the scope

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
: The coordinate of atoms

atype_
: The type of atoms

natoms
: The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
: [`tf.Tensor`] The box of the system

mesh
: For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
: Dictionary for additional inputs

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

descriptor
: The output descriptor

prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
: The atomic energy

natoms
: The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
: The force on atoms

virial
: The total virial

atom_virial
: The atomic virial

_pass_filter(inputs, atype, natoms, reuse=None, suffix='', trainable=True)

_compute_dstats_sys_se_r(data_coord, data_box, data_atype, natoms_vec, mesh)

_compute_std(sumv2, sumv, sumn)

_filter_r(inputs, type_input, natoms, activation_fn=tf.nn.tanh, stddev=1.0, bavg=0.0, name='linear', reuse=None, trainable=True)

classmethod deserialize(data: dict, suffix: str = '')

Deserialize the model.

Parameters

data
: [dict] The serialized data

Returns

Model
The deserialized model

serialize(suffix: str = '') → dict

Serialize the model.

Parameters

suffix
[str, optional] The suffix of the scope

Returns

dict
The serialized data

class deepmd.tf.descriptor.DescrptSeT(rcut: float, rcut_smth: float, sel: List[int], neuron: List[int] = [24, 48, 96], resnet_dt: bool = False, trainable: bool = True, seed: int | None = None, set_davg_zero: bool = False, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, multi_task: bool = False, **kwargs)

Bases: *deepmd.tf.descriptor.se.DescrptSe*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations.

The embedding takes angles between two neighboring atoms as input.

Parameters

rcut
The cut-off radius

rcut_smth
From where the environment matrix should be smoothed

sel
[list[int]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
[list[int]] Number of neurons in each hidden layers of the embedding net

resnet_dt
Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
If the weights of embedding net are trainable.

seed
Random seed for initializing the network parameters.

set_davg_zero
Set the shift of embedding net input to zero.

activation_function
The activation function in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

uniform_seed
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

get_rcut() → float

Returns the cut-off radius.

get_ntypes() → int

Returns the number of atom types.

get_dim_out() → int

Returns the output dimension of this descriptor.

get_nlist() → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, List[int], List[int]]

Returns neighbor information.

Returns

nlist
Neighbor list

rij
The relative distance between the neighbor and the center atom.

sel_a
The number of neighbors with full information

sel_r
The number of neighbors with only radial information

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord
The coordinates. Can be generated by deepmd.tf.model.make_stat_input

data_box
The box. Can be generated by deepmd.tf.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.tf.model.make_stat_input

natoms_vec
The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.tf.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.tf.model.make_stat_input

input_dict
Dictionary for additional input

**kwargs
Additional keyword arguments.

`merge_input_stats`(stat_dict)

Merge the statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict

The dict of statisitcs computed from compute_input_stats, including:

sumr
The sum of radial statisitcs.

suma
The sum of relative coord statisitcs.

sumn
The sum of neighbor numbers.

sumr2
The sum of square of radial statisitcs.

suma2
The sum of square of relative coord statisitcs.

`enable_compression`(min_nbor_dist: float, graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, table_extrapolate: float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
The nearest distance between atoms

graph
[`tf.Graph`] The graph of the model

graph_def
[`tf.GraphDef`] The graph_def of the model

table_extrapolate
The scale of model extrapolation

table_stride_1
The uniform stride of the first table

table_stride_2
The uniform stride of the second table

check_frequency
The overflow check frequency

suffix
[`str`, `optional`] The suffix of the scope

`build`(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for the descriptor.

Parameters

coord_
The coordinate of atoms

atype_
The type of atoms

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box_
[tf.Tensor] The box of the system

mesh
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input_dict
Dictionary for additional inputs

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

descriptor
The output descriptor

prod_force_virial(atom_ener: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Compute force and virial.

Parameters

atom_ener
The atomic energy

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
The total virial

atom_virial
The atomic virial

_pass_filter(inputs, atype, natoms, input_dict, reuse=None, suffix='', trainable=True)

_compute_dstats_sys_smth(data_coord, data_box, data_atype, natoms_vec, mesh)

_compute_std(sumv2, sumv, sumn)

_filter(inputs, type_input, natoms, activation_fn=tf.nn.tanh, stddev=1.0, bavg=0.0, name='linear', reuse=None, trainable=True)

deepmd.tf.entrypoints

Submodule that contains all the DeePMD-Kit entry point scripts.

Submodules

deepmd.tf.entrypoints.compress

Compress a model, which including tabulating the embedding-net.

Module Contents

Functions

compress(*, input, output, extrapolate, step, ...)	Compress model.

deepmd.tf.entrypoints.compress.compress(*, input: str, output: str, extrapolate: int, step: float, frequency: str, checkpoint_folder: str, training_script: str, mpi_log: str, log_path: str | None, log_level: int, **kwargs)

Compress model.

The table is composed of fifth-order polynomial coefficients and is assembled from two sub-tables. The first table takes the step parameter as the domain's uniform step size, while the second table takes 10 * step as it's uniform step size. The range of the first table is automatically detected by the code, while the second table ranges from the first table's upper boundary(upper) to the extrapolate(parameter) * upper.

Parameters

input
[str] frozen model file to compress

output
[str] compressed model filename

extrapolate
[int] scale of model extrapolation

step
[float] uniform step size of the tabulation's first table

frequency
[str] frequency of tabulation overflow check

checkpoint_folder
[str] trining checkpoint folder for freezing

training_script
: [str] training script of the input frozen model

mpi_log
: [str] mpi logging mode for training

log_path
: [Optional[str]] if speccified log will be written to this file

log_level
: [int] logging level

**kwargs
: additional arguments

deepmd.tf.entrypoints.convert

Module Contents

Functions

convert(*, FROM, input_model, output_model, **kwargs)	

deepmd.tf.entrypoints.convert.**convert**(*, FROM: str, input_model: str, output_model: str, **kwargs)

deepmd.tf.entrypoints.doc

Module Contents

Functions

doc_train_input(*[, out_type])	Print out trining input arguments to console.

deepmd.tf.entrypoints.doc.**doc_train_input**(*, out_type: str = 'rst', **kwargs)

Print out trining input arguments to console.

deepmd.tf.entrypoints.freeze

Script for freezing TF trained graph so it can be used with LAMMPS and i-PI.

References

https://blog.metaflow.fr/tensorflow-how-to-freeze-a-model-and-serve-it-with-a-python-api-d4f3596b3adc

Module Contents

Functions

freeze(*, checkpoint_folder, output[, node_names, ...])	Freeze the graph in supplied folder.

deepmd.tf.entrypoints.freeze.**freeze**(*, checkpoint_folder: str, output: str, node_names: str | None = None, nvnmd_weight: str | None = None, united_model: bool = False, **kwargs)

Freeze the graph in supplied folder.

Parameters

checkpoint_folder
: [str] location of either the folder with checkpoint or the checkpoint prefix

output
: [str] output file name

node_names
: [Optional[str], optional] names of nodes to output, by default None

nvnmd_weight
: [Optional[str], optional] nvnmd weight file

united_model
: [bool] when in multi-task mode, freeze all nodes into one unit model

**kwargs
: other arguments

deepmd.tf.entrypoints.gui

Module Contents

Functions

start_dpgui(*, port, bind_all, **kwargs)	Host DP-GUI server.

deepmd.tf.entrypoints.gui.**start_dpgui**(*, port: int, bind_all: bool, **kwargs)

Host DP-GUI server.

Parameters

port
: [int] The port to serve DP-GUI on.

bind_all
: [bool] Serve on all public interfaces. This will expose your DP-GUI instance to the network on both IPv4 and IPv6 (where available).

**kwargs
: additional arguments

Raises

`ModuleNotFoundError`
: The dpgui package is not installed

deepmd.tf.entrypoints.ipi

Use dp_ipi inside the Python package.

Module Contents

Functions

_program(name, args)	Execute a program.
dp_ipi()	dp_ipi.

Attributes

ROOT_DIR

deepmd.tf.entrypoints.ipi.**ROOT_DIR**

deepmd.tf.entrypoints.ipi.**_program**(name: str, args: List[str])
: Execute a program.

  Parameters

  name
  : [`str`] the name of the program

  args
  : [`list of str`] list of arguments

deepmd.tf.entrypoints.ipi.**dp_ipi**()
: dp_ipi.

deepmd.tf.entrypoints.main

DeePMD-Kit entry point module.

Module Contents

Functions

get_ll(→ int)	Convert string to python logging level.
main_parser(→ argparse.ArgumentParser)	DeePMD-Kit commandline options argument parser.
parse_args(→ argparse.Namespace)	Parse arguments and convert argument strings to objects.
main([args])	DeePMD-Kit entry point.

deepmd.tf.entrypoints.main.**get_ll**(log_level: str) → int

Convert string to python logging level.

Parameters

log_level
[str] allowed input values are: DEBUG, INFO, WARNING, ERROR, 3, 2, 1, 0

Returns

int
one of python logging module log levels - 10, 20, 30 or 40

deepmd.tf.entrypoints.main.**main_parser**() → argparse.ArgumentParser

DeePMD-Kit commandline options argument parser.

Returns

argparse.ArgumentParser
main parser of DeePMD-kit

deepmd.tf.entrypoints.main.**parse_args**(args: List[str] | None = None) → argparse.Namespace

Parse arguments and convert argument strings to objects.

Parameters

args
[List[str]] list of command line arguments, main purpose is testing default option None takes arguments from sys.argv

Returns

argparse.Namespace
the populated namespace

deepmd.tf.entrypoints.main.**main**(args: List[str] | argparse.Namespace | None = None)

DeePMD-Kit entry point.

Parameters

args
: [List[str] or argparse.Namespace, optional] list of command line arguments, used to avoid calling from the subprocess, as it is quite slow to import tensorflow; if Namespace is given, it will be used directly

Raises

RuntimeError
: if no command was input

deepmd.tf.entrypoints.neighbor_stat

Module Contents

Functions

neighbor_stat(*, system, rcut, type_map[, mixed_type, ...])	Calculate neighbor statistics.

deepmd.tf.entrypoints.neighbor_stat.**neighbor_stat**(*, system: str, rcut: float, type_map: List[str], mixed_type: bool = False, backend: str = 'tensorflow', **kwargs)

Calculate neighbor statistics.

Parameters

system
: [str] system to stat

rcut
: [float] cutoff radius

type_map
: [list[str]] type map

mixed_type
: [bool, optional, default=False] treat all types as a single type

backend
: [str, optional, default="tensorflow"] backend to use

**kwargs
: additional arguments

Examples

```
>>> neighbor_stat(
...     system=".",
...     rcut=6.0,
...     type_map=[
...         "C",
...         "H",
...         "O",
...         "N",
```

(continues on next page)

(continued from previous page)

```
...         "P",
...         "S",
...         "Mg",
...         "Na",
...         "HW",
...         "OW",
...         "mNa",
...         "mCl",
...         "mC",
...         "mH",
...         "mMg",
...         "mN",
...         "mO",
...         "mP",
...     ],
... )
min_nbor_dist: 0.6599510670195264
max_nbor_size: [23, 26, 19, 16, 2, 2, 1, 1, 72, 37, 5, 0, 31, 29, 1, 21, 20, 5]
```

deepmd.tf.entrypoints.test

Module Contents

Functions

test(*, model, system, datafile, numb_test, rand_seed, ...)	Test model predictions.

deepmd.tf.entrypoints.test.**test**(*, model: str, system: str, datafile: str, numb_test: int, rand_seed: int | None, shuffle_test: bool, detail_file: str, atomic: bool, head: str | None = None, **kwargs)

Test model predictions.

Parameters

model
: [str] path where model is stored

system
: [str] system directory

datafile
: [str] the path to the list of systems to test

numb_test
: [int] munber of tests to do. 0 means all data.

rand_seed
: [Optional[int]] seed for random generator

shuffle_test
: [bool] whether to shuffle tests

detail_file
: [Optional[str]] file where test details will be output

atomic
: [bool] whether per atom quantities should be computed

head
: [Optional[str], optional] (Supported backend: PyTorch) Task head to test if in multi-task mode.

**kwargs
: additional arguments

Raises

RuntimeError
: if no valid system was found

deepmd.tf.entrypoints.train

DeePMD training entrypoint script.

Can handle local or distributed training.

Module Contents

Functions

train(*, INPUT, init_model, restart, output, ...[, ...])	Run DeePMD model training.

deepmd.tf.entrypoints.train.train(*, INPUT: str, init_model: str | None, restart: str | None, output: str, init_frz_model: str, mpi_log: str, log_level: int, log_path: str | None, is_compress: bool = False, skip_neighbor_stat: bool = False, finetune: str | None = None, **kwargs)

Run DeePMD model training.

Parameters

INPUT
: [str] json/yaml control file

init_model
: [Optional[str]] path prefix of checkpoint files or None

restart
: [Optional[str]] path prefix of checkpoint files or None

output
: [str] path for dump file with arguments

init_frz_model
: [str] path to frozen model or None

mpi_log
: [str] mpi logging mode

log_level
[int] logging level defined by int 0-3

log_path
[Optional[str]] logging file path or None if logs are to be output only to stdout

is_compress
[bool] indicates whether in the model compress mode

skip_neighbor_stat
[bool, default=False] skip checking neighbor statistics

finetune
[Optional[str]] path to pretrained model or None

**kwargs
additional arguments

Raises

RuntimeError
if distributed training job name is wrong

deepmd.tf.entrypoints.transfer

Module used for transfering parameters between models.

Module Contents

Functions

transfer(*, old_model, raw_model, output, **kwargs)	Transfer operation from old fron graph to new prepared raw graph.

deepmd.tf.entrypoints.transfer.**transfer**(*, old_model: str, raw_model: str, output: str, **kwargs)

Transfer operation from old fron graph to new prepared raw graph.

Parameters

old_model
[str] frozen old graph model

raw_model
[str] new model that will accept ops from old model

output
[str] new model with transfered parameters will be saved to this location

**kwargs
additional arguments

Package Contents

Functions

make_model_devi(*, models, system, output, frequency)	Make model deviation calculation.
compress	Compress a model, which including tabulating the embedding-net.
convert	
doc_train_input(*[, out_type])	Print out trining input arguments to console.
freeze(*, checkpoint_folder, output[, node_names, ...])	Freeze the graph in supplied folder.
start_dpgui(*, port, bind_all, **kwargs)	Host DP-GUI server.
neighbor_stat(*, system, rcut, type_map[, mixed_type, ...])	Calculate neighbor statistics.
test(*, model, system, datafile, numb_test, rand_seed, ...)	Test model predictions.
train_dp(*, INPUT, init_model, restart, output, ...[, ...])	Run DeePMD model training.
transfer(*, old_model, raw_model, output, **kwargs)	Transfer operation from old fron graph to new prepared raw graph.

deepmd.tf.entrypoints.make_model_devi(*, models: list, system: str, output: str, frequency: int, real_error: bool = False, atomic: bool = False, relative: float | None = None, relative_v: float | None = None, **kwargs)

Make model deviation calculation.

Parameters

models
: [list] A list of paths of models to use for making model deviation

system
: [str] The path of system to make model deviation calculation

output
: [str] The output file for model deviation results

frequency
: [int] The number of steps that elapse between writing coordinates in a trajectory by a MD engine (such as Gromacs / Lammps). This paramter is used to determine the index in the output file.

real_error
: [bool, default: False] If True, calculate the RMS real error instead of model deviation.

atomic
: [bool, default: False] If True, calculate the force model deviation of each atom.

relative
: [float, default: None] If given, calculate the relative model deviation of force. The value is the level parameter for computing the relative model deviation of the force.

relative_v
[float, default: None] If given, calculate the relative model deviation of virial. The value is the level parameter for computing the relative model deviation of the virial.

**kwargs
Arbitrary keyword arguments.

deepmd.tf.entrypoints.compress(*, input: str, output: str, extrapolate: int, step: float, frequency: str, checkpoint_folder: str, training_script: str, mpi_log: str, log_path: str | None, log_level: int, **kwargs)

Compress model.

The table is composed of fifth-order polynomial coefficients and is assembled from two sub-tables. The first table takes the step parameter as the domain's uniform step size, while the second table takes 10 * step as it's uniform step size. The range of the first table is automatically detected by the code, while the second table ranges from the first table's upper boundary(upper) to the extrapolate(parameter) * upper.

Parameters

input
[str] frozen model file to compress

output
[str] compressed model filename

extrapolate
[int] scale of model extrapolation

step
[float] uniform step size of the tabulation's first table

frequency
[str] frequency of tabulation overflow check

checkpoint_folder
[str] trining checkpoint folder for freezing

training_script
[str] training script of the input frozen model

mpi_log
[str] mpi logging mode for training

log_path
[Optional[str]] if speccified log will be written to this file

log_level
[int] logging level

**kwargs
additional arguments

deepmd.tf.entrypoints.convert(*, FROM: str, input_model: str, output_model: str, **kwargs)

deepmd.tf.entrypoints.doc_train_input(*, out_type: str = 'rst', **kwargs)

Print out trining input arguments to console.

deepmd.tf.entrypoints.freeze(*, checkpoint_folder: str, output: str, node_names: str | None = None, nvnmd_weight: str | None = None, united_model: bool = False, **kwargs)

Freeze the graph in supplied folder.

Parameters

checkpoint_folder
[str] location of either the folder with checkpoint or the checkpoint prefix

output
[str] output file name

node_names
[Optional[str], optional] names of nodes to output, by default None

nvnmd_weight
[Optional[str], optional] nvnmd weight file

united_model
[bool] when in multi-task mode, freeze all nodes into one unit model

**kwargs
other arguments

deepmd.tf.entrypoints.start_dpgui(*, port: int, bind_all: bool, **kwargs)

Host DP-GUI server.

Parameters

port
[int] The port to serve DP-GUI on.

bind_all
[bool] Serve on all public interfaces. This will expose your DP-GUI instance to the network on both IPv4 and IPv6 (where available).

**kwargs
additional arguments

Raises

ModuleNotFoundError
The dpgui package is not installed

deepmd.tf.entrypoints.neighbor_stat(*, system: str, rcut: float, type_map: List[str], mixed_type: bool = False, backend: str = 'tensorflow', **kwargs)

Calculate neighbor statistics.

Parameters

system
[str] system to stat

rcut
[float] cutoff radius

type_map
[list[str]] type map

mixed_type
[bool, optional, default=False] treat all types as a single type

backend
[str, optional, default="tensorflow"] backend to use

**kwargs
additional arguments

Examples

```
>>> neighbor_stat(
...     system=".",
...     rcut=6.0,
...     type_map=[
...         "C",
...         "H",
...         "O",
...         "N",
...         "P",
...         "S",
...         "Mg",
...         "Na",
...         "HW",
...         "OW",
...         "mNa",
...         "mCl",
...         "mC",
...         "mH",
...         "mMg",
...         "mN",
...         "mO",
...         "mP",
...     ],
... )
min_nbor_dist: 0.6599510670195264
max_nbor_size: [23, 26, 19, 16, 2, 2, 1, 1, 72, 37, 5, 0, 31, 29, 1, 21, 20, 5]
```

deepmd.tf.entrypoints.test(*, model: str, system: str, datafile: str, numb_test: int, rand_seed: int | None, shuffle_test: bool, detail_file: str, atomic: bool, head: str | None = None, **kwargs)

Test model predictions.

Parameters

model
: [str] path where model is stored

system
: [str] system directory

datafile
: [str] the path to the list of systems to test

numb_test
: [int] munber of tests to do. 0 means all data.

rand_seed
: [Optional[int]] seed for random generator

shuffle_test
: [bool] whether to shuffle tests

detail_file
: [Optional[str]] file where test details will be output

atomic
: [bool] whether per atom quantities should be computed

head
[Optional[str], optional] (Supported backend: PyTorch) Task head to test if in multi-task mode.

**kwargs
additional arguments

Raises

RuntimeError
if no valid system was found

deepmd.tf.entrypoints.train_dp(*, INPUT: str, init_model: str | None, restart: str | None, output: str, init_frz_model: str, mpi_log: str, log_level: int, log_path: str | None, is_compress: bool = False, skip_neighbor_stat: bool = False, finetune: str | None = None, **kwargs)

Run DeePMD model training.

Parameters

INPUT
[str] json/yaml control file

init_model
[Optional[str]] path prefix of checkpoint files or None

restart
[Optional[str]] path prefix of checkpoint files or None

output
[str] path for dump file with arguments

init_frz_model
[str] path to frozen model or None

mpi_log
[str] mpi logging mode

log_level
[int] logging level defined by int 0-3

log_path
[Optional[str]] logging file path or None if logs are to be output only to stdout

is_compress
[bool] indicates whether in the model compress mode

skip_neighbor_stat
[bool, default=False] skip checking neighbor statistics

finetune
[Optional[str]] path to pretrained model or None

**kwargs
additional arguments

Raises

RuntimeError
if distributed training job name is wrong

deepmd.tf.entrypoints.transfer(*, old_model: str, raw_model: str, output: str, **kwargs)

Transfer operation from old fron graph to new prepared raw graph.

Parameters

old_model
[str] frozen old graph model

raw_model
[str] new model that will accept ops from old model

output
[str] new model with transfered parameters will be saved to this location

**kwargs
additional arguments

deepmd.tf.fit

Submodules

deepmd.tf.fit.dipole

Module Contents

Classes

DipoleFittingSeA	Fit the atomic dipole with descriptor se_a.

class deepmd.tf.fit.dipole.DipoleFittingSeA(ntypes: int, dim_descrpt: int, embedding_width: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, sel_type: List[int] | None = None, seed: int | None = None, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, mixed_types: bool = False, **kwargs)

Bases: *deepmd.tf.fit.fitting.Fitting*

Fit the atomic dipole with descriptor se_a.

Parameters

ntypes
The ntypes of the descrptor $\mathcal{D}$

dim_descrpt
The dimension of the descrptor $\mathcal{D}$

embedding_width
The rotation matrix dimension of the descrptor $\mathcal{D}$

neuron
[List[int]] Number of neurons in each hidden layer of the fitting net

resnet_dt
[bool] Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

sel_type
[List[int]] The atom types selected to have an atomic dipole prediction. If is None, all atoms are selected.

seed
: [int] Random seed for initializing the network parameters.

activation_function
: [str] The activation function in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: [str] The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

mixed_types
: [bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

get_sel_type() → int

Get selected type.

get_out_size() → int

Get the output size. Should be 3.

_build_lower(start_index, natoms, inputs, rot_mat, suffix='', reuse=None)

build(input_d: deepmd.tf.env.tf.Tensor, rot_mat: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, input_dict: dict | None = None, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for fitting net.

Parameters

input_d
: The input descriptor

rot_mat
: The rotation matrix from the descriptor.

natoms
: The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

input_dict
: Additional dict for inputs.

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

dipole
: The atomic dipole.

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str] suffix to name scope

enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss

Get the loss function.

Parameters

loss
[dict] the loss dict

lr
[LearningRateExp] the learning rate

Returns

Loss
the loss function

serialize(suffix: str) → dict

Serialize the model.

Returns

dict
The serialized data

classmethod deserialize(data: dict, suffix: str)

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

Model
The deserialized model

deepmd.tf.fit.dos

Module Contents

Classes

DOSFitting	Fitting the density of states (DOS) of the system.

Attributes

log

deepmd.tf.fit.dos.log

class deepmd.tf.fit.dos.**DOSFitting**(ntypes: int, dim_descrpt: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, numb_dos: int = 300, rcond: float | None = None, trainable: List[bool] | None = None, seed: int | None = None, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, layer_name: List[str | None] | None = None, use_aparam_as_mask: bool = False, mixed_types: bool = False, **kwargs)

Bases: *deepmd.tf.fit.fitting.Fitting*

Fitting the density of states (DOS) of the system. The energy should be shifted by the fermi level.

Parameters

ntypes
: The ntypes of the descrptor $\mathcal{D}$

dim_descrpt
: The dimension of the descrptor $\mathcal{D}$

neuron
: Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

! numb_dos (added)
: Number of gridpoints on which the DOS is evaluated (NEDOS in VASP)

rcond
: The condition number for the regression of atomic energy.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

seed
: Random seed for initializing the network parameters.

activation_function
: The activation function $\boldsymbol{\phi}$ in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

layer_name
: [`list`[`Optional`[`str`]], `optional`] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
: If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
: [bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

get_numb_fparam() → int

Get the number of frame parameters.

get_numb_aparam() → int

Get the number of atomic parameters.

get_numb_dos() → int

Get the number of gridpoints in energy space.

compute_output_stats(all_stat: dict, mixed_type: bool = False) → None

Compute the ouput statistics.

Parameters

all_stat
: must have the following components: all_stat['dos'] of shape n_sys x n_batch x n_frame x numb_dos can be prepared by model.make_stat_input

mixed_type
: Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

_compute_output_stats(all_stat, rcond=0.001, mixed_type=False)

compute_input_stats(all_stat: dict, protection: float = 0.01) → None

Compute the input statistics.

Parameters

all_stat
if numb_fparam > 0 must have all_stat['fparam'] if numb_aparam > 0 must have all_stat['aparam'] can be prepared by model.make_stat_input

protection
Divided-by-zero protection

_compute_std(sumv2, sumv, sumn)

_build_lower(start_index, natoms, inputs, fparam=None, aparam=None, bias_dos=0.0, type_suffix='', suffix='', reuse=None)

build(inputs: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, input_dict: dict | None = None, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for fitting net.

Parameters

inputs
The input descriptor

input_dict
Additional dict for inputs. if numb_fparam > 0, should have input_dict['fparam'] if numb_aparam > 0, should have input_dict['aparam']

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

ener
The system energy

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str] suffix to name scope

enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss

Get the loss function.

Parameters

loss
[dict] the loss dict

lr
[LearningRateExp] the learning rate

Returns

Loss
the loss function

classmethod deserialize(data: dict, suffix: str = '')

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

Model
The deserialized model

serialize(suffix: str = '') → dict

Serialize the model.

Returns

dict
The serialized data

deepmd.tf.fit.ener

Module Contents

Classes

EnerFitting	Fitting the energy of the system. The force and the virial can also be trained.

Attributes

log

deepmd.tf.fit.ener.log

class deepmd.tf.fit.ener.**EnerFitting**(ntypes: int, dim_descrpt: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, rcond: float | None = None, tot_ener_zero: bool = False, trainable: List[bool] | None = None, seed: int | None = None, atom_ener: List[float] = [], activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, layer_name: List[str | None] | None = None, use_aparam_as_mask: bool = False, spin: deepmd.tf.utils.spin.Spin | None = None, mixed_types: bool = False, **kwargs)

Bases: *deepmd.tf.fit.fitting.Fitting*

Fitting the energy of the system. The force and the virial can also be trained.

The potential energy E is a fitting network function of the descriptor $\mathcal{D}$:

$$E(\mathcal{D}) = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \cdots \circ \mathcal{L}^{(1)} \circ \mathcal{L}^{(0)}$$

The first n hidden layers $\mathcal{L}^{(0)}, \cdots, \mathcal{L}^{(n-1)}$ are given by

$$\mathrm{y} = \mathcal{L}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \boldsymbol{\phi}(\mathrm{x}^T \mathrm{w} + \mathrm{b})$$

where $\mathrm{x} \in \mathbb{R}^{N_1}$ is the input vector and $\mathrm{y} \in \mathbb{R}^{N_2}$ is the output vector. $\mathrm{w} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathrm{b} \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable[i] is True. $\boldsymbol{\phi}$ is the activation function.

The output layer $\mathcal{L}^{(n)}$ is given by

$$\mathrm{y} = \mathcal{L}^{(n)}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \mathrm{x}^T \mathrm{w} + \mathrm{b}$$

where $\mathrm{x} \in \mathbb{R}^{N_{n-1}}$ is the input vector and $\mathrm{y} \in \mathbb{R}$ is the output scalar. $\mathrm{w} \in \mathbb{R}^{N_{n-1}}$ and $\mathrm{b} \in \mathbb{R}$ are weights and bias, respectively, both of which are trainable if trainable[n] is True.

Parameters

ntypes
: The ntypes of the descrptor $\mathcal{D}$

dim_descrpt
: The dimension of the descrptor $\mathcal{D}$

neuron
: Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

rcond
: The condition number for the regression of atomic energy.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

seed
: Random seed for initializing the network parameters.

atom_ener
: Specifying atomic energy contribution in vacuum. The set_davg_zero key in the descrptor should be set.

activation_function
: The activation function ϕ in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

layer_name
: [`list`[`Optional`[`str`]], `optional`] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
: If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
: [bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

get_numb_fparam() → int

Get the number of frame parameters.

get_numb_aparam() → int

Get the number of atomic parameters.

compute_output_stats(all_stat: dict, mixed_type: bool = False) → None

Compute the ouput statistics.

Parameters

all_stat
: must have the following components: all_stat['energy'] of shape n_sys x n_batch x n_frame can be prepared by model.make_stat_input

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

_compute_output_stats(all_stat, rcond=0.001, mixed_type=False)

compute_input_stats(all_stat: dict, protection: float = 0.01) → None
Compute the input statistics.

Parameters

all_stat
if numb_fparam > 0 must have all_stat['fparam'] if numb_aparam > 0 must have all_stat['aparam'] can be prepared by model.make_stat_input

protection
Divided-by-zero protection

_compute_std(sumv2, sumv, sumn)

_build_lower(start_index, natoms, inputs, fparam=None, aparam=None, bias_atom_e=0.0, type_suffix='', suffix='', reuse=None)

build(inputs: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, input_dict: dict | None = None, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor
Build the computational graph for fitting net.

Parameters

inputs
The input descriptor

input_dict
Additional dict for inputs. if numb_fparam > 0, should have input_dict['fparam'] if numb_aparam > 0, should have input_dict['aparam']

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

ener
The system energy

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None
Init the fitting net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str] suffix to name scope

change_energy_bias(data, frozen_model, origin_type_map, full_type_map, bias_adjust_mode='change-by-statistic', ntest=10) → None

enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss

Get the loss function.

Parameters

loss
[dict] The loss function parameters.

lr
[LearningRateExp] The learning rate.

Returns

Loss
The loss function.

classmethod deserialize(data: dict, suffix: str = '')

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

Model
The deserialized model

serialize(suffix: str = '') → dict

Serialize the model.

Returns

dict
The serialized data

deepmd.tf.fit.fitting

Module Contents

Classes

Fitting	A class to remove type from input arguments.

class deepmd.tf.fit.fitting.Fitting

Bases: *deepmd.tf.utils.PluginVariant*, make_plugin_registry('fitting')

A class to remove type from input arguments.

property precision: deepmd.tf.env.tf.DType

Precision of fitting network.

abstract init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str] suffix to name scope

Notes

This method is called by others when the fitting supported initialization from the given variables.

abstract get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss

Get the loss function.

Parameters

loss
[dict] the loss dict

lr
[LearningRateExp] the learning rate

Returns

Loss
the loss function

classmethod deserialize(data: dict, suffix: str = '') → Fitting

Deserialize the fitting.

There is no suffix in a native DP model, but it is important for the TF backend.

Parameters

data
[dict] The serialized data

suffix
[str, optional] Name suffix to identify this fitting

Returns

Fitting
The deserialized fitting

abstract serialize(suffix: str = '') → dict

Serialize the fitting.

There is no suffix in a native DP model, but it is important for the TF backend.

Returns

dict
The serialized data

suffix
[str, optional] Name suffix to identify this fitting

serialize_network(ntypes: int, ndim: int, in_dim: int, neuron: List[int], activation_function: str, resnet_dt: bool, variables: dict, out_dim: int | None = 1, suffix: str = '') → dict

Serialize network.

Parameters

ntypes
[int] The number of types

ndim
[int] The dimension of elements

in_dim
[int] The input dimension

neuron
[List[int]] The neuron list

activation_function
[str] The activation function

resnet_dt
[bool] Whether to use resnet

variables
[dict] The input variables

suffix
[str, optional] The suffix of the scope

out_dim
[int, optional] The output dimension

Returns

dict
The converted network data

classmethod deserialize_network(data: dict, suffix: str = '') → dict

Deserialize network.

Parameters

data
[dict] The input network data

suffix
[str, optional] The suffix of the scope

Returns

variables
[dict] The input variables

deepmd.tf.fit.polar

Module Contents

Classes

PolarFittingSeA	Fit the atomic polarizability with descriptor se_a.
GlobalPolarFittingSeA	Fit the system polarizability with descriptor se_a.

class deepmd.tf.fit.polar.PolarFittingSeA(ntypes: int, dim_descrpt: int, embedding_width: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, sel_type: List[int] | None = None, fit_diag: bool = True, scale: List[float] | None = None, shift_diag: bool = True, seed: int | None = None, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, mixed_types: bool = False, **kwargs)

Bases: *deepmd.tf.fit.fitting.Fitting*

Fit the atomic polarizability with descriptor se_a.

Parameters

ntypes
The ntypes of the descrptor $\mathcal{D}$

dim_descrpt
The dimension of the descrptor $\mathcal{D}$

embedding_width
The rotation matrix dimension of the descrptor $\mathcal{D}$

neuron
[List[int]] Number of neurons in each hidden layer of the fitting net

resnet_dt
[bool] Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

sel_type
[List[int]] The atom types selected to have an atomic polarizability prediction. If is None, all atoms are selected.

fit_diag
: [bool] Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.

scale
: [List[float]] The output of the fitting net (polarizability matrix) for type i atom will be scaled by scale[i]

diag_shift
: [List[float]] The diagonal part of the polarizability matrix of type i will be shifted by diag_shift[i]. The shift operation is carried out after scale.

seed
: [int] Random seed for initializing the network parameters.

activation_function
: [str] The activation function in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: [str] The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

mixed_types
: [bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

get_sel_type() → List[int]

Get selected atom types.

get_out_size() → int

Get the output size. Should be 9.

compute_output_stats(all_stat)

Compute the output statistics.

Parameters

all_stat
: Dictionary of inputs. can be prepared by model.make_stat_input

_build_lower(start_index, natoms, inputs, rot_mat, suffix='', reuse=None)

build(input_d: deepmd.tf.env.tf.Tensor, rot_mat: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, input_dict: dict | None = None, reuse: bool | None = None, suffix: str = '')

Build the computational graph for fitting net.

Parameters

input_d
: The input descriptor

rot_mat
: The rotation matrix from the descriptor.

natoms
: The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

input_dict
: Additional dict for inputs.

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

atomic_polar
: The atomic polarizability

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

graph
: [tf.Graph] The input frozen model graph

graph_def
: [tf.GraphDef] The input frozen model graph_def

suffix
: [str] suffix to name scope

enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
: The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss

Get the loss function.

serialize(suffix: str) → dict

Serialize the model.

Returns

dict
: The serialized data

classmethod deserialize(data: dict, suffix: str)

Deserialize the model.

Parameters

data
: [dict] The serialized data

Returns

Model
The deserialized model

class deepmd.tf.fit.polar.GlobalPolarFittingSeA(descrpt: deepmd.tf.env.tf.Tensor, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, sel_type: List[int] | None = None, fit_diag: bool = True, scale: List[float] | None = None, diag_shift: List[float] | None = None, seed: int | None = None, activation_function: str = 'tanh', precision: str = 'default')

Fit the system polarizability with descriptor se_a.

Parameters

descrpt
[tf.Tensor] The descrptor

neuron
[List[int]] Number of neurons in each hidden layer of the fitting net

resnet_dt
[bool] Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

sel_type
[List[int]] The atom types selected to have an atomic polarizability prediction

fit_diag
[bool] Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.

scale
[List[float]] The output of the fitting net (polarizability matrix) for type i atom will be scaled by scale[i]

diag_shift
[List[float]] The diagonal part of the polarizability matrix of type i will be shifted by diag_shift[i]. The shift operation is carried out after scale.

seed
[int] Random seed for initializing the network parameters.

activation_function
[str] The activation function in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
[str] The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

get_sel_type() → int
Get selected atom types.

get_out_size() → int
Get the output size. Should be 9.

build(input_d, rot_mat, natoms, input_dict: dict | None = None, reuse=None, suffix='') → deepmd.tf.env.tf.Tensor
Build the computational graph for fitting net.

Parameters

input_d
: The input descriptor

rot_mat
: The rotation matrix from the descriptor.

natoms
: The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

input_dict
: Additional dict for inputs.

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

polar
: The system polarizability

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

graph
: [tf.Graph] The input frozen model graph

graph_def
: [tf.GraphDef] The input frozen model graph_def

suffix
: [str] suffix to name scope

enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
: The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss

Get the loss function.

Parameters

loss
: [dict] the loss dict

lr
: [LearningRateExp] the learning rate

Returns

Loss
: the loss function

Package Contents

Classes

DipoleFittingSeA	Fit the atomic dipole with descriptor se_a.
DOSFitting	Fitting the density of states (DOS) of the system.
EnerFitting	Fitting the energy of the system. The force and the virial can also be trained.
Fitting	A class to remove type from input arguments.
GlobalPolarFittingSeA	Fit the system polarizability with descriptor se_a.
PolarFittingSeA	Fit the atomic polarizability with descriptor se_a.

class deepmd.tf.fit.DipoleFittingSeA(ntypes: int, dim_descrpt: int, embedding_width: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, sel_type: List[int] | None = None, seed: int | None = None, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, mixed_types: bool = False, **kwargs)

Bases: *deepmd.tf.fit.fitting.Fitting*

Fit the atomic dipole with descriptor se_a.

Parameters

ntypes
: The ntypes of the descrptor $\mathcal{D}$

dim_descrpt
: The dimension of the descrptor $\mathcal{D}$

embedding_width
: The rotation matrix dimension of the descrptor $\mathcal{D}$

neuron
: [List[int]] Number of neurons in each hidden layer of the fitting net

resnet_dt
: [bool] Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

sel_type
: [List[int]] The atom types selected to have an atomic dipole prediction. If is None, all atoms are selected.

seed
: [int] Random seed for initializing the network parameters.

activation_function
: [str] The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
: [str] The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

mixed_types
[bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

get_sel_type() → int

Get selected type.

get_out_size() → int

Get the output size. Should be 3.

_build_lower(start_index, natoms, inputs, rot_mat, suffix='', reuse=None)

build(input_d: deepmd.tf.env.tf.Tensor, rot_mat: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, input_dict: dict | None = None, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for fitting net.

Parameters

input_d
The input descriptor

rot_mat
The rotation matrix from the descriptor.

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

input_dict
Additional dict for inputs.

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

dipole
The atomic dipole.

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str] suffix to name scope

enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

`get_loss`(loss: dict, lr) → deepmd.tf.loss.loss.Loss

Get the loss function.

Parameters

loss
[`dict`] the loss dict

lr
[`LearningRateExp`] the learning rate

Returns

`Loss`
the loss function

`serialize`(suffix: str) → dict

Serialize the model.

Returns

`dict`
The serialized data

`classmethod deserialize`(data: dict, suffix: str)

Deserialize the model.

Parameters

data
[`dict`] The serialized data

Returns

`Model`
The deserialized model

`class deepmd.tf.fit.DOSFitting`(ntypes: int, dim_descrpt: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, numb_dos: int = 300, rcond: float | None = None, trainable: List[bool] | None = None, seed: int | None = None, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, layer_name: List[str | None] | None = None, use_aparam_as_mask: bool = False, mixed_types: bool = False, **kwargs)

Bases: *deepmd.tf.fit.fitting.Fitting*

Fitting the density of states (DOS) of the system. The energy should be shifted by the fermi level.

Parameters

ntypes
The ntypes of the descrptor $\mathcal{D}$

dim_descrpt
The dimension of the descrptor $\mathcal{D}$

neuron
Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

! numb_dos (added)
: Number of gridpoints on which the DOS is evaluated (NEDOS in VASP)

rcond
: The condition number for the regression of atomic energy.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

seed
: Random seed for initializing the network parameters.

activation_function
: The activation function $\boldsymbol{\phi}$ in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

layer_name
: [`list`[`Optional`[`str`]], `optional`] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
: If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
: [bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

get_numb_fparam() → int
: Get the number of frame parameters.

get_numb_aparam() → int
: Get the number of atomic parameters.

get_numb_dos() → int
: Get the number of gridpoints in energy space.

compute_output_stats(all_stat: dict, mixed_type: bool = False) → None
: Compute the ouput statistics.

  Parameters

all_stat
must have the following components: all_stat['dos'] of shape n_sys x n_batch x n_frame x numb_dos can be prepared by model.make_stat_input

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

_compute_output_stats(all_stat, rcond=0.001, mixed_type=False)

compute_input_stats(all_stat: dict, protection: float = 0.01) → None

Compute the input statistics.

Parameters

all_stat
if numb_fparam > 0 must have all_stat['fparam'] if numb_aparam > 0 must have all_stat['aparam'] can be prepared by model.make_stat_input

protection
Divided-by-zero protection

_compute_std(sumv2, sumv, sumn)

_build_lower(start_index, natoms, inputs, fparam=None, aparam=None, bias_dos=0.0, type_suffix='', suffix='', reuse=None)

build(inputs: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, input_dict: dict | None = None, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for fitting net.

Parameters

inputs
The input descriptor

input_dict
Additional dict for inputs. if numb_fparam > 0, should have input_dict['fparam'] if numb_aparam > 0, should have input_dict['aparam']

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

ener
The system energy

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str] suffix to name scope

enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss

Get the loss function.

Parameters

loss
[dict] the loss dict

lr
[LearningRateExp] the learning rate

Returns

Loss
the loss function

classmethod deserialize(data: dict, suffix: str = '')

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

Model
The deserialized model

serialize(suffix: str = '') → dict

Serialize the model.

Returns

dict
The serialized data

class deepmd.tf.fit.EnerFitting(ntypes: int, dim_descrpt: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam: int = 0, numb_aparam: int = 0, rcond: float | None = None, tot_ener_zero: bool = False, trainable: List[bool] | None = None, seed: int | None = None, atom_ener: List[float] = [], activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, layer_name: List[str | None] | None = None, use_aparam_as_mask: bool = False, spin: deepmd.tf.utils.spin.Spin | None = None, mixed_types: bool = False, **kwargs)

Bases: *deepmd.tf.fit.fitting.Fitting*

Fitting the energy of the system. The force and the virial can also be trained.

The potential energy E is a fitting network function of the descriptor $\mathcal{D}$:

$$E(\mathcal{D}) = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \cdots \circ \mathcal{L}^{(1)} \circ \mathcal{L}^{(0)}$$

The first n hidden layers $\mathcal{L}^{(0)}, \cdots, \mathcal{L}^{(n-1)}$ are given by

$$\mathrm{y} = \mathcal{L}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \boldsymbol{\phi}(\mathrm{x}^T \mathrm{w} + \mathrm{b})$$

where $\mathrm{x} \in \mathbb{R}^{N_1}$ is the input vector and $\mathrm{y} \in \mathbb{R}^{N_2}$ is the output vector. $\mathrm{w} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathrm{b} \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable[i] is True. $\boldsymbol{\phi}$ is the activation function.

The output layer $\mathcal{L}^{(n)}$ is given by

$$\mathrm{y} = \mathcal{L}^{(n)}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \mathrm{x}^T \mathrm{w} + \mathrm{b}$$

where $\mathrm{x} \in \mathbb{R}^{N_{n-1}}$ is the input vector and $\mathrm{y} \in \mathbb{R}$ is the output scalar. $\mathrm{w} \in \mathbb{R}^{N_{n-1}}$ and $\mathrm{b} \in \mathbb{R}$ are weights and bias, respectively, both of which are trainable if trainable[n] is True.

Parameters

ntypes
: The ntypes of the descrptor $\mathcal{D}$

dim_descrpt
: The dimension of the descrptor $\mathcal{D}$

neuron
: Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

rcond
: The condition number for the regression of atomic energy.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

seed
: Random seed for initializing the network parameters.

atom_ener
: Specifying atomic energy contribution in vacuum. The set_davg_zero key in the descrptor should be set.

activation_function
: The activation function ϕ in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

layer_name
: [`list[Optional[str]]`, `optional`] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use_aparam_as_mask: bool, optional
: If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

mixed_types
: [bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

get_numb_fparam() → int

Get the number of frame parameters.

get_numb_aparam() → int

Get the number of atomic parameters.

compute_output_stats(all_stat: dict, mixed_type: bool = False) → None

Compute the ouput statistics.

Parameters

all_stat
: must have the following components: all_stat['energy'] of shape n_sys x n_batch x n_frame can be prepared by model.make_stat_input

mixed_type
: Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

_compute_output_stats(all_stat, rcond=0.001, mixed_type=False)

compute_input_stats(all_stat: dict, protection: float = 0.01) → None

Compute the input statistics.

Parameters

all_stat
: if numb_fparam > 0 must have all_stat['fparam'] if numb_aparam > 0 must have all_stat['aparam'] can be prepared by model.make_stat_input

protection
: Divided-by-zero protection

_compute_std(sumv2, sumv, sumn)

_build_lower(start_index, natoms, inputs, fparam=None, aparam=None, bias_atom_e=0.0, type_suffix='', suffix='', reuse=None)

build(inputs: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, input_dict: dict | None = None, reuse: bool | None = None, suffix: str = '') → deepmd.tf.env.tf.Tensor

Build the computational graph for fitting net.

Parameters

inputs
The input descriptor

input_dict
Additional dict for inputs. if numb_fparam > 0, should have input_dict['fparam'] if numb_aparam > 0, should have input_dict['aparam']

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

ener
The system energy

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str] suffix to name scope

change_energy_bias(data, frozen_model, origin_type_map, full_type_map, bias_adjust_mode='change-by-statistic', ntest=10) → None

enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss

Get the loss function.

Parameters

loss
[dict] The loss function parameters.

lr
[LearningRateExp] The learning rate.

Returns

Loss
The loss function.

classmethod deserialize(data: dict, suffix: str = '')

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

Model
The deserialized model

serialize(suffix: str = '') → dict

Serialize the model.

Returns

dict
The serialized data

class deepmd.tf.fit.Fitting

Bases: *deepmd.tf.utils.PluginVariant*, make_plugin_registry('fitting')

A class to remove type from input arguments.

property precision: deepmd.tf.env.tf.DType

Precision of fitting network.

abstract init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str] suffix to name scope

Notes

This method is called by others when the fitting supported initialization from the given variables.

abstract get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss

Get the loss function.

Parameters

loss
[dict] the loss dict

lr
[LearningRateExp] the learning rate

Returns

Loss
the loss function

classmethod deserialize(data: dict, suffix: str = '') → Fitting

Deserialize the fitting.

There is no suffix in a native DP model, but it is important for the TF backend.

Parameters

data
[dict] The serialized data

suffix
[str, optional] Name suffix to identify this fitting

Returns

Fitting
The deserialized fitting

abstract serialize(suffix: str = '') → dict

Serialize the fitting.

There is no suffix in a native DP model, but it is important for the TF backend.

Returns

dict
The serialized data

suffix
[str, optional] Name suffix to identify this fitting

serialize_network(ntypes: int, ndim: int, in_dim: int, neuron: List[int], activation_function: str, resnet_dt: bool, variables: dict, out_dim: int | None = 1, suffix: str = '') → dict

Serialize network.

Parameters

ntypes
[int] The number of types

ndim
[int] The dimension of elements

in_dim
[int] The input dimension

neuron
[List[int]] The neuron list

activation_function
[str] The activation function

resnet_dt
[bool] Whether to use resnet

variables
[dict] The input variables

suffix
[str, optional] The suffix of the scope

out_dim
[int, optional] The output dimension

Returns

dict
The converted network data

classmethod deserialize_network(data: dict, suffix: str = '') → dict

Deserialize network.

Parameters

data
[dict] The input network data

suffix
[str, optional] The suffix of the scope

Returns

variables
[dict] The input variables

class deepmd.tf.fit.GlobalPolarFittingSeA(descrpt: deepmd.tf.env.tf.Tensor, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, sel_type: List[int] | None = None, fit_diag: bool = True, scale: List[float] | None = None, diag_shift: List[float] | None = None, seed: int | None = None, activation_function: str = 'tanh', precision: str = 'default')

Fit the system polarizability with descriptor se_a.

Parameters

descrpt
[tf.Tensor] The descrptor

neuron
[List[int]] Number of neurons in each hidden layer of the fitting net

resnet_dt
[bool] Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

sel_type
[List[int]] The atom types selected to have an atomic polarizability prediction

fit_diag
: [bool] Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.

scale
: [List[float]] The output of the fitting net (polarizability matrix) for type i atom will be scaled by scale[i]

diag_shift
: [List[float]] The diagonal part of the polarizability matrix of type i will be shifted by diag_shift[i]. The shift operation is carried out after scale.

seed
: [int] Random seed for initializing the network parameters.

activation_function
: [str] The activation function in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: [str] The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

get_sel_type() → int

Get selected atom types.

get_out_size() → int

Get the output size. Should be 9.

build(input_d, rot_mat, natoms, input_dict: dict | None = None, reuse=None, suffix='') → deepmd.tf.env.tf.Tensor

Build the computational graph for fitting net.

Parameters

input_d
: The input descriptor

rot_mat
: The rotation matrix from the descriptor.

natoms
: The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

input_dict
: Additional dict for inputs.

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

polar
: The system polarizability

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

suffix
[str] suffix to name scope

enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss

Get the loss function.

Parameters

loss
[dict] the loss dict

lr
[LearningRateExp] the learning rate

Returns

Loss
the loss function

class deepmd.tf.fit.PolarFittingSeA(ntypes: int, dim_descrpt: int, embedding_width: int, neuron: List[int] = [120, 120, 120], resnet_dt: bool = True, sel_type: List[int] | None = None, fit_diag: bool = True, scale: List[float] | None = None, shift_diag: bool = True, seed: int | None = None, activation_function: str = 'tanh', precision: str = 'default', uniform_seed: bool = False, mixed_types: bool = False, **kwargs)

Bases: *deepmd.tf.fit.fitting.Fitting*

Fit the atomic polarizability with descriptor se_a.

Parameters

ntypes
The ntypes of the descrptor $\mathcal{D}$

dim_descrpt
The dimension of the descrptor $\mathcal{D}$

embedding_width
The rotation matrix dimension of the descrptor $\mathcal{D}$

neuron
[List[int]] Number of neurons in each hidden layer of the fitting net

resnet_dt
: [bool] Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

sel_type
: [List[int]] The atom types selected to have an atomic polarizability prediction. If is None, all atoms are selected.

fit_diag
: [bool] Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.

scale
: [List[float]] The output of the fitting net (polarizability matrix) for type i atom will be scaled by scale[i]

diag_shift
: [List[float]] The diagonal part of the polarizability matrix of type i will be shifted by diag_shift[i]. The shift operation is carried out after scale.

seed
: [int] Random seed for initializing the network parameters.

activation_function
: [str] The activation function in the embedding net. Supported options are "tanh", "gelu", "none", "linear", "sigmoid", "softplus", "relu", "relu6", "gelu_tf".

precision
: [str] The precision of the embedding net parameters. Supported options are "default", "float64", "float16", "float32".

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

mixed_types
: [bool] If true, use a uniform fitting net for all atom types, otherwise use different fitting nets for different atom types.

get_sel_type() → List[int]

Get selected atom types.

get_out_size() → int

Get the output size. Should be 9.

compute_output_stats(all_stat)

Compute the output statistics.

Parameters

all_stat
: Dictionary of inputs. can be prepared by model.make_stat_input

_build_lower(start_index, natoms, inputs, rot_mat, suffix='', reuse=None)

build(input_d: deepmd.tf.env.tf.Tensor, rot_mat: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, input_dict: dict | None = None, reuse: bool | None = None, suffix: str = '')

Build the computational graph for fitting net.

Parameters

input_d
: The input descriptor

rot_mat
: The rotation matrix from the descriptor.

natoms
: The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

input_dict
: Additional dict for inputs.

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

atomic_polar
: The atomic polarizability

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

graph
: [tf.Graph] The input frozen model graph

graph_def
: [tf.GraphDef] The input frozen model graph_def

suffix
: [str] suffix to name scope

enable_mixed_precision(mixed_prec: dict | None = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
: The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss

Get the loss function.

serialize(suffix: str) → dict

Serialize the model.

Returns

dict
: The serialized data

classmethod deserialize(data: dict, suffix: str)

Deserialize the model.

Parameters

data
[dict] The serialized data

Returns

Model
The deserialized model

deepmd.tf.infer

Submodule containing all the implemented potentials.

Submodules

deepmd.tf.infer.data_modifier

Module Contents

Classes

DipoleChargeModifier	Parameters

class deepmd.tf.infer.data_modifier.DipoleChargeModifier(model_name: str, model_charge_map: List[float], sys_charge_map: List[float], ewald_h: float = 1, ewald_beta: float = 1)

Bases: **deepmd.tf.infer.deep_dipole.DeepDipoleOld**

Parameters

model_name
The model file for the DeepDipole model

model_charge_map
Gives the amount of charge for the wfcc

sys_charge_map
Gives the amount of charge for the real atoms

ewald_h
Grid spacing of the reciprocal part of Ewald sum. Unit: A

ewald_beta
Splitting parameter of the Ewald sum. Unit: A^{-1}

build_fv_graph() → deepmd.tf.env.tf.Tensor
Build the computational graph for the force and virial inference.

_build_fv_graph_inner()

_enrich(dipole, dof=3)

_slice_descrpt_deriv(deriv)

eval(coord: numpy.ndarray, box: numpy.ndarray, atype: numpy.ndarray, eval_fv: bool = True) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Evaluate the modification.

Parameters

coord
The coordinates of atoms

box
The simulation region. PBC is assumed

atype
The atom types

eval_fv
Evaluate force and virial

Returns

tot_e
The energy modification

tot_f
The force modification

tot_v
The virial modification

_eval_fv(coords, cells, atom_types, ext_f)

_extend_system(coord, box, atype, charge)

modify_data(data: dict, data_sys: deepmd.tf.utils.data.DeepmdData) → None

Modify data.

Parameters

data
Internal data of DeepmdData. Be a dict, has the following keys - coord coordinates - box simulation box - type atom types - find_energy tells if data has energy - find_force tells if data has force - find_virial tells if data has virial - energy energy - force force - virial virial

data_sys
[**DeepmdData**] The data system.

deepmd.tf.infer.deep_dipole

Module Contents

Classes

DeepDipole	Deep dipole model.

class deepmd.tf.infer.deep_dipole.DeepDipole(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_tensor.DeepTensor*

Deep dipole model.

Parameters

model_file
[Path] The name of the frozen model file.

*args
[list] Positional arguments.

auto_batch_size
[bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

property output_tensor_name: str

The name of the tensor.

deepmd.tf.infer.deep_dos

Module Contents

Classes

DeepDOS	Deep density of states model.

class deepmd.tf.infer.deep_dos.DeepDOS(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_eval.DeepEval*

Deep density of states model.

Parameters

model_file
[Path] The name of the frozen model file.

*args
[list] Positional arguments.

auto_batch_size
: [bool or `int` or `AutoBatchSize`, default: `True`] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
: [`ase.neighborlist.NewPrimitiveNeighborList`, `optional`] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
: [`dict`] Keyword arguments.

property output_def: *deepmd.dpmodel.output_def.ModelOutputDef*

Get the output definition of this model.

property numb_dos: int

Get the number of DOS.

eval(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: List[int] | numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: Dict[str, Any]) → Tuple[numpy.ndarray, Ellipsis]

Evaluate energy, force, and virial. If atomic is True, also return atomic energy and atomic virial.

Parameters

coords
: [`np.ndarray`] The coordinates of the atoms, in shape (nframes, natoms, 3).

cells
: [`np.ndarray`] The cell vectors of the system, in shape (nframes, 9). If the system is not periodic, set it to None.

atom_types
: [`List[int]` or `np.ndarray`] The types of the atoms. If mixed_type is False, the shape is (natoms,); otherwise, the shape is (nframes, natoms).

atomic
: [bool, `optional`] Whether to return atomic energy and atomic virial, by default False.

fparam
: [`np.ndarray`, `optional`] The frame parameters, by default None.

aparam
: [`np.ndarray`, `optional`] The atomic parameters, by default None.

mixed_type
: [bool, `optional`] Whether the atom_types is mixed type, by default False.

**kwargs
: [`Dict[str, Any]`] Keyword arguments.

Returns

energy
: The energy of the system, in shape (nframes,).

force
: The force of the system, in shape (nframes, natoms, 3).

virial
: The virial of the system, in shape (nframes, 9).

atomic_energy
The atomic energy of the system, in shape (nframes, natoms). Only returned when atomic is True.

atomic_virial
The atomic virial of the system, in shape (nframes, natoms, 9). Only returned when atomic is True.

get_numb_dos() → int

deepmd.tf.infer.deep_eval

Module Contents

Classes

DeepEval	TensorFlow backend implementation for DeepEval.
DeepEvalOld	Common methods for DeepPot, DeepWFC, DeepPolar, ...

class deepmd.tf.infer.deep_eval.DeepEval(model_file: pathlib.Path, output_def: deepmd.dpmodel.output_def.ModelOutputDef, *args: list, load_prefix: str = 'load', default_tf_graph: bool = False, auto_batch_size: bool | int | deepmd.tf.utils.batch_size.AutoBatchSize = False, input_map: dict | None = None, neighbor_list=None, **kwargs: dict)

Bases: *deepmd.infer.deep_eval.DeepEvalBackend*

TensorFlow backend implementation for DeepEval.

Parameters

model_file
[Path] The name of the frozen model file.

output_def
[ModelOutputDef] The output definition of the model.

*args
[list] Positional arguments.

load_prefix: str
The prefix in the load computational graph

default_tf_graph
[bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

auto_batch_size
[bool or int or AutomaticBatchSize, default: False] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

input_map
[dict, optional] The input map for tf.import_graph_def. Only work with default tf graph

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

property model_type: *deepmd.infer.deep_eval.DeepEval*

Get type of model.

:type:str

property model_version: str

Get version of model.

Returns

str
version of model

property sess: deepmd.tf.env.tf.Session

Get TF session.

_init_tensors()

_init_attr()

_graph_compatable() → bool

Check the model compatability.

Returns

bool
If the model stored in the graph file is compatable with the current code

_get_tensor(tensor_name: str) → deepmd.tf.env.tf.Tensor

Get TF graph tensor.

Parameters

tensor_name
[str] name of tensor to get

Returns

tf.Tensor
loaded tensor

static _load_graph(frozen_graph_filename: pathlib.Path, prefix: str = 'load', default_tf_graph: bool = False, input_map: dict | None = None)

static sort_input(coord: numpy.ndarray, atom_type: numpy.ndarray, sel_atoms: List[int] | None = None)

Sort atoms in the system according their types.

Parameters

coord
The coordinates of atoms. Should be of shape [nframes, natoms, 3]

atom_type
The type of atoms Should be of shape [natoms]

sel_atoms
The selected atoms by type

Returns

coord_out
The coordinates after sorting

atom_type_out
The atom types after sorting

idx_map
The index mapping from the input to the output. For example coord_out = coord[:,idx_map,:]

sel_atom_type
Only output if sel_atoms is not None The sorted selected atom types

sel_idx_map
Only output if sel_atoms is not None The index mapping from the selected atoms to sorted selected atoms.

static reverse_map(vec: numpy.ndarray, imap: List[int]) → numpy.ndarray

Reverse mapping of a vector according to the index map.

Parameters

vec
Input vector. Be of shape [nframes, natoms, -1]

imap
Index map. Be of shape [natoms]

Returns

vec_out
Reverse mapped vector.

make_natoms_vec(atom_types: numpy.ndarray) → numpy.ndarray

Make the natom vector used by deepmd-kit.

Parameters

atom_types
The type of atoms

Returns

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

eval_typeebd() → numpy.ndarray

Evaluate output of type embedding network by using this model.

Returns

np.ndarray
The output of type embedding network. The shape is [ntypes, o_size], where ntypes is the number of types, and o_size is the number of nodes in the output layer.

Raises

KeyError
If the model does not enable type embedding.

See also:

deepmd.tf.utils.type_embed.TypeEmbedNet
The type embedding network.

Examples

Get the output of type embedding network of graph.pb:

```
>>> from deepmd.tf.infer import DeepPotential
>>> dp = DeepPotential("graph.pb")
>>> dp.eval_typeebd()
```

build_neighbor_list(coords: numpy.ndarray, cell: numpy.ndarray | None, atype: numpy.ndarray, imap: numpy.ndarray, neighbor_list)

Make the mesh with neighbor list for a single frame.

Parameters

coords
[np.ndarray] The coordinates of atoms. Should be of shape [natoms, 3]

cell
[Optional[np.ndarray]] The cell of the system. Should be of shape [3, 3]

atype
[np.ndarray] The type of atoms. Should be of shape [natoms]

imap
[np.ndarray] The index map of atoms. Should be of shape [natoms]

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList] ASE neighbor list. The following method or attribute will be used/set: bothways, self_interaction, update, build, first_neigh, pair_second, offset_vec.

Returns

natoms_vec
[np.ndarray] The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: nloc natoms[1]: nall natoms[i]: 2 <= i < Ntypes+2, number of type i atoms for nloc

coords
[np.ndarray] The coordinates of atoms, including ghost atoms. Should be of shape [nframes, nall, 3]

atype
[np.ndarray] The type of atoms, including ghost atoms. Should be of shape [nall]

mesh
[np.ndarray] The mesh in nei_mode=4.

imap
[np.ndarray] The index map of atoms. Should be of shape [nall]

ghost_map
[np.ndarray] The index map of ghost atoms. Should be of shape [nghost]

get_ntypes() → int

Get the number of atom types of this model.

get_ntypes_spin() → int

Get the number of spin atom types of this model.

get_rcut() → float

Get the cut-off radius of this model.

get_type_map() → List[str]

Get the type map (element name of the atom types) of this model.

get_sel_type() → numpy.ndarray | None

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this DP.

get_dim_aparam() → int

Get the number (dimension) of atomic parameters of this DP.

_eval_func(inner_func: Callable, numb_test: int, natoms: int) → Callable

Wrapper method with auto batch size.

Parameters

inner_func
[Callable] the method to be wrapped

numb_test
[int] number of tests

natoms
[int] number of atoms

Returns

Callable
the wrapper

_get_natoms_and_nframes(coords: numpy.ndarray, atom_types: List[int] | numpy.ndarray) → Tuple[int, int]

eval(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, efield: numpy.ndarray | None = None, **kwargs: Dict[str, Any]) → Dict[str, numpy.ndarray]

Evaluate the energy, force and virial by using this DP.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

atomic
Calculate the atomic energy and virial

fparam
The frame parameter. The array can be of size : - nframes x dim_fparam. - dim_fparam. Then all frames are assumed to be provided with the same fparam.

aparam
The atomic parameter The array can be of size : - nframes x natoms x dim_aparam. - natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. - dim_aparam. Then all frames and atoms are provided with the same aparam.

efield
The external field on atoms. The array should be of size nframes x natoms x 3

**kwargs
Other parameters

Returns

output_dict
[dict] The output of the evaluation. The keys are the names of the output variables, and the values are the corresponding output arrays.

_prepare_feed_dict(coords, cells, atom_types, fparam=None, aparam=None, efield=None)

_eval_inner(coords, cells, atom_types, fparam=None, aparam=None, efield=None, **kwargs)

_get_output_shape(odef, nframes, natoms)

eval_descriptor(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: numpy.ndarray, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, efield: numpy.ndarray | None = None) → numpy.ndarray

Evaluate descriptors by using this DP.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

fparam
The frame parameter. The array can be of size : - nframes x dim_fparam. - dim_fparam. Then all frames are assumed to be provided with the same fparam.

aparam
The atomic parameter The array can be of size : - nframes x natoms x dim_aparam. - natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. - dim_aparam. Then all frames and atoms are provided with the same aparam.

efield
The external field on atoms. The array should be of size nframes x natoms x 3

Returns

descriptor
Descriptors.

_eval_descriptor_inner(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: numpy.ndarray, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, efield: numpy.ndarray | None = None) → numpy.ndarray

get_numb_dos() → int
Get the number of DOS.

get_has_efield() → bool
Check if the model has efield.

class deepmd.tf.infer.deep_eval.DeepEvalOld(model_file: pathlib.Path, load_prefix: str = 'load', default_tf_graph: bool = False, auto_batch_size: bool | int | deepmd.tf.utils.batch_size.AutoBatchSize = False, input_map: dict | None = None, neighbor_list=None)

Common methods for DeepPot, DeepWFC, DeepPolar, ...

Parameters

model_file
[Path] The name of the frozen model file.

load_prefix: str
The prefix in the load computational graph

default_tf_graph
[bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

auto_batch_size
[bool or int or AutomaticBatchSize, default: False] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

input_map
[dict, optional] The input map for tf.import_graph_def. Only work with default tf graph

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

property model_type: str
Get type of model.

:type:str

property model_version: str
Get version of model.

Returns

str
version of model

property sess: deepmd.tf.env.tf.Session

Get TF session.

load_prefix: str

_graph_compatable() → bool

Check the model compatability.

Returns

bool
If the model stored in the graph file is compatable with the current code

_get_tensor(tensor_name: str, attr_name: str | None = None) → deepmd.tf.env.tf.Tensor

Get TF graph tensor and assign it to class namespace.

Parameters

tensor_name
[str] name of tensor to get

attr_name
[Optional[str], optional] if specified, class attribute with this name will be created and tensor will be assigned to it, by default None

Returns

tf.Tensor
loaded tensor

static _load_graph(frozen_graph_filename: pathlib.Path, prefix: str = 'load', default_tf_graph: bool = False, input_map: dict | None = None)

static sort_input(coord: numpy.ndarray, atom_type: numpy.ndarray, sel_atoms: List[int] | None = None, mixed_type: bool = False)

Sort atoms in the system according their types.

Parameters

coord
The coordinates of atoms. Should be of shape [nframes, natoms, 3]

atom_type
The type of atoms Should be of shape [natoms]

sel_atoms
The selected atoms by type

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

coord_out
The coordinates after sorting

atom_type_out
The atom types after sorting

idx_map
The index mapping from the input to the output. For example coord_out = coord[:,idx_map,:]

sel_atom_type
Only output if sel_atoms is not None The sorted selected atom types

sel_idx_map
Only output if sel_atoms is not None The index mapping from the selected atoms to sorted selected atoms.

static reverse_map(vec: numpy.ndarray, imap: List[int]) → numpy.ndarray

Reverse mapping of a vector according to the index map.

Parameters

vec
Input vector. Be of shape [nframes, natoms, -1]

imap
Index map. Be of shape [natoms]

Returns

vec_out
Reverse mapped vector.

make_natoms_vec(atom_types: numpy.ndarray, mixed_type: bool = False) → numpy.ndarray

Make the natom vector used by deepmd-kit.

Parameters

atom_types
The type of atoms

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

eval_typeebd() → numpy.ndarray

Evaluate output of type embedding network by using this model.

Returns

np.ndarray
The output of type embedding network. The shape is [ntypes, o_size], where ntypes is the number of types, and o_size is the number of nodes in the output layer.

Raises

KeyError
If the model does not enable type embedding.

See also:

deepmd.tf.utils.type_embed.TypeEmbedNet
The type embedding network.

Examples

Get the output of type embedding network of graph.pb:

```
>>> from deepmd.tf.infer import DeepPotential
>>> dp = DeepPotential("graph.pb")
>>> dp.eval_typeebd()
```

build_neighbor_list(coords: numpy.ndarray, cell: numpy.ndarray | None, atype: numpy.ndarray, imap: numpy.ndarray, neighbor_list)

Make the mesh with neighbor list for a single frame.

Parameters

coords
[np.ndarray] The coordinates of atoms. Should be of shape [natoms, 3]

cell
[Optional[np.ndarray]] The cell of the system. Should be of shape [3, 3]

atype
[np.ndarray] The type of atoms. Should be of shape [natoms]

imap
[np.ndarray] The index map of atoms. Should be of shape [natoms]

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList] ASE neighbor list. The following method or attribute will be used/set: bothways, self_interaction, update, build, first_neigh, pair_second, offset_vec.

Returns

natoms_vec
[np.ndarray] The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: nloc natoms[1]: nall natoms[i]: 2 <= i < Ntypes+2, number of type i atoms for nloc

coords
[np.ndarray] The coordinates of atoms, including ghost atoms. Should be of shape [nframes, nall, 3]

atype
[np.ndarray] The type of atoms, including ghost atoms. Should be of shape [nall]

mesh
[np.ndarray] The mesh in nei_mode=4.

imap
[np.ndarray] The index map of atoms. Should be of shape [nall]

ghost_map
[np.ndarray] The index map of ghost atoms. Should be of shape [nghost]

deepmd.tf.infer.deep_polar

Module Contents

Classes

DeepGlobalPolar	Deep Tensor Model.
DeepPolar	Deep polar model.

class deepmd.tf.infer.deep_polar.DeepGlobalPolar(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_tensor.DeepTensor*

Deep Tensor Model.

Parameters

model_file
[Path] The name of the frozen model file.

*args
[list] Positional arguments.

auto_batch_size
[bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

property output_tensor_name: str

The name of the tensor.

eval(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: List[int] | numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: dict) → numpy.ndarray

Evaluate the model.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
[list[int] or np.ndarray] The atom types The list should contain natoms ints

atomic
If True (default), return the atomic tensor Otherwise return the global tensor

fparam
Not used in this model

aparam
Not used in this model

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

tensor
The returned tensor If atomic == False then of size nframes x output_dim else of size nframes x natoms x output_dim

class deepmd.tf.infer.deep_polar.DeepPolar(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_tensor.DeepTensor*

Deep polar model.

Parameters

model_file
[Path] The name of the frozen model file.

*args
[list] Positional arguments.

auto_batch_size
[bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

property output_tensor_name: str

The name of the tensor.

deepmd.tf.infer.deep_pot

Module Contents

Classes

DeepPot	Potential energy model.

class deepmd.tf.infer.deep_pot.DeepPot(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_eval.DeepEval*

Potential energy model.

Parameters

model_file
: [Path] The name of the frozen model file.

*args
: [list] Positional arguments.

auto_batch_size
: [bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
: [ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
: [dict] Keyword arguments.

Examples

```
>>> from deepmd.infer import DeepPot
>>> import numpy as np
>>> dp = DeepPot("graph.pb")
>>> coord = np.array([[1, 0, 0], [0, 0, 1.5], [1, 0, 3]]).reshape([1, -1])
>>> cell = np.diag(10 * np.ones(3)).reshape([1, -1])
>>> atype = [1, 0, 1]
>>> e, f, v = dp.eval(coord, cell, atype)
```

where e, f and v are predicted energy, force and virial of the system, respectively.

property output_def: *deepmd.dpmodel.output_def.ModelOutputDef*
: Get the output definition of this model.

property output_def_mag: *deepmd.dpmodel.output_def.ModelOutputDef*
: Get the output definition of this model with magnetic parts.

eval(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: List[int] | numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: Dict[str, Any]) → Tuple[numpy.ndarray, Ellipsis]

Evaluate energy, force, and virial. If atomic is True, also return atomic energy and atomic virial.

Parameters

coords
[np.ndarray] The coordinates of the atoms, in shape (nframes, natoms, 3).

cells
[np.ndarray] The cell vectors of the system, in shape (nframes, 9). If the system is not periodic, set it to None.

atom_types
[List[int] or np.ndarray] The types of the atoms. If mixed_type is False, the shape is (natoms,); otherwise, the shape is (nframes, natoms).

atomic
[bool, optional] Whether to return atomic energy and atomic virial, by default False.

fparam
[np.ndarray, optional] The frame parameters, by default None.

aparam
[np.ndarray, optional] The atomic parameters, by default None.

mixed_type
[bool, optional] Whether the atom_types is mixed type, by default False.

**kwargs
[Dict[str, *Any*]] Keyword arguments.

Returns

energy
The energy of the system, in shape (nframes,).

force
The force of the system, in shape (nframes, natoms, 3).

virial
The virial of the system, in shape (nframes, 9).

atomic_energy
The atomic energy of the system, in shape (nframes, natoms). Only returned when atomic is True.

atomic_virial
The atomic virial of the system, in shape (nframes, natoms, 9). Only returned when atomic is True.

deepmd.tf.infer.deep_tensor

Module Contents

Classes

DeepTensor	Evaluates a tensor model.

class deepmd.tf.infer.deep_tensor.DeepTensor(model_file: pathlib.Path, load_prefix: str = 'load', default_tf_graph: bool = False, input_map: dict | None = None, neighbor_list=None)

Bases: *deepmd.tf.infer.deep_eval.DeepEvalOld*

Evaluates a tensor model.

Parameters

model_file: str
The name of the frozen model file.

load_prefix: str
The prefix in the load computational graph

default_tf_graph
[bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

input_map
[dict, optional] The input map for tf.import_graph_def. Only work with default tf graph

neighbor_list
[ase.neighborlist.NeighborList, optional] The neighbor list object. If None, then build the native neighbor list.

tensors: ClassVar[Dict[str, str]]

_run_default_sess()

get_ntypes() → int
Get the number of atom types of this model.

get_rcut() → float
Get the cut-off radius of this model.

get_type_map() → List[str]
Get the type map (element name of the atom types) of this model.

get_sel_type() → List[int]
Get the selected atom types of this model.

get_dim_fparam() → int
Get the number (dimension) of frame parameters of this DP.

get_dim_aparam() → int
Get the number (dimension) of atomic parameters of this DP.

eval(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: List[int], atomic: bool = True, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, efield: numpy.ndarray | None = None, mixed_type: bool = False) → numpy.ndarray

Evaluate the model.

Parameters

coords
: The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
: The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
: The atom types The list should contain natoms ints

atomic
: If True (default), return the atomic tensor Otherwise return the global tensor

fparam
: Not used in this model

aparam
: Not used in this model

efield
: Not used in this model

mixed_type
: Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

tensor
: The returned tensor If atomic == False then of size nframes x output_dim else of size nframes x natoms x output_dim

eval_full(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: List[int], atomic: bool = False, fparam: numpy.array | None = None, aparam: numpy.array | None = None, efield: numpy.array | None = None, mixed_type: bool = False) → Tuple[numpy.ndarray, Ellipsis]

Evaluate the model with interface similar to the energy model. Will return global tensor, component-wise force and virial and optionally atomic tensor and atomic virial.

Parameters

coords
: The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
: The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
: The atom types The list should contain natoms ints

atomic
: Whether to calculate atomic tensor and virial

fparam
Not used in this model

aparam
Not used in this model

efield
Not used in this model

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

tensor
The global tensor. shape: [nframes x nout]

force
The component-wise force (negative derivative) on each atom. shape: [nframes x nout x natoms x 3]

virial
The component-wise virial of the tensor. shape: [nframes x nout x 9]

atom_tensor
The atomic tensor. Only returned when atomic == True shape: [nframes x natoms x nout]

atom_virial
The atomic virial. Only returned when atomic == True shape: [nframes x nout x natoms x 9]

deepmd.tf.infer.deep_wfc

Module Contents

Classes

DeepWFC	Deep WFC model.

class deepmd.tf.infer.deep_wfc.**DeepWFC**(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_tensor.DeepTensor*

Deep WFC model.

Parameters

model_file
[Path] The name of the frozen model file.

*args
[list] Positional arguments.

auto_batch_size
: [bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
: [ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
: [dict] Keyword arguments.

property output_tensor_name: str

The name of the tensor.

deepmd.tf.infer.ewald_recp

Module Contents

Classes

EwaldRecp	Evaluate the reciprocal part of the Ewald sum.

class deepmd.tf.infer.ewald_recp.EwaldRecp(hh, beta)

Evaluate the reciprocal part of the Ewald sum.

eval(coord: numpy.ndarray, charge: numpy.ndarray, box: numpy.ndarray) →
Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Evaluate.

Parameters

coord
: The coordinates of atoms

charge
: The atomic charge

box
: The simulation region. PBC is assumed

Returns

e
: The energy

f
: The force

v
: The virial

deepmd.tf.infer.model_devi

Module Contents

Functions

calc_model_devi(coord, box, atype, models[, fname, ...])	Python interface to calculate model deviation.
calc_model_devi_e(→ numpy.ndarray)	Calculate model deviation of total energy per atom.
calc_model_devi_f(...)	Calculate model deviation of force.
calc_model_devi_v(→ Tuple[numpy.ndarray, ...)	Calculate model deviation of virial.
make_model_devi(*, models, system, output, frequency)	Make model deviation calculation.
write_model_devi_out(devi, fname[, header, atomic])	Write output of model deviation.

deepmd.tf.infer.model_devi.calc_model_devi(coord, box, atype, models, fname=None, frequency=1, mixed_type=False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, real_data: dict | None = None, atomic: bool = False, relative: float | None = None, relative_v: float | None = None)

Python interface to calculate model deviation.

Parameters

coord
: [numpy.ndarray, n_frames x n_atoms x 3] Coordinates of system to calculate

box
: [numpy.ndarray or None, n_frames x 3 x 3] Box to specify periodic boundary condition. If None, no pbc will be used

atype
: [numpy.ndarray, n_atoms x 1] Atom types

models
: [list of DeepPot models] Models used to evaluate deviation

fname
: [str or None] File to dump results, default None

frequency
: [int] Steps between frames (if the system is given by molecular dynamics engine), default 1

mixed_type
: [bool] Whether the input atype is in mixed_type format or not

fparam
: [numpy.ndarray] frame specific parameters

aparam
: [numpy.ndarray] atomic specific parameters

real_data
: [dict, optional] real data to calculate RMS real error

atomic
: [bool, default: `False`] If True, calculate the force model deviation of each atom.

relative
: [`float`, default: `None`] If given, calculate the relative model deviation of force. The value is the level parameter for computing the relative model deviation of the force.

relative_v
: [`float`, default: `None`] If given, calculate the relative model deviation of virial. The value is the level parameter for computing the relative model deviation of the virial.

Returns

model_devi
: [`numpy.ndarray`, n_frames x 8] Model deviation results. The first column is index of steps, the other 7 columns are max_devi_v, min_devi_v, avg_devi_v, max_devi_f, min_devi_f, avg_devi_f, devi_e.

Examples

```
>>> from deepmd.tf.infer import calc_model_devi
>>> from deepmd.tf.infer import DeepPot as DP
>>> import numpy as np
>>> coord = np.array([[1, 0, 0], [0, 0, 1.5], [1, 0, 3]]).reshape([1, -1])
>>> cell = np.diag(10 * np.ones(3)).reshape([1, -1])
>>> atype = [1, 0, 1]
>>> graphs = [DP("graph.000.pb"), DP("graph.001.pb")]
>>> model_devi = calc_model_devi(coord, cell, atype, graphs)
```

deepmd.tf.infer.model_devi.**calc_model_devi_e**(es: numpy.ndarray, real_e: numpy.ndarray | None = None) → numpy.ndarray

Calculate model deviation of total energy per atom.

Here we don't use the atomic energy, as the decomposition of energy is arbitrary and not unique. There is no fitting target for atomic energy.

Parameters

es
: [`numpy.ndarray`] size of `n_models x n_frames x 1

real_e
: [`numpy.ndarray`] real energy, size of n_frames x 1. If given, the RMS real error is calculated instead.

Returns

max_devi_e
: [`numpy.ndarray`] maximum deviation of energy

deepmd.tf.infer.model_devi.**calc_model_devi_f**(fs: numpy.ndarray, real_f: numpy.ndarray | None = None, relative: float | None = None, atomic: Literal[False] = False) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

deepmd.tf.infer.model_devi.**calc_model_devi_f**(fs: numpy.ndarray, real_f: numpy.ndarray | None = None, relative: float | None = None, *, atomic: Literal[True]) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray]

Calculate model deviation of force.

Parameters

fs
[numpy.ndarray] size of n_models x n_frames x n_atoms x 3

real_f
[numpy.ndarray or None] real force, size of n_frames x n_atoms x 3. If given, the RMS real error is calculated instead.

relative
[float, default: None] If given, calculate the relative model deviation of force. The value is the level parameter for computing the relative model deviation of the force.

atomic
[bool, default: False] Whether return deviation of force in all atoms

Returns

max_devi_f
[numpy.ndarray] maximum deviation of force in all atoms

min_devi_f
[numpy.ndarray] minimum deviation of force in all atoms

avg_devi_f
[numpy.ndarray] average deviation of force in all atoms

fs_devi
[numpy.ndarray] deviation of force in all atoms, returned if atomic=True

deepmd.tf.infer.model_devi.calc_model_devi_v(vs: numpy.ndarray, real_v: numpy.ndarray | None = None, relative: float | None = None) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Calculate model deviation of virial.

Parameters

vs
[numpy.ndarray] size of n_models x n_frames x 9

real_v
[numpy.ndarray] real virial, size of n_frames x 9. If given, the RMS real error is calculated instead.

relative
[float, default: None] If given, calculate the relative model deviation of virial. The value is the level parameter for computing the relative model deviation of the virial.

Returns

max_devi_v
[numpy.ndarray] maximum deviation of virial in 9 elements

min_devi_v
[numpy.ndarray] minimum deviation of virial in 9 elements

avg_devi_v
[numpy.ndarray] average deviation of virial in 9 elements

deepmd.tf.infer.model_devi.**make_model_devi**(*, models: list, system: str, output: str, frequency: int, real_error: bool = False, atomic: bool = False, relative: float | None = None, relative_v: float | None = None, **kwargs)

Make model deviation calculation.

Parameters

models
[list] A list of paths of models to use for making model deviation

system
[str] The path of system to make model deviation calculation

output
[str] The output file for model deviation results

frequency
[int] The number of steps that elapse between writing coordinates in a trajectory by a MD engine (such as Gromacs / Lammps). This paramter is used to determine the index in the output file.

real_error
[bool, default: False] If True, calculate the RMS real error instead of model deviation.

atomic
[bool, default: False] If True, calculate the force model deviation of each atom.

relative
[float, default: None] If given, calculate the relative model deviation of force. The value is the level parameter for computing the relative model deviation of the force.

relative_v
[float, default: None] If given, calculate the relative model deviation of virial. The value is the level parameter for computing the relative model deviation of the virial.

**kwargs
Arbitrary keyword arguments.

deepmd.tf.infer.model_devi.**write_model_devi_out**(devi: numpy.ndarray, fname: str, header: str = '', atomic: bool = False)

Write output of model deviation.

Parameters

devi
[numpy.ndarray] the first column is the steps index

fname
[str] the file name to dump

header
[str, default=""] the header to dump

atomic
[bool, default: False] whether atomic model deviation is printed

Package Contents

Classes

DipoleChargeModifier	Parameters
DeepDipole	Deep dipole model.
DeepDOS	Deep density of states model.
DeepEval	TensorFlow backend implementation for DeepEval.
DeepGlobalPolar	Deep Tensor Model.
DeepPolar	Deep polar model.
DeepPot	Potential energy model.
DeepWFC	Deep WFC model.
EwaldRecp	Evaluate the reciprocal part of the Ewald sum.

Functions

DeepPotential(→ deep_eval.DeepEval)	Factory function that forwards to DeepEval (for compatbility).
calc_model_devi(coord, box, atype, models[, fname, ...])	Python interface to calculate model deviation.

`deepmd.tf.infer.DeepPotential`(*args, **kwargs) → deep_eval.DeepEval

Factory function that forwards to DeepEval (for compatbility).

Parameters

*args
positional arguments

**kwargs
keyword arguments

Returns

DeepEval
potentials

`class deepmd.tf.infer.DipoleChargeModifier`(model_name: str, model_charge_map: List[float], sys_charge_map: List[float], ewald_h: float = 1, ewald_beta: float = 1)

Bases: `deepmd.tf.infer.deep_dipole.DeepDipoleOld`

Parameters

model_name
The model file for the DeepDipole model

model_charge_map
Gives the amount of charge for the wfcc

sys_charge_map
: Gives the amount of charge for the real atoms

ewald_h
: Grid spacing of the reciprocal part of Ewald sum. Unit: A

ewald_beta
: Splitting parameter of the Ewald sum. Unit: A^{-1}

build_fv_graph() → deepmd.tf.env.tf.Tensor

Build the computational graph for the force and virial inference.

_build_fv_graph_inner()

_enrich(dipole, dof=3)

_slice_descrpt_deriv(deriv)

eval(coord: numpy.ndarray, box: numpy.ndarray, atype: numpy.ndarray, eval_fv: bool = True) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Evaluate the modification.

Parameters

coord
: The coordinates of atoms

box
: The simulation region. PBC is assumed

atype
: The atom types

eval_fv
: Evaluate force and virial

Returns

tot_e
: The energy modification

tot_f
: The force modification

tot_v
: The virial modification

_eval_fv(coords, cells, atom_types, ext_f)

_extend_system(coord, box, atype, charge)

modify_data(data: dict, data_sys: deepmd.tf.utils.data.DeepmdData) → None

Modify data.

Parameters

data
: Internal data of DeepmdData. Be a dict, has the following keys - coord coordinates - box simulation box - type atom types - find_energy tells if data has energy - find_force tells if data has force - find_virial tells if data has virial - energy energy - force force - virial virial

data_sys
[DeepmdData] The data system.

class deepmd.tf.infer.DeepDipole(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_tensor.DeepTensor*

Deep dipole model.

Parameters

model_file
[Path] The name of the frozen model file.

*args
[list] Positional arguments.

auto_batch_size
[bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

property output_tensor_name: str

The name of the tensor.

class deepmd.tf.infer.DeepDOS(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_eval.DeepEval*

Deep density of states model.

Parameters

model_file
[Path] The name of the frozen model file.

*args
[list] Positional arguments.

auto_batch_size
[bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

property output_def: *deepmd.dpmodel.output_def.ModelOutputDef*

Get the output definition of this model.

property numb_dos: int

Get the number of DOS.

eval(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: List[int] | numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: Dict[str, Any]) → Tuple[numpy.ndarray, Ellipsis]

Evaluate energy, force, and virial. If atomic is True, also return atomic energy and atomic virial.

Parameters

coords
[np.ndarray] The coordinates of the atoms, in shape (nframes, natoms, 3).

cells
[np.ndarray] The cell vectors of the system, in shape (nframes, 9). If the system is not periodic, set it to None.

atom_types
[List[int] or np.ndarray] The types of the atoms. If mixed_type is False, the shape is (natoms,); otherwise, the shape is (nframes, natoms).

atomic
[bool, optional] Whether to return atomic energy and atomic virial, by default False.

fparam
[np.ndarray, optional] The frame parameters, by default None.

aparam
[np.ndarray, optional] The atomic parameters, by default None.

mixed_type
[bool, optional] Whether the atom_types is mixed type, by default False.

**kwargs
[Dict[str, *Any*]] Keyword arguments.

Returns

energy
The energy of the system, in shape (nframes,).

force
The force of the system, in shape (nframes, natoms, 3).

virial
The virial of the system, in shape (nframes, 9).

atomic_energy
The atomic energy of the system, in shape (nframes, natoms). Only returned when atomic is True.

atomic_virial
The atomic virial of the system, in shape (nframes, natoms, 9). Only returned when atomic is True.

get_numb_dos() → int

class deepmd.tf.infer.DeepEval(model_file: pathlib.Path, output_def: deepmd.dpmodel.output_def.ModelOutputDef, *args: list, load_prefix: str = 'load', default_tf_graph: bool = False, auto_batch_size: bool | int | deepmd.tf.utils.batch_size.AutoBatchSize = False, input_map: dict | None = None, neighbor_list=None, **kwargs: dict)

Bases: *deepmd.infer.deep_eval.DeepEvalBackend*

TensorFlow backend implementation for DeepEval.

Parameters

model_file
[Path] The name of the frozen model file.

output_def
[ModelOutputDef] The output definition of the model.

*args
[list] Positional arguments.

load_prefix: str
The prefix in the load computational graph

default_tf_graph
[bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

auto_batch_size
[bool or int or AutomaticBatchSize, default: False] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

input_map
[dict, optional] The input map for tf.import_graph_def. Only work with default tf graph

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

property model_type: *deepmd.infer.deep_eval.DeepEval*

Get type of model.

:type:str

property model_version: str

Get version of model.

Returns

str
version of model

property sess: deepmd.tf.env.tf.Session

Get TF session.

_init_tensors()

_init_attr()

_graph_compatable() → bool

Check the model compatability.

Returns

bool
If the model stored in the graph file is compatable with the current code

_get_tensor(tensor_name: str) → deepmd.tf.env.tf.Tensor

Get TF graph tensor.

Parameters

tensor_name
[str] name of tensor to get

Returns

tf.Tensor
loaded tensor

static _load_graph(frozen_graph_filename: pathlib.Path, prefix: str = 'load', default_tf_graph: bool = False, input_map: dict | None = None)

static sort_input(coord: numpy.ndarray, atom_type: numpy.ndarray, sel_atoms: List[int] | None = None)

Sort atoms in the system according their types.

Parameters

coord
The coordinates of atoms. Should be of shape [nframes, natoms, 3]

atom_type
The type of atoms Should be of shape [natoms]

sel_atoms
The selected atoms by type

Returns

coord_out
The coordinates after sorting

atom_type_out
The atom types after sorting

idx_map
The index mapping from the input to the output. For example coord_out = coord[:,idx_map,:]

sel_atom_type
Only output if sel_atoms is not None The sorted selected atom types

sel_idx_map
Only output if sel_atoms is not None The index mapping from the selected atoms to sorted selected atoms.

static reverse_map(vec: numpy.ndarray, imap: List[int]) → numpy.ndarray

Reverse mapping of a vector according to the index map.

Parameters

vec
Input vector. Be of shape [nframes, natoms, -1]

imap
Index map. Be of shape [natoms]

Returns

vec_out
Reverse mapped vector.

make_natoms_vec(atom_types: numpy.ndarray) → numpy.ndarray

Make the natom vector used by deepmd-kit.

Parameters

atom_types
The type of atoms

Returns

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

eval_typeebd() → numpy.ndarray

Evaluate output of type embedding network by using this model.

Returns

np.ndarray
The output of type embedding network. The shape is [ntypes, o_size], where ntypes is the number of types, and o_size is the number of nodes in the output layer.

Raises

KeyError
If the model does not enable type embedding.

See also:

deepmd.tf.utils.type_embed.TypeEmbedNet
The type embedding network.

Examples

Get the output of type embedding network of graph.pb:

```
>>> from deepmd.tf.infer import DeepPotential
>>> dp = DeepPotential("graph.pb")
>>> dp.eval_typeebd()
```

build_neighbor_list(coords: numpy.ndarray, cell: numpy.ndarray | None, atype: numpy.ndarray, imap: numpy.ndarray, neighbor_list)

Make the mesh with neighbor list for a single frame.

Parameters

coords
[np.ndarray] The coordinates of atoms. Should be of shape [natoms, 3]

cell
[Optional[np.ndarray]] The cell of the system. Should be of shape [3, 3]

atype
[np.ndarray] The type of atoms. Should be of shape [natoms]

imap
[np.ndarray] The index map of atoms. Should be of shape [natoms]

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList] ASE neighbor list. The following method or attribute will be used/set: bothways, self_interaction, update, build, first_neigh, pair_second, offset_vec.

Returns

natoms_vec
[np.ndarray] The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: nloc natoms[1]: nall natoms[i]: 2 <= i < Ntypes+2, number of type i atoms for nloc

coords
[np.ndarray] The coordinates of atoms, including ghost atoms. Should be of shape [nframes, nall, 3]

atype
[np.ndarray] The type of atoms, including ghost atoms. Should be of shape [nall]

mesh
[np.ndarray] The mesh in nei_mode=4.

imap
[np.ndarray] The index map of atoms. Should be of shape [nall]

ghost_map
[np.ndarray] The index map of ghost atoms. Should be of shape [nghost]

get_ntypes() → int

Get the number of atom types of this model.

get_ntypes_spin() → int

Get the number of spin atom types of this model.

get_rcut() → float

Get the cut-off radius of this model.

get_type_map() → List[str]

Get the type map (element name of the atom types) of this model.

get_sel_type() → numpy.ndarray | None

Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

get_dim_fparam() → int

Get the number (dimension) of frame parameters of this DP.

`get_dim_aparam()` → int

Get the number (dimension) of atomic parameters of this DP.

`_eval_func`(inner_func: Callable, numb_test: int, natoms: int) → Callable

Wrapper method with auto batch size.

Parameters

inner_func
[`Callable`] the method to be wrapped

numb_test
[`int`] number of tests

natoms
[`int`] number of atoms

Returns

`Callable`
the wrapper

`_get_natoms_and_nframes`(coords: numpy.ndarray, atom_types: List[int] | numpy.ndarray) → Tuple[int, int]

`eval`(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, efield: numpy.ndarray | None = None, **kwargs: Dict[str, Any]) → Dict[str, numpy.ndarray]

Evaluate the energy, force and virial by using this DP.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

atomic
Calculate the atomic energy and virial

fparam
The frame parameter. The array can be of size : - nframes x dim_fparam. - dim_fparam. Then all frames are assumed to be provided with the same fparam.

aparam
The atomic parameter The array can be of size : - nframes x natoms x dim_aparam. - natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. - dim_aparam. Then all frames and atoms are provided with the same aparam.

efield
The external field on atoms. The array should be of size nframes x natoms x 3

**kwargs
Other parameters

Returns

output_dict
[dict] The output of the evaluation. The keys are the names of the output variables, and the values are the corresponding output arrays.

_prepare_feed_dict(coords, cells, atom_types, fparam=None, aparam=None, efield=None)

_eval_inner(coords, cells, atom_types, fparam=None, aparam=None, efield=None, **kwargs)

_get_output_shape(odef, nframes, natoms)

eval_descriptor(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: numpy.ndarray, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, efield: numpy.ndarray | None = None) → numpy.ndarray

Evaluate descriptors by using this DP.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

fparam
The frame parameter. The array can be of size : - nframes x dim_fparam. - dim_fparam. Then all frames are assumed to be provided with the same fparam.

aparam
The atomic parameter The array can be of size : - nframes x natoms x dim_aparam. - natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. - dim_aparam. Then all frames and atoms are provided with the same aparam.

efield
The external field on atoms. The array should be of size nframes x natoms x 3

Returns

descriptor
Descriptors.

_eval_descriptor_inner(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: numpy.ndarray, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, efield: numpy.ndarray | None = None) → numpy.ndarray

get_numb_dos() → int

Get the number of DOS.

get_has_efield() → bool

Check if the model has efield.

class deepmd.tf.infer.DeepGlobalPolar(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_tensor.DeepTensor*

Deep Tensor Model.

Parameters

model_file
[Path] The name of the frozen model file.

*args
[list] Positional arguments.

auto_batch_size
[bool or int or AutoBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[dict] Keyword arguments.

property output_tensor_name: str

The name of the tensor.

eval(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: List[int] | numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: dict) → numpy.ndarray

Evaluate the model.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
[list[int] or np.ndarray] The atom types The list should contain natoms ints

atomic
If True (default), return the atomic tensor Otherwise return the global tensor

fparam
Not used in this model

aparam
Not used in this model

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

tensor
The returned tensor If atomic == False then of size nframes x output_dim else of size nframes x natoms x output_dim

class deepmd.tf.infer.DeepPolar(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_tensor.DeepTensor*

Deep polar model.

Parameters

model_file
[**Path**] The name of the frozen model file.

*args
[**list**] Positional arguments.

auto_batch_size
[bool or **int** or **AutoBatchSize**, default: **True**] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[**ase.neighborlist.NewPrimitiveNeighborList**, **optional**] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[**dict**] Keyword arguments.

property output_tensor_name: str
The name of the tensor.

class deepmd.tf.infer.DeepPot(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_eval.DeepEval*

Potential energy model.

Parameters

model_file
[**Path**] The name of the frozen model file.

*args
[**list**] Positional arguments.

auto_batch_size
[bool or **int** or **AutoBatchSize**, default: **True**] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[**ase.neighborlist.NewPrimitiveNeighborList**, **optional**] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[**dict**] Keyword arguments.

Examples

```
>>> from deepmd.infer import DeepPot
>>> import numpy as np
>>> dp = DeepPot("graph.pb")
>>> coord = np.array([[1, 0, 0], [0, 0, 1.5], [1, 0, 3]]).reshape([1, -1])
>>> cell = np.diag(10 * np.ones(3)).reshape([1, -1])
>>> atype = [1, 0, 1]
>>> e, f, v = dp.eval(coord, cell, atype)
```

where e, f and v are predicted energy, force and virial of the system, respectively.

property output_def: *deepmd.dpmodel.output_def.ModelOutputDef*

Get the output definition of this model.

property output_def_mag: *deepmd.dpmodel.output_def.ModelOutputDef*

Get the output definition of this model with magnetic parts.

eval(coords: numpy.ndarray, cells: numpy.ndarray | None, atom_types: List[int] | numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, mixed_type: bool = False, **kwargs: Dict[str, Any]) → Tuple[numpy.ndarray, Ellipsis]

Evaluate energy, force, and virial. If atomic is True, also return atomic energy and atomic virial.

Parameters

coords
: [np.ndarray] The coordinates of the atoms, in shape (nframes, natoms, 3).

cells
: [np.ndarray] The cell vectors of the system, in shape (nframes, 9). If the system is not periodic, set it to None.

atom_types
: [List[int] or np.ndarray] The types of the atoms. If mixed_type is False, the shape is (natoms,); otherwise, the shape is (nframes, natoms).

atomic
: [bool, optional] Whether to return atomic energy and atomic virial, by default False.

fparam
: [np.ndarray, optional] The frame parameters, by default None.

aparam
: [np.ndarray, optional] The atomic parameters, by default None.

mixed_type
: [bool, optional] Whether the atom_types is mixed type, by default False.

**kwargs
: [Dict[str, *Any*]] Keyword arguments.

Returns

energy
: The energy of the system, in shape (nframes,).

force
: The force of the system, in shape (nframes, natoms, 3).

virial
The virial of the system, in shape (nframes, 9).

atomic_energy
The atomic energy of the system, in shape (nframes, natoms). Only returned when atomic is True.

atomic_virial
The atomic virial of the system, in shape (nframes, natoms, 9). Only returned when atomic is True.

class deepmd.tf.infer.DeepWFC(model_file: str, *args: List[Any], auto_batch_size: bool | int | deepmd.utils.batch_size.AutoBatchSize = True, neighbor_list: ase.neighborlist.NewPrimitiveNeighborList | None = None, **kwargs: Dict[str, Any])

Bases: *deepmd.infer.deep_tensor.DeepTensor*

Deep WFC model.

Parameters

model_file
[**Path**] The name of the frozen model file.

*args
[**list**] Positional arguments.

auto_batch_size
[bool or **int** or **AutoBatchSize**, default: **True**] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

neighbor_list
[**ase.neighborlist.NewPrimitiveNeighborList**, **optional**] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
[**dict**] Keyword arguments.

property output_tensor_name: str
The name of the tensor.

class deepmd.tf.infer.EwaldRecp(hh, beta)

Evaluate the reciprocal part of the Ewald sum.

eval(coord: numpy.ndarray, charge: numpy.ndarray, box: numpy.ndarray) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Evaluate.

Parameters

coord
The coordinates of atoms

charge
The atomic charge

box
The simulation region. PBC is assumed

Returns

e
The energy

f
The force

v
The virial

deepmd.tf.infer.calc_model_devi(coord, box, atype, models, fname=None, frequency=1, mixed_type=False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, real_data: dict | None = None, atomic: bool = False, relative: float | None = None, relative_v: float | None = None)

Python interface to calculate model deviation.

Parameters

coord
[numpy.ndarray, n_frames x n_atoms x 3] Coordinates of system to calculate

box
[numpy.ndarray or None, n_frames x 3 x 3] Box to specify periodic boundary condition. If None, no pbc will be used

atype
[numpy.ndarray, n_atoms x 1] Atom types

models
[list of *DeepPot* models] Models used to evaluate deviation

fname
[str or None] File to dump results, default None

frequency
[int] Steps between frames (if the system is given by molecular dynamics engine), default 1

mixed_type
[bool] Whether the input atype is in mixed_type format or not

fparam
[numpy.ndarray] frame specific parameters

aparam
[numpy.ndarray] atomic specific parameters

real_data
[dict, optional] real data to calculate RMS real error

atomic
[bool, default: False] If True, calculate the force model deviation of each atom.

relative
[float, default: None] If given, calculate the relative model deviation of force. The value is the level parameter for computing the relative model deviation of the force.

relative_v
[float, default: None] If given, calculate the relative model deviation of virial. The value is the level parameter for computing the relative model deviation of the virial.

Returns

model_devi
: [numpy.ndarray, n_frames x 8] Model deviation results. The first column is index of steps, the other 7 columns are max_devi_v, min_devi_v, avg_devi_v, max_devi_f, min_devi_f, avg_devi_f, devi_e.

Examples

```
>>> from deepmd.tf.infer import calc_model_devi
>>> from deepmd.tf.infer import DeepPot as DP
>>> import numpy as np
>>> coord = np.array([[1, 0, 0], [0, 0, 1.5], [1, 0, 3]]).reshape([1, -1])
>>> cell = np.diag(10 * np.ones(3)).reshape([1, -1])
>>> atype = [1, 0, 1]
>>> graphs = [DP("graph.000.pb"), DP("graph.001.pb")]
>>> model_devi = calc_model_devi(coord, cell, atype, graphs)
```

deepmd.tf.loggers

Alias of deepmd.loggers for backward compatibility.

Submodules

deepmd.tf.loggers.loggers

Alias of deepmd.loggers.loggers for backward compatibility.

Module Contents

Functions

set_log_handles(level[, log_path, mpi_log])	Set desired level for package loggers and add file handlers.

deepmd.tf.loggers.loggers.**set_log_handles**(level: int, log_path: pathlib.Path | None = None, mpi_log: str | None = None)

Set desired level for package loggers and add file handlers.

Parameters

level
: [int] logging level

log_path
: [Optional[str]] path to log file, if None logs will be send only to console. If the parent directory does not exist it will be automatically created, by default None

mpi_log
: [Optional[str], optional] mpi log type. Has three options. master will output logs to file and console only from rank==0. collect will write messages from all ranks

to one file opened under rank==0 and to console. workers will open one log file for each worker designated by its rank, console behaviour is the same as for collect. If this argument is specified, package 'mpi4py' must be already installed. by default None

Raises

`RuntimeError`
If the argument mpi_log is specified, package mpi4py is not installed.

Notes

Logging levels:

	our notation	python logging	tensorflow cpp	OpenMP
debug	10	10	0	1/on/true/yes
info	20	20	1	0/off/false/no
warning	30	30	2	0/off/false/no
error	40	40	3	0/off/false/no

References

https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U https://stackoverflow.com/questions/35869137/avoid-tensorflow-print-on-standard-error https://stackoverflow.com/questions/56085015/suppress-openmp-debug-messages-when-running-tensorflow-on-cpu

Package Contents

Functions

set_log_handles(level[, log_path, mpi_log])	Set desired level for package loggers and add file handlers.

deepmd.tf.loggers.set_log_handles(level: int, log_path: pathlib.Path | None = None, mpi_log: str | None = None)

Set desired level for package loggers and add file handlers.

Parameters

level
[int] logging level

log_path
[Optional[str]] path to log file, if None logs will be send only to console. If the parent directory does not exist it will be automatically created, by default None

mpi_log
[Optional[str], optional] mpi log type. Has three options. master will output logs to file and console only from rank==0. collect will write messages from all ranks to one file opened under rank==0 and to console. workers will open one log file for each worker designated by its rank, console behaviour is the same as for collect. If

this argument is specified, package 'mpi4py' must be already installed. by default None

Raises

`RuntimeError`
If the argument mpi_log is specified, package mpi4py is not installed.

Notes

Logging levels:

	our notation	python logging	tensorflow cpp	OpenMP
debug	10	10	0	1/on/true/yes
info	20	20	1	0/off/false/no
warning	30	30	2	0/off/false/no
error	40	40	3	0/off/false/no

References

https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U https://stackoverflow.com/questions/35869137/avoid-tensorflow-print-on-standard-error https://stackoverflow.com/questions/56085015/suppress-openmp-debug-messages-when-running-tensorflow-on-cpu

deepmd.tf.loss

Submodules

deepmd.tf.loss.dos

Module Contents

Classes

DOSLoss	Loss function for DeepDOS models.

class deepmd.tf.loss.dos.**DOSLoss**(starter_learning_rate: float, numb_dos: int = 500, start_pref_dos: float = 1.0, limit_pref_dos: float = 1.0, start_pref_cdf: float = 1000, limit_pref_cdf: float = 1.0, start_pref_ados: float = 0.0, limit_pref_ados: float = 0.0, start_pref_acdf: float = 0.0, limit_pref_acdf: float = 0.0, protect_value: float = 1e-08, log_fit: bool = False, **kwargs)

Bases: *deepmd.tf.loss.loss.Loss*

Loss function for DeepDOS models.

`build`(learning_rate, natoms, model_dict, label_dict, suffix)

Build the loss function graph.

Parameters

learning_rate
[`tf.Tensor`] learning rate

natoms
[`tf.Tensor`] number of atoms

model_dict
[`dict`[`str`, `tf.Tensor`]] A dictionary that maps model keys to tensors

label_dict
[`dict`[`str`, `tf.Tensor`]] A dictionary that maps label keys to tensors

suffix
[`str`] suffix

Returns

`tf.Tensor`
the total squared loss

`dict`[`str`, `tf.Tensor`]
A dictionary that maps loss keys to more loss tensors

`eval`(sess, feed_dict, natoms)

Eval the loss function.

Parameters

sess
[`tf.Session`] TensorFlow session

feed_dict
[`dict`[`tf.placeholder`, `tf.Tensor`]] A dictionary that maps graph elements to values

natoms
[`tf.Tensor`] number of atoms

Returns

`dict`
A dictionary that maps keys to values. It should contain key natoms

deepmd.tf.loss.ener

Module Contents

Classes

EnerStdLoss	Standard loss function for DP models.
EnerSpinLoss	The abstract class for the loss function.
EnerDipoleLoss	The abstract class for the loss function.

class deepmd.tf.loss.ener.**EnerStdLoss**(starter_learning_rate: float, start_pref_e: float = 0.02, limit_pref_e: float = 1.0, start_pref_f: float = 1000, limit_pref_f: float = 1.0, start_pref_v: float = 0.0, limit_pref_v: float = 0.0, start_pref_ae: float = 0.0, limit_pref_ae: float = 0.0, start_pref_pf: float = 0.0, limit_pref_pf: float = 0.0, relative_f: float | None = None, enable_atom_ener_coeff: bool = False, start_pref_gf: float = 0.0, limit_pref_gf: float = 0.0, numb_generalized_coord: int = 0, **kwargs)

Bases: *deepmd.tf.loss.loss.Loss*

Standard loss function for DP models.

Parameters

starter_learning_rate
: [float] The learning rate at the start of the training.

start_pref_e
: [float] The prefactor of energy loss at the start of the training.

limit_pref_e
: [float] The prefactor of energy loss at the end of the training.

start_pref_f
: [float] The prefactor of force loss at the start of the training.

limit_pref_f
: [float] The prefactor of force loss at the end of the training.

start_pref_v
: [float] The prefactor of virial loss at the start of the training.

limit_pref_v
: [float] The prefactor of virial loss at the end of the training.

start_pref_ae
: [float] The prefactor of atomic energy loss at the start of the training.

limit_pref_ae
: [float] The prefactor of atomic energy loss at the end of the training.

start_pref_pf
: [float] The prefactor of atomic prefactor force loss at the start of the training.

limit_pref_pf
: [float] The prefactor of atomic prefactor force loss at the end of the training.

relative_f
: [float] If provided, relative force error will be used in the loss. The difference of force will be normalized by the magnitude of the force in the label with a shift given by relative_f

enable_atom_ener_coeff
: [bool] if true, the energy will be computed as sum_i c_i E_i

start_pref_gf
: [float] The prefactor of generalized force loss at the start of the training.

limit_pref_gf
: [float] The prefactor of generalized force loss at the end of the training.

numb_generalized_coord
[int] The dimension of generalized coordinates.

**kwargs
Other keyword arguments.

build(learning_rate, natoms, model_dict, label_dict, suffix)

Build the loss function graph.

Parameters

learning_rate
[tf.Tensor] learning rate

natoms
[tf.Tensor] number of atoms

model_dict
[dict[str, tf.Tensor]] A dictionary that maps model keys to tensors

label_dict
[dict[str, tf.Tensor]] A dictionary that maps label keys to tensors

suffix
[str] suffix

Returns

tf.Tensor
the total squared loss

dict[str, tf.Tensor]
A dictionary that maps loss keys to more loss tensors

eval(sess, feed_dict, natoms)

Eval the loss function.

Parameters

sess
[tf.Session] TensorFlow session

feed_dict
[dict[tf.placeholder, tf.Tensor]] A dictionary that maps graph elements to values

natoms
[tf.Tensor] number of atoms

Returns

dict
A dictionary that maps keys to values. It should contain key natoms

class deepmd.tf.loss.ener.**EnerSpinLoss**(starter_learning_rate: float, start_pref_e: float = 0.02, limit_pref_e: float = 1.0, start_pref_fr: float = 1000, limit_pref_fr: float = 1.0, start_pref_fm: float = 10000, limit_pref_fm: float = 10.0, start_pref_v: float = 0.0, limit_pref_v: float = 0.0, start_pref_ae: float = 0.0, limit_pref_ae: float = 0.0, start_pref_pf: float = 0.0, limit_pref_pf: float = 0.0, relative_f: float | None = None, enable_atom_ener_coeff: bool = False, use_spin: list | None = None)

Bases: *deepmd.tf.loss.loss.Loss*

The abstract class for the loss function.

build(learning_rate, natoms, model_dict, label_dict, suffix)

Build the loss function graph.

Parameters

learning_rate
[tf.Tensor] learning rate

natoms
[tf.Tensor] number of atoms

model_dict
[dict[str, tf.Tensor]] A dictionary that maps model keys to tensors

label_dict
[dict[str, tf.Tensor]] A dictionary that maps label keys to tensors

suffix
[str] suffix

Returns

tf.Tensor
the total squared loss

dict[**str**, **tf.Tensor**]
A dictionary that maps loss keys to more loss tensors

eval(sess, feed_dict, natoms)

Eval the loss function.

Parameters

sess
[**tf.Session**] TensorFlow session

feed_dict
[dict[**tf.placeholder**, tf.Tensor]] A dictionary that maps graph elements to values

natoms
[tf.Tensor] number of atoms

Returns

dict
A dictionary that maps keys to values. It should contain key natoms

print_header()

print_on_training(tb_writer, cur_batch, sess, natoms, feed_dict_test, feed_dict_batch)

class deepmd.tf.loss.ener.EnerDipoleLoss(starter_learning_rate: float, start_pref_e: float = 0.1, limit_pref_e: float = 1.0, start_pref_ed: float = 1.0, limit_pref_ed: float = 1.0)

Bases: *deepmd.tf.loss.loss.Loss*

The abstract class for the loss function.

build(learning_rate, natoms, model_dict, label_dict, suffix)

Build the loss function graph.

Parameters

learning_rate
[tf.Tensor] learning rate

natoms
[tf.Tensor] number of atoms

model_dict
[dict[str, tf.Tensor]] A dictionary that maps model keys to tensors

label_dict
[dict[str, tf.Tensor]] A dictionary that maps label keys to tensors

suffix
[str] suffix

Returns

tf.Tensor
the total squared loss

dict[str, tf.Tensor]
A dictionary that maps loss keys to more loss tensors

eval(sess, feed_dict, natoms)

Eval the loss function.

Parameters

sess
[tf.Session] TensorFlow session

feed_dict
[dict[tf.placeholder, tf.Tensor]] A dictionary that maps graph elements to values

natoms
[tf.Tensor] number of atoms

Returns

dict
A dictionary that maps keys to values. It should contain key natoms

deepmd.tf.loss.loss

Module Contents

Classes

Loss	The abstract class for the loss function.

class deepmd.tf.loss.loss.Loss

The abstract class for the loss function.

abstract build(learning_rate: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, model_dict: Dict[str, deepmd.tf.env.tf.Tensor], label_dict: Dict[str, deepmd.tf.env.tf.Tensor], suffix: str) → Tuple[deepmd.tf.env.tf.Tensor, Dict[str, deepmd.tf.env.tf.Tensor]]

Build the loss function graph.

Parameters

learning_rate
[tf.Tensor] learning rate

natoms
[tf.Tensor] number of atoms

model_dict
[dict[str, tf.Tensor]] A dictionary that maps model keys to tensors

label_dict
[dict[str, tf.Tensor]] A dictionary that maps label keys to tensors

suffix
[str] suffix

Returns

tf.Tensor
the total squared loss

dict[str, tf.Tensor]
A dictionary that maps loss keys to more loss tensors

abstract eval(sess: deepmd.tf.env.tf.Session, feed_dict: Dict[deepmd.tf.env.tf.placeholder, deepmd.tf.env.tf.Tensor], natoms: deepmd.tf.env.tf.Tensor) → dict

Eval the loss function.

Parameters

sess
[tf.Session] TensorFlow session

feed_dict
[dict[tf.placeholder, tf.Tensor]] A dictionary that maps graph elements to values

natoms
[tf.Tensor] number of atoms

Returns

dict
A dictionary that maps keys to values. It should contain key natoms

static display_if_exist(loss: deepmd.tf.env.tf.Tensor, find_property: float) → deepmd.tf.env.tf.Tensor

Display NaN if labeled property is not found.

Parameters

loss
[tf.Tensor] the loss tensor

find_property
[float] whether the property is found

deepmd.tf.loss.tensor

Module Contents

Classes

TensorLoss	Loss function for tensorial properties.

class deepmd.tf.loss.tensor.TensorLoss(jdata, **kwarg)

Bases: *deepmd.tf.loss.loss.Loss*

Loss function for tensorial properties.

build(learning_rate, natoms, model_dict, label_dict, suffix)

Build the loss function graph.

Parameters

learning_rate
[tf.Tensor] learning rate

natoms
[tf.Tensor] number of atoms

model_dict
[dict[str, tf.Tensor]] A dictionary that maps model keys to tensors

label_dict
[dict[str, tf.Tensor]] A dictionary that maps label keys to tensors

suffix
[str] suffix

Returns

tf.Tensor
the total squared loss

dict[str, tf.Tensor]
A dictionary that maps loss keys to more loss tensors

eval(sess, feed_dict, natoms)

Eval the loss function.

Parameters

sess
[tf.Session] TensorFlow session

feed_dict
[dict[tf.placeholder, tf.Tensor]] A dictionary that maps graph elements to values

natoms
[tf.Tensor] number of atoms

Returns

dict
A dictionary that maps keys to values. It should contain key natoms

Package Contents

Classes

DOSLoss	Loss function for DeepDOS models.
EnerDipoleLoss	The abstract class for the loss function.
EnerSpinLoss	The abstract class for the loss function.
EnerStdLoss	Standard loss function for DP models.
TensorLoss	Loss function for tensorial properties.

class deepmd.tf.loss.DOSLoss(starter_learning_rate: float, numb_dos: int = 500, start_pref_dos: float = 1.0, limit_pref_dos: float = 1.0, start_pref_cdf: float = 1000, limit_pref_cdf: float = 1.0, start_pref_ados: float = 0.0, limit_pref_ados: float = 0.0, start_pref_acdf: float = 0.0, limit_pref_acdf: float = 0.0, protect_value: float = 1e-08, log_fit: bool = False, **kwargs)

Bases: *deepmd.tf.loss.loss.Loss*

Loss function for DeepDOS models.

build(learning_rate, natoms, model_dict, label_dict, suffix)

Build the loss function graph.

Parameters

learning_rate
[tf.Tensor] learning rate

natoms
[tf.Tensor] number of atoms

model_dict
[dict[str, tf.Tensor]] A dictionary that maps model keys to tensors

label_dict
[dict[str, tf.Tensor]] A dictionary that maps label keys to tensors

suffix
[str] suffix

Returns

tf.Tensor
the total squared loss

dict[**str**, **tf.Tensor**]
A dictionary that maps loss keys to more loss tensors

eval(sess, feed_dict, natoms)

Eval the loss function.

Parameters

sess
[**tf.Session**] TensorFlow session

feed_dict
[dict[**tf.placeholder**, tf.Tensor]] A dictionary that maps graph elements to values

natoms
[tf.Tensor] number of atoms

Returns

dict
A dictionary that maps keys to values. It should contain key natoms

class deepmd.tf.loss.EnerDipoleLoss(starter_learning_rate: float, start_pref_e: float = 0.1, limit_pref_e: float = 1.0, start_pref_ed: float = 1.0, limit_pref_ed: float = 1.0)

Bases: *deepmd.tf.loss.loss.Loss*

The abstract class for the loss function.

build(learning_rate, natoms, model_dict, label_dict, suffix)

Build the loss function graph.

Parameters

learning_rate
[tf.Tensor] learning rate

natoms
[tf.Tensor] number of atoms

model_dict
[dict[str, tf.Tensor]] A dictionary that maps model keys to tensors

label_dict
[dict[str, tf.Tensor]] A dictionary that maps label keys to tensors

suffix
[str] suffix

Returns

tf.Tensor
the total squared loss

dict[str, tf.Tensor]
A dictionary that maps loss keys to more loss tensors

eval(sess, feed_dict, natoms)

Eval the loss function.

Parameters

sess
[tf.Session] TensorFlow session

feed_dict
[dict[tf.placeholder, tf.Tensor]] A dictionary that maps graph elements to values

natoms
[tf.Tensor] number of atoms

Returns

dict
A dictionary that maps keys to values. It should contain key natoms

class deepmd.tf.loss.EnerSpinLoss(starter_learning_rate: float, start_pref_e: float = 0.02, limit_pref_e: float = 1.0, start_pref_fr: float = 1000, limit_pref_fr: float = 1.0, start_pref_fm: float = 10000, limit_pref_fm: float = 10.0, start_pref_v: float = 0.0, limit_pref_v: float = 0.0, start_pref_ae: float = 0.0, limit_pref_ae: float = 0.0, start_pref_pf: float = 0.0, limit_pref_pf: float = 0.0, relative_f: float | None = None, enable_atom_ener_coeff: bool = False, use_spin: list | None = None)

Bases: *deepmd.tf.loss.loss.Loss*

The abstract class for the loss function.

build(learning_rate, natoms, model_dict, label_dict, suffix)

Build the loss function graph.

Parameters

learning_rate
[tf.Tensor] learning rate

natoms
[tf.Tensor] number of atoms

model_dict
[dict[str, tf.Tensor]] A dictionary that maps model keys to tensors

label_dict
[dict[str, tf.Tensor]] A dictionary that maps label keys to tensors

suffix
[str] suffix

Returns

tf.Tensor
the total squared loss

dict[str, tf.Tensor]
A dictionary that maps loss keys to more loss tensors

eval(sess, feed_dict, natoms)

Eval the loss function.

Parameters

sess
[tf.Session] TensorFlow session

feed_dict
[dict[tf.placeholder, tf.Tensor]] A dictionary that maps graph elements to values

natoms
[tf.Tensor] number of atoms

Returns

dict
A dictionary that maps keys to values. It should contain key natoms

print_header()

print_on_training(tb_writer, cur_batch, sess, natoms, feed_dict_test, feed_dict_batch)

class deepmd.tf.loss.EnerStdLoss(starter_learning_rate: float, start_pref_e: float = 0.02, limit_pref_e: float = 1.0, start_pref_f: float = 1000, limit_pref_f: float = 1.0, start_pref_v: float = 0.0, limit_pref_v: float = 0.0, start_pref_ae: float = 0.0, limit_pref_ae: float = 0.0, start_pref_pf: float = 0.0, limit_pref_pf: float = 0.0, relative_f: float | None = None, enable_atom_ener_coeff: bool = False, start_pref_gf: float = 0.0, limit_pref_gf: float = 0.0, numb_generalized_coord: int = 0, **kwargs)

Bases: *deepmd.tf.loss.loss.Loss*

Standard loss function for DP models.

Parameters

starter_learning_rate
: [float] The learning rate at the start of the training.

start_pref_e
: [float] The prefactor of energy loss at the start of the training.

limit_pref_e
: [float] The prefactor of energy loss at the end of the training.

start_pref_f
: [float] The prefactor of force loss at the start of the training.

limit_pref_f
: [float] The prefactor of force loss at the end of the training.

start_pref_v
: [float] The prefactor of virial loss at the start of the training.

limit_pref_v
: [float] The prefactor of virial loss at the end of the training.

start_pref_ae
: [float] The prefactor of atomic energy loss at the start of the training.

limit_pref_ae
: [float] The prefactor of atomic energy loss at the end of the training.

start_pref_pf
: [float] The prefactor of atomic prefactor force loss at the start of the training.

limit_pref_pf
: [float] The prefactor of atomic prefactor force loss at the end of the training.

relative_f
: [float] If provided, relative force error will be used in the loss. The difference of force will be normalized by the magnitude of the force in the label with a shift given by relative_f

enable_atom_ener_coeff
: [bool] if true, the energy will be computed as sum_i c_i E_i

start_pref_gf
: [float] The prefactor of generalized force loss at the start of the training.

limit_pref_gf
: [float] The prefactor of generalized force loss at the end of the training.

numb_generalized_coord
[int] The dimension of generalized coordinates.

**kwargs
Other keyword arguments.

build(learning_rate, natoms, model_dict, label_dict, suffix)

Build the loss function graph.

Parameters

learning_rate
[tf.Tensor] learning rate

natoms
[tf.Tensor] number of atoms

model_dict
[dict[str, tf.Tensor]] A dictionary that maps model keys to tensors

label_dict
[dict[str, tf.Tensor]] A dictionary that maps label keys to tensors

suffix
[str] suffix

Returns

tf.Tensor
the total squared loss

dict[str, tf.Tensor]
A dictionary that maps loss keys to more loss tensors

eval(sess, feed_dict, natoms)

Eval the loss function.

Parameters

sess
[tf.Session] TensorFlow session

feed_dict
[dict[tf.placeholder, tf.Tensor]] A dictionary that maps graph elements to values

natoms
[tf.Tensor] number of atoms

Returns

dict
A dictionary that maps keys to values. It should contain key natoms

class deepmd.tf.loss.TensorLoss(jdata, **kwarg)

Bases: *deepmd.tf.loss.loss.Loss*

Loss function for tensorial properties.

build(learning_rate, natoms, model_dict, label_dict, suffix)

Build the loss function graph.

Parameters

learning_rate
[tf.Tensor] learning rate

natoms
[tf.Tensor] number of atoms

model_dict
[dict[str, tf.Tensor]] A dictionary that maps model keys to tensors

label_dict
[dict[str, tf.Tensor]] A dictionary that maps label keys to tensors

suffix
[str] suffix

Returns

tf.Tensor
the total squared loss

dict[str, tf.Tensor]
A dictionary that maps loss keys to more loss tensors

eval(sess, feed_dict, natoms)

Eval the loss function.

Parameters

sess
[tf.Session] TensorFlow session

feed_dict
[dict[tf.placeholder, tf.Tensor]] A dictionary that maps graph elements to values

natoms
[tf.Tensor] number of atoms

Returns

dict
A dictionary that maps keys to values. It should contain key natoms

deepmd.tf.model

Submodules

deepmd.tf.model.dos

Module Contents

Classes

DOSModel	DOS model.

class deepmd.tf.model.dos.DOSModel(descriptor: dict, fitting_net: dict, type_embedding: dict | deepmd.tf.utils.type_embed.TypeEmbedNet | None = None, type_map: List[str] | None = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01, **kwargs)

Bases: *deepmd.tf.model.model.StandardModel*

DOS model.

Parameters

descriptor
Descriptor

fitting_net
Fitting net

type_embedding
Type embedding net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
Number of frames used for data statistic

data_stat_protect
Protect parameter for atomic energy regression

model_type = 'dos'

get_numb_dos()
Get the number of gridpoints in energy space.

get_rcut()
Get cutoff radius of the model.

get_ntypes()
Get the number of types.

get_type_map()
Get the type map.

get_numb_fparam() → int
Get the number of frame parameters.

get_numb_aparam() → int
Get the number of atomic parameters.

data_stat(data)
Data staticis.

_compute_input_stat(all_stat, protection=0.01, mixed_type=False)

_compute_output_stat(all_stat, mixed_type=False)

build(coord_, atype_, natoms, box, mesh, input_dict, frz_model=None, ckpt_meta: str | None = None, suffix='', reuse=None)
Build the model.

Parameters

coord_
[tf.Tensor] The coordinates of atoms

atype_
[tf.Tensor] The atom types of atoms

natoms
[tf.Tensor] The number of atoms

box
[tf.Tensor] The box vectors

mesh
[tf.Tensor] The mesh vectors

input_dict
[dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

dict
The output dict

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None

Init the embedding net variables with the given frozen model.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

model_type
[str] the type of the model

suffix
[str] suffix to name scope

deepmd.tf.model.ener

Module Contents

Classes

EnerModel	Energy model.

class deepmd.tf.model.ener.EnerModel(descriptor: dict, fitting_net: dict, type_embedding: dict | deepmd.tf.utils.type_embed.TypeEmbedNet | None = None, type_map: List[str] | None = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01, use_srtab: str | None = None, smin_alpha: float | None = None, sw_rmin: float | None = None, sw_rmax: float | None = None, srtab_add_bias: bool = True, spin: deepmd.tf.utils.spin.Spin | None = None, data_bias_nsample: int = 10, **kwargs)

Bases: *deepmd.tf.model.model.StandardModel*

Energy model.

Parameters

descriptor
: Descriptor

fitting_net
: Fitting net

type_embedding
: Type embedding net

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
: Number of frames used for data statistic

data_stat_protect
: Protect parameter for atomic energy regression

use_srtab
: The table for the short-range pairwise interaction added on top of DP. The table is a text data file with (N_t + 1) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

smin_alpha
: The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when use_srtab is provided.

sw_rmin
: The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

sw_rmin
: The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

srtab_add_bias
: [bool] Whether add energy bias from the statistics of the data to short-range tabulated atomic energy. It only takes effect when use_srtab is provided.

spin
: spin

data_stat_nsample
: The number of training samples in a system to compute and change the energy bias.

model_type = 'ener'

get_rcut()

Get cutoff radius of the model.

get_ntypes()

Get the number of types.

get_type_map()

Get the type map.

get_numb_fparam() → int

Get the number of frame parameters.

get_numb_aparam() → int

Get the number of atomic parameters.

data_stat(data)

Data staticis.

_compute_input_stat(all_stat, protection=0.01, mixed_type=False)

_compute_output_stat(all_stat, mixed_type=False)

build(coord_, atype_, natoms, box, mesh, input_dict, frz_model=None, ckpt_meta: str | None = None, suffix='', reuse=None)

Build the model.

Parameters

coord_
: [tf.Tensor] The coordinates of atoms

atype_
: [tf.Tensor] The atom types of atoms

natoms
: [tf.Tensor] The number of atoms

box
: [tf.Tensor] The box vectors

mesh
: [tf.Tensor] The mesh vectors

input_dict
: [dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

dict
The output dict

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None

Init the embedding net variables with the given frozen model.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

model_type
[str] the type of the model

suffix
[str] suffix to name scope

natoms_match(force, natoms)

natoms_not_match(force, natoms, atype)

change_energy_bias(data: deepmd.tf.utils.data_system.DeepmdDataSystem, frozen_model: str, origin_type_map: list, full_type_map: str, bias_adjust_mode: str = 'change-by-statistic') → None

Change the energy bias according to the input data and the pretrained model.

Parameters

data
[DeepmdDataSystem] The training data.

frozen_model
[str] The path file of frozen model.

origin_type_map
[list] The original type_map in dataset, they are targets to change the energy bias.

full_type_map
[str] The full type_map in pretrained model

bias_adjust_mode
[str] The mode for changing energy bias : ['change-by-statistic', 'set-by-statistic']
'change-by-statistic' : perform predictions on energies of target dataset,

and do least sqaure on the errors to obtain the target shift as bias.

'set-by-statistic' : directly use the statistic energy bias in the target dataset.

deepmd.tf.model.frozen

Module Contents

Classes

FrozenModel	Load model from a frozen model, which cannot be trained.

class deepmd.tf.model.frozen.**FrozenModel**(model_file: str, **kwargs)

Bases: *deepmd.tf.model.model.Model*

Load model from a frozen model, which cannot be trained.

Parameters

model_file
[str] The path to the frozen model

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, frz_model: str | None = None, ckpt_meta: str | None = None, suffix: str = '', reuse: bool | enum.Enum | None = None) → dict

Build the model.

Parameters

coord_
[tf.Tensor] The coordinates of atoms

atype_
[tf.Tensor] The atom types of atoms

natoms
[tf.Tensor] The number of atoms

box
[tf.Tensor] The box vectors

mesh
[tf.Tensor] The mesh vectors

input_dict
[dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

dict
The output dict

get_fitting() → deepmd.tf.fit.fitting.Fitting | dict

Get the fitting(s).

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss | dict | None

Get the loss function(s).

get_rcut()

Get cutoff radius of the model.

get_ntypes() → int

Get the number of types.

data_stat(data)

Data staticis.

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None

Init the embedding net variables with the given frozen model.

Parameters

graph
[**tf.Graph**] The input frozen model graph

graph_def
[**tf.GraphDef**] The input frozen model graph_def

model_type
[**str**] the type of the model

suffix
[**str**] suffix to name scope

enable_compression(suffix: str = '') → None

Enable compression.

Parameters

suffix
[**str**] suffix to name scope

get_type_map() → list

Get the type map.

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[**dict**] The global data, containing the training section

local_jdata
[**dict**] The local data refer to the current class

serialize(suffix: str = '') → dict

Serialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Returns

dict
The serialized data

suffix
[str, optional] Name suffix to identify this descriptor

classmethod deserialize(data: dict, suffix: str = '')

Deserialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Parameters

data
[dict] The serialized data

suffix
[str, optional] Name suffix to identify this model

Returns

Model
The deserialized Model

deepmd.tf.model.linear

Module Contents

Classes

LinearModel	Linear model make linear combinations of several existing models.
LinearEnergyModel	Linear energy model make linear combinations of several existing energy models.

class deepmd.tf.model.linear.LinearModel(models: List[dict], weights: List[float], **kwargs)

Bases: *deepmd.tf.model.model.Model*

Linear model make linear combinations of several existing models.

Parameters

models
[list[dict]] A list of models to be combined.

weights
[list[float] or str] If the type is list[float], a list of weights for each model. If "mean", the weights are set to be 1 / len(models). If "sum", the weights are set to be 1.

`get_fitting()` → deepmd.tf.fit.fitting.Fitting | dict

Get the fitting(s).

`get_loss`(loss: dict, lr) → deepmd.tf.loss.loss.Loss | dict | None

Get the loss function(s).

`get_rcut()`

Get cutoff radius of the model.

`get_ntypes()` → int

Get the number of types.

`data_stat`(data)

Data staticis.

`init_variables`(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None

Init the embedding net variables with the given frozen model.

Parameters

graph
[`tf.Graph`] The input frozen model graph

graph_def
[`tf.GraphDef`] The input frozen model graph_def

model_type
[`str`] the type of the model

suffix
[`str`] suffix to name scope

`enable_compression`(suffix: str = '') → None

Enable compression.

Parameters

suffix
[`str`] suffix to name scope

`get_type_map()` → list

Get the type map.

`classmethod update_sel`(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[`dict`] The global data, containing the training section

local_jdata
[`dict`] The local data refer to the current class

`class deepmd.tf.model.linear.LinearEnergyModel`(models: List[dict], weights: List[float], **kwargs)

Bases: *LinearModel*

Linear energy model make linear combinations of several existing energy models.

`model_type = 'ener'`

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, frz_model: str | None = None, ckpt_meta: str | None = None, suffix: str = '', reuse: bool | enum.Enum | None = None) → dict

Build the model.

Parameters

coord_
[tf.Tensor] The coordinates of atoms

atype_
[tf.Tensor] The atom types of atoms

natoms
[tf.Tensor] The number of atoms

box
[tf.Tensor] The box vectors

mesh
[tf.Tensor] The mesh vectors

input_dict
[dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

dict
The output dict

deepmd.tf.model.model

Module Contents

Classes

Model	Abstract base model.
StandardModel	Standard model, which must contain a descriptor and a fitting.

class deepmd.tf.model.model.Model(type_embedding: dict | deepmd.tf.utils.type_embed.TypeEmbedNet | None = None, type_map: List[str] | None = None, data_stat_nbatch: int = 10, data_bias_nsample: int = 10, data_stat_protect: float = 0.01, use_srtab: str | None = None, smin_alpha: float | None = None, sw_rmin: float | None = None, sw_rmax: float | None = None, srtab_add_bias: bool = True, spin: deepmd.tf.utils.spin.Spin | None = None, compress: dict | None = None, **kwargs)

Bases: abc.ABC, make_plugin_registry('model')

Abstract base model.

Parameters

type_embedding
: Type embedding net

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
: Number of frames used for data statistic

data_bias_nsample
: The number of training samples in a system to compute and change the energy bias.

data_stat_protect
: Protect parameter for atomic energy regression

use_srtab
: The table for the short-range pairwise interaction added on top of DP. The table is a text data file with (N_t + 1) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

smin_alpha
: The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when use_srtab is provided.

sw_rmin
: The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

sw_rmin
: The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

srtab_add_bias
: [bool] Whether add energy bias from the statistics of the data to short-range tabulated atomic energy. It only takes effect when use_srtab is provided.

spin
: spin

compress
: Compression information for internal use

get_type_map() → list

Get the type map.

abstract build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, frz_model: str | None = None, ckpt_meta: str | None = None, suffix: str = '', reuse: bool | enum.Enum | None = None)

Build the model.

Parameters

coord_
[tf.Tensor] The coordinates of atoms

atype_
[tf.Tensor] The atom types of atoms

natoms
[tf.Tensor] The number of atoms

box
[tf.Tensor] The box vectors

mesh
[tf.Tensor] The mesh vectors

input_dict
[dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

dict
The output dict

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None

Init the embedding net variables with the given frozen model.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

model_type
[str] the type of the model

suffix
[str] suffix to name scope

build_descrpt(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, frz_model: str | None = None, ckpt_meta: str | None = None, suffix: str = '', reuse: bool | enum.Enum | None = None)

Build the descriptor part of the model.

Parameters

coord_
[tf.Tensor] The coordinates of atoms

atype_
[tf.Tensor] The atom types of atoms

natoms
[tf.Tensor] The number of atoms

box
[tf.Tensor] The box vectors

mesh
[tf.Tensor] The mesh vectors

input_dict
[dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

tf.Tensor
The descriptor tensor

build_type_embedding(ntypes: int, frz_model: str | None = None, ckpt_meta: str | None = None, suffix: str = '', reuse: bool | enum.Enum | None = None) → deepmd.tf.env.tf.Tensor

Build the type embedding part of the model.

Parameters

ntypes
[int] The number of types

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or `tf.AUTO_REUSE`, `optional`] Whether to reuse the variables

Returns

`tf.Tensor`
The type embedding tensor

_import_graph_def_from_frz_model(frz_model: str, feed_dict: dict, return_elements: List[str])

_import_graph_def_from_ckpt_meta(ckpt_meta: str, feed_dict: dict, return_elements: List[str])

enable_mixed_precision(mixed_prec: dict)

Enable mixed precision for the model.

Parameters

mixed_prec
[`dict`] The mixed precision config

change_energy_bias(data: deepmd.tf.utils.data_system.DeepmdDataSystem, frozen_model: str, origin_type_map: list, full_type_map: str, bias_adjust_mode: str = 'change-by-statistic') → None

Change the energy bias according to the input data and the pretrained model.

Parameters

data
[`DeepmdDataSystem`] The training data.

frozen_model
[`str`] The path file of frozen model.

origin_type_map
[`list`] The original type_map in dataset, they are targets to change the energy bias.

full_type_map
[`str`] The full type_map in pretrained model

bias_adjust_mode
[`str`] The mode for changing energy bias : ['change-by-statistic', 'set-by-statistic']
'change-by-statistic' : perform predictions on energies of target dataset,

and do least sqaure on the errors to obtain the target shift as bias.

'set-by-statistic' : directly use the statistic energy bias in the target dataset.

enable_compression(suffix: str = '')

Enable compression.

Parameters

suffix
[`str`] suffix to name scope

get_numb_fparam() → int | dict

Get the number of frame parameters.

get_numb_aparam() → int | dict

Get the number of atomic parameters.

get_numb_dos() → int | dict

Get the number of gridpoints in energy space.

abstract get_fitting() → deepmd.tf.fit.fitting.Fitting | dict

Get the fitting(s).

abstract get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss | dict | None

Get the loss function(s).

abstract get_rcut() → float

Get cutoff radius of the model.

abstract get_ntypes() → int

Get the number of types.

abstract data_stat(data: dict)

Data staticis.

get_feed_dict(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, **kwargs) → Dict[str, deepmd.tf.env.tf.Tensor]

Generate the feed_dict for current descriptor.

Parameters

coord_
[tf.Tensor] The coordinate of atoms

atype_
[tf.Tensor] The type of atoms

natoms
[tf.Tensor] The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box
[tf.Tensor] The box. Can be generated by deepmd.tf.model.make_stat_input

mesh
[tf.Tensor] For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

**kwargs
[dict] The additional arguments

Returns

feed_dict
[dict[str, tf.Tensor]] The output feed_dict of current descriptor

abstract classmethod update_sel(global_jdata: dict, local_jdata: dict) → dict

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

Returns

dict
The updated local data

Notes

Do not modify the input data without copying it.

classmethod deserialize(data: dict, suffix: str = '') → Model

Deserialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Parameters

data
[dict] The serialized data

suffix
[str, optional] Name suffix to identify this model

Returns

Model
The deserialized Model

abstract serialize(suffix: str = '') → dict

Serialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Returns

dict
The serialized data

suffix
[str, optional] Name suffix to identify this descriptor

class deepmd.tf.model.model.StandardModel(descriptor: dict | deepmd.tf.descriptor.descriptor.Descriptor, fitting_net: dict | deepmd.tf.fit.fitting.Fitting, type_embedding: dict | deepmd.tf.utils.type_embed.TypeEmbedNet | None = None, type_map: List[str] | None = None, **kwargs)

Bases: *Model*

Standard model, which must contain a descriptor and a fitting.

Parameters

descriptor
[Union[dict, Descriptor]] The descriptor

fitting_net
[Union[dict, Fitting]] The fitting network

type_embedding
[dict, optional] The type embedding

type_map
[list of dict, optional] The type map

enable_mixed_precision(mixed_prec: dict)

Enable mixed precision for the model.

Parameters

mixed_prec
[dict] The mixed precision config

enable_compression(suffix: str = '')

Enable compression.

Parameters

suffix
[str] suffix to name scope

get_fitting() → deepmd.tf.fit.fitting.Fitting | dict

Get the fitting(s).

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss | dict

Get the loss function(s).

get_rcut() → float

Get cutoff radius of the model.

get_ntypes() → int

Get the number of types.

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

classmethod deserialize(data: dict, suffix: str = '') → deepmd.tf.descriptor.descriptor.Descriptor

Deserialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Parameters

data
[dict] The serialized data

suffix
[str, optional] Name suffix to identify this descriptor

Returns

Descriptor
The deserialized descriptor

serialize(suffix: str = '') → dict

Serialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Returns

dict
The serialized data

suffix
[str, optional] Name suffix to identify this descriptor

deepmd.tf.model.model_stat

Alias for backward compatibility.

Module Contents

Functions

_make_all_stat_ref(data, nbatches)	
make_stat_input(data, nbatches[, merge_sys])	Pack data for statistics.
merge_sys_stat(all_stat)	

deepmd.tf.model.model_stat.**_make_all_stat_ref**(data, nbatches)

deepmd.tf.model.model_stat.**make_stat_input**(data, nbatches, merge_sys=True)

Pack data for statistics.

Parameters

data
The data

nbatches
[int] The number of batches

merge_sys
[bool (True)] Merge system data

Returns

all_stat:
A dictionary of list of list storing data for stat. if merge_sys == False data can be accessed by

all_stat[key][sys_idx][batch_idx][frame_idx]

else merge_sys == True can be accessed by
all_stat[key][batch_idx][frame_idx]

deepmd.tf.model.model_stat.**merge_sys_stat**(all_stat)

deepmd.tf.model.multi

Module Contents

Classes

MultiModel	Multi-task model.

class deepmd.tf.model.multi.MultiModel(descriptor: dict, fitting_net_dict: dict, fitting_type_dict: dict | None = None, type_embedding=None, type_map: List[str] | None = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01, use_srtab: str | None = None, smin_alpha: float | None = None, sw_rmin: float | None = None, sw_rmax: float | None = None, **kwargs)

Bases: *deepmd.tf.model.model.Model*

Multi-task model.

Parameters

descriptor
: Descriptor

fitting_net_dict
: Dictionary of fitting nets

fitting_type_dict
: deprecated argument

type_embedding
: Type embedding net

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
: Number of frames used for data statistic

data_stat_protect
: Protect parameter for atomic energy regression

use_srtab
: The table for the short-range pairwise interaction added on top of DP. The table is a text data file with (N_t + 1) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

smin_alpha
: The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when use_srtab is provided.

sw_rmin
: The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

sw_rmin
: The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

model_type = 'multi_task'

get_rcut()
: Get cutoff radius of the model.

get_ntypes()
: Get the number of types.

get_type_map()
: Get the type map.

data_stat(data)
: Data staticis.

_compute_input_stat(all_stat, protection=0.01, mixed_type=False, fitting_key='')

_compute_output_stat(all_stat, mixed_type=False, fitting_key='')

build(coord_, atype_, natoms, box, mesh, input_dict, frz_model=None, ckpt_meta: str | None = None, suffix='', reuse=None)
: Build the model.

  Parameters

  coord_
  : [tf.Tensor] The coordinates of atoms

  atype_
  : [tf.Tensor] The atom types of atoms

  natoms
  : [tf.Tensor] The number of atoms

  box
  : [tf.Tensor] The box vectors

  mesh
  : [tf.Tensor] The mesh vectors

  input_dict
  : [dict] The input dict

  frz_model
  : [str, optional] The path to the frozen model

  ckpt_meta
  : [str, optional] The path prefix of the checkpoint and meta files

  suffix
  : [str, optional] The suffix of the scope

  reuse
  : [bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

  Returns

  dict
  : The output dict

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None

Init the embedding net variables with the given frozen model.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

model_type
[str] the type of the model

suffix
[str] suffix to name scope

enable_mixed_precision(mixed_prec: dict)

Enable mixed precision for the model.

Parameters

mixed_prec
[dict] The mixed precision config

get_numb_fparam() → dict

Get the number of frame parameters.

get_numb_aparam() → dict

Get the number of atomic parameters.

get_numb_dos() → dict

Get the number of gridpoints in energy space.

get_fitting() → dict

Get the fitting(s).

get_loss(loss: dict, lr: dict) → Dict[str, deepmd.tf.loss.loss.Loss]

Get the loss function(s).

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

deepmd.tf.model.pairtab

Module Contents

Classes

PairTabModel	Pairwise tabulation energy model.

class deepmd.tf.model.pairtab.**PairTabModel**(tab_file: str, rcut: float, sel: int | List[int], **kwargs)

Bases: *deepmd.tf.model.model.Model*

Pairwise tabulation energy model.

This model can be used to tabulate the pairwise energy between atoms for either short-range or long-range interactions, such as D3, LJ, ZBL, etc. It should not be used alone, but rather as one submodel of a linear (sum) model, such as DP+D3.

Do not put the model on the first model of a linear model, since the linear model fetches the type map from the first model.

At this moment, the model does not smooth the energy at the cutoff radius, so one needs to make sure the energy has been smoothed to zero.

Parameters

tab_file
[str] The path to the tabulation file.

rcut
[float] The cutoff radius

sel
[int or list[int]] The maxmum number of atoms in the cut-off radius

model_type = 'ener'

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, frz_model: str | None = None, ckpt_meta: str | None = None, suffix: str = '', reuse: bool | enum.Enum | None = None)

Build the model.

Parameters

coord_
[tf.Tensor] The coordinates of atoms

atype_
[tf.Tensor] The atom types of atoms

natoms
[tf.Tensor] The number of atoms

box
[tf.Tensor] The box vectors

mesh
[tf.Tensor] The mesh vectors

input_dict
[dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

dict
The output dict

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None

Init the embedding net variables with the given frozen model.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

model_type
[str] the type of the model

suffix
[str] suffix to name scope

get_fitting() → deepmd.tf.fit.fitting.Fitting | dict

Get the fitting(s).

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss | dict | None

Get the loss function(s).

get_rcut() → float

Get cutoff radius of the model.

get_ntypes() → int

Get the number of types.

data_stat(data: dict)

Data staticis.

enable_compression(suffix: str = '') → None

Enable compression.

Parameters

suffix
[str] suffix to name scope

classmethod update_sel(global_jdata: dict, local_jdata: dict) → dict

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

Returns

dict
The updated local data

Notes

Do not modify the input data without copying it.

deepmd.tf.model.pairwise_dprc

Module Contents

Classes

PairwiseDPRc	Pairwise Deep Potential - Range Correction.

Functions

gather_placeholder(→ deepmd.tf.env.tf.Tensor)	Call tf.gather but allow indices to contain placeholders (-1).

class deepmd.tf.model.pairwise_dprc.PairwiseDPRc(qm_model: dict, qmmm_model: dict, type_embedding: dict | deepmd.tf.utils.type_embed.TypeEmbedNet, type_map: List[str], data_stat_nbatch: int = 10, data_stat_nsample: int = 10, data_stat_protect: float = 0.01, use_srtab: str | None = None, smin_alpha: float | None = None, sw_rmin: float | None = None, sw_rmax: float | None = None, spin: deepmd.tf.utils.spin.Spin | None = None, compress: dict | None = None, **kwargs)

Bases: *deepmd.tf.model.model.Model*

Pairwise Deep Potential - Range Correction.

model_type = 'ener'

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, frz_model=None, ckpt_meta: str | None = None, suffix: str = '', reuse: bool | None = None)

Build the model.

Parameters

coord_
[tf.Tensor] The coordinates of atoms

atype_
[tf.Tensor] The atom types of atoms

natoms
[tf.Tensor] The number of atoms

box
[tf.Tensor] The box vectors

mesh
[tf.Tensor] The mesh vectors

input_dict
[dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

dict
The output dict

get_fitting() → str | dict

Get the fitting(s).

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss | dict

Get the loss function(s).

get_rcut()

Get cutoff radius of the model.

get_ntypes() → int

Get the number of types.

data_stat(data)

Data staticis.

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None

Init the embedding net variables with the given frozen model.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

model_type
[str] the type of the model

suffix
[str] suffix to name scope

enable_compression(suffix: str = '') → None

Enable compression.

Parameters

suffix
[str] suffix to name scope

get_feed_dict(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, **kwargs) → Dict[str, deepmd.tf.env.tf.Tensor]

Generate the feed_dict for current descriptor.

Parameters

coord_
[tf.Tensor] The coordinate of atoms

atype_
[tf.Tensor] The type of atoms

natoms
[tf.Tensor] The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box
[tf.Tensor] The box. Can be generated by deepmd.tf.model.make_stat_input

mesh
[tf.Tensor] For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

aparam
[tf.Tensor] The parameters of the descriptor

**kwargs
[dict] The keyword arguments

Returns

feed_dict
[dict[str, tf.Tensor]] The output feed_dict of current descriptor

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

deepmd.tf.model.pairwise_dprc.**gather_placeholder**(params: deepmd.tf.env.tf.Tensor, indices: deepmd.tf.env.tf.Tensor, placeholder: float = 0.0, **kwargs) → deepmd.tf.env.tf.Tensor

Call tf.gather but allow indices to contain placeholders (-1).

deepmd.tf.model.tensor

Module Contents

Classes

TensorModel	Tensor model.
WFCModel	Tensor model.
DipoleModel	Tensor model.
PolarModel	Tensor model.
GlobalPolarModel	Tensor model.

class deepmd.tf.model.tensor.**TensorModel**(tensor_name: str, descriptor: dict, fitting_net: dict, type_embedding: dict | deepmd.tf.utils.type_embed.TypeEmbedNet | None = None, type_map: List[str] | None = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01, **kwargs)

Bases: *deepmd.tf.model.model.StandardModel*

Tensor model.

Parameters

tensor_name
Name of the tensor.

descriptor
Descriptor

fitting_net
Fitting net

type_embedding
Type embedding net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
Number of frames used for data statistic

data_stat_protect
Protect parameter for atomic energy regression

`get_rcut()`
Get cutoff radius of the model.

`get_ntypes()`
Get the number of types.

`get_type_map()`
Get the type map.

`get_sel_type()`

`get_out_size()`

`data_stat(data)`
Data staticis.

`_compute_input_stat(all_stat, protection=0.01)`

`_compute_output_stat(all_stat)`

`build(coord_, atype_, natoms, box, mesh, input_dict, frz_model=None, ckpt_meta: str | None = None, suffix='', reuse=None)`
Build the model.

Parameters

coord_
[`tf.Tensor`] The coordinates of atoms

atype_
[`tf.Tensor`] The atom types of atoms

natoms
[`tf.Tensor`] The number of atoms

box
[`tf.Tensor`] The box vectors

mesh
[`tf.Tensor`] The mesh vectors

input_dict
[`dict`] The input dict

frz_model
[`str`, `optional`] The path to the frozen model

ckpt_meta
[`str`, `optional`] The path prefix of the checkpoint and meta files

suffix
[`str`, `optional`] The suffix of the scope

reuse
[bool or `tf.AUTO_REUSE`, `optional`] Whether to reuse the variables

Returns

`dict`
The output dict

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None

Init the embedding net variables with the given frozen model.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

model_type
[str] the type of the model

suffix
[str] suffix to name scope

class deepmd.tf.model.tensor.WFCModel(*args, **kwargs)

Bases: *TensorModel*

Tensor model.

Parameters

tensor_name
Name of the tensor.

descriptor
Descriptor

fitting_net
Fitting net

type_embedding
Type embedding net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
Number of frames used for data statistic

data_stat_protect
Protect parameter for atomic energy regression

class deepmd.tf.model.tensor.DipoleModel(*args, **kwargs)

Bases: *TensorModel*

Tensor model.

Parameters

tensor_name
Name of the tensor.

descriptor
Descriptor

fitting_net
Fitting net

type_embedding
: Type embedding net

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
: Number of frames used for data statistic

data_stat_protect
: Protect parameter for atomic energy regression

class deepmd.tf.model.tensor.PolarModel(*args, **kwargs)

Bases: *TensorModel*

Tensor model.

Parameters

tensor_name
: Name of the tensor.

descriptor
: Descriptor

fitting_net
: Fitting net

type_embedding
: Type embedding net

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
: Number of frames used for data statistic

data_stat_protect
: Protect parameter for atomic energy regression

class deepmd.tf.model.tensor.GlobalPolarModel(*args, **kwargs)

Bases: *TensorModel*

Tensor model.

Parameters

tensor_name
: Name of the tensor.

descriptor
: Descriptor

fitting_net
: Fitting net

type_embedding
: Type embedding net

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
: Number of frames used for data statistic

data_stat_protect
: Protect parameter for atomic energy regression

Package Contents

Classes

FrozenModel	Load model from a frozen model, which cannot be trained.
LinearEnergyModel	Linear energy model make linear combinations of several existing energy models.
PairTabModel	Pairwise tabulation energy model.
PairwiseDPRc	Pairwise Deep Potential - Range Correction.
DOSModel	DOS model.
EnerModel	Energy model.
MultiModel	Multi-task model.
DipoleModel	Tensor model.
GlobalPolarModel	Tensor model.
PolarModel	Tensor model.
WFCModel	Tensor model.

class deepmd.tf.model.**FrozenModel**(model_file: str, **kwargs)

Bases: *deepmd.tf.model.model.Model*

Load model from a frozen model, which cannot be trained.

Parameters

model_file
: [str] The path to the frozen model

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, frz_model: str | None = None, ckpt_meta: str | None = None, suffix: str = '', reuse: bool | enum.Enum | None = None) → dict

Build the model.

Parameters

coord_
: [tf.Tensor] The coordinates of atoms

atype_
: [tf.Tensor] The atom types of atoms

natoms
: [tf.Tensor] The number of atoms

box
: [tf.Tensor] The box vectors

mesh
: [tf.Tensor] The mesh vectors

input_dict
[dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

dict
The output dict

get_fitting() → deepmd.tf.fit.fitting.Fitting | dict
Get the fitting(s).

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss | dict | None
Get the loss function(s).

get_rcut()
Get cutoff radius of the model.

get_ntypes() → int
Get the number of types.

data_stat(data)
Data staticis.

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None
Init the embedding net variables with the given frozen model.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

model_type
[str] the type of the model

suffix
[str] suffix to name scope

enable_compression(suffix: str = '') → None
Enable compression.

Parameters

suffix
[str] suffix to name scope

get_type_map() → list

Get the type map.

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

serialize(suffix: str = '') → dict

Serialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Returns

dict
The serialized data

suffix
[str, optional] Name suffix to identify this descriptor

classmethod deserialize(data: dict, suffix: str = '')

Deserialize the model.

There is no suffix in a native DP model, but it is important for the TF backend.

Parameters

data
[dict] The serialized data

suffix
[str, optional] Name suffix to identify this model

Returns

Model
The deserialized Model

class deepmd.tf.model.LinearEnergyModel(models: List[dict], weights: List[float], **kwargs)

Bases: LinearModel

Linear energy model make linear combinations of several existing energy models.

model_type = 'ener'

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, frz_model: str | None = None, ckpt_meta: str | None = None, suffix: str = '', reuse: bool | enum.Enum | None = None) → dict

Build the model.

Parameters

coord_
[tf.Tensor] The coordinates of atoms

atype_
[tf.Tensor] The atom types of atoms

natoms
[tf.Tensor] The number of atoms

box
[tf.Tensor] The box vectors

mesh
[tf.Tensor] The mesh vectors

input_dict
[dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

dict
The output dict

class deepmd.tf.model.PairTabModel(tab_file: str, rcut: float, sel: int | List[int], **kwargs)

Bases: *deepmd.tf.model.model.Model*

Pairwise tabulation energy model.

This model can be used to tabulate the pairwise energy between atoms for either short-range or long-range interactions, such as D3, LJ, ZBL, etc. It should not be used alone, but rather as one submodel of a linear (sum) model, such as DP+D3.

Do not put the model on the first model of a linear model, since the linear model fetches the type map from the first model.

At this moment, the model does not smooth the energy at the cutoff radius, so one needs to make sure the energy has been smoothed to zero.

Parameters

tab_file
[str] The path to the tabulation file.

rcut
[float] The cutoff radius

sel
[int or list[int]] The maxmum number of atoms in the cut-off radius

model_type = 'ener'

build(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, frz_model: str | None = None, ckpt_meta: str | None = None, suffix: str = '', reuse: bool | enum.Enum | None = None)

Build the model.

Parameters

coord_
[tf.Tensor] The coordinates of atoms

atype_
[tf.Tensor] The atom types of atoms

natoms
[tf.Tensor] The number of atoms

box
[tf.Tensor] The box vectors

mesh
[tf.Tensor] The mesh vectors

input_dict
[dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

dict
The output dict

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None

Init the embedding net variables with the given frozen model.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

model_type
[str] the type of the model

suffix
[str] suffix to name scope

get_fitting() → deepmd.tf.fit.fitting.Fitting | dict

Get the fitting(s).

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss | dict | None

Get the loss function(s).

`get_rcut()` → float

Get cutoff radius of the model.

`get_ntypes()` → int

Get the number of types.

`data_stat`(data: dict)

Data staticis.

`enable_compression`(suffix: str = '') → None

Enable compression.

Parameters

suffix
[`str`] suffix to name scope

`classmethod update_sel`(global_jdata: dict, local_jdata: dict) → dict

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[`dict`] The global data, containing the training section

local_jdata
[`dict`] The local data refer to the current class

Returns

`dict`
The updated local data

Notes

Do not modify the input data without copying it.

`class deepmd.tf.model.PairwiseDPRc`(qm_model: dict, qmmm_model: dict, type_embedding: dict | deepmd.tf.utils.type_embed.TypeEmbedNet, type_map: List[str], data_stat_nbatch: int = 10, data_stat_nsample: int = 10, data_stat_protect: float = 0.01, use_srtab: str | None = None, smin_alpha: float | None = None, sw_rmin: float | None = None, sw_rmax: float | None = None, spin: deepmd.tf.utils.spin.Spin | None = None, compress: dict | None = None, **kwargs)

Bases: *deepmd.tf.model.model.Model*

Pairwise Deep Potential - Range Correction.

`model_type = 'ener'`

`build`(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box_: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, input_dict: dict, frz_model=None, ckpt_meta: str | None = None, suffix: str = '', reuse: bool | None = None)

Build the model.

Parameters

coord_
[`tf.Tensor`] The coordinates of atoms

atype_
[tf.Tensor] The atom types of atoms

natoms
[tf.Tensor] The number of atoms

box
[tf.Tensor] The box vectors

mesh
[tf.Tensor] The mesh vectors

input_dict
[dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

dict
The output dict

get_fitting() → str | dict

Get the fitting(s).

get_loss(loss: dict, lr) → deepmd.tf.loss.loss.Loss | dict

Get the loss function(s).

get_rcut()

Get cutoff radius of the model.

get_ntypes() → int

Get the number of types.

data_stat(data)

Data staticis.

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None

Init the embedding net variables with the given frozen model.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

model_type
[str] the type of the model

suffix
[str] suffix to name scope

enable_compression(suffix: str = '') → None

Enable compression.

Parameters

suffix
[str] suffix to name scope

get_feed_dict(coord_: deepmd.tf.env.tf.Tensor, atype_: deepmd.tf.env.tf.Tensor, natoms: deepmd.tf.env.tf.Tensor, box: deepmd.tf.env.tf.Tensor, mesh: deepmd.tf.env.tf.Tensor, **kwargs) → Dict[str, deepmd.tf.env.tf.Tensor]

Generate the feed_dict for current descriptor.

Parameters

coord_
[tf.Tensor] The coordinate of atoms

atype_
[tf.Tensor] The type of atoms

natoms
[tf.Tensor] The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

box
[tf.Tensor] The box. Can be generated by deepmd.tf.model.make_stat_input

mesh
[tf.Tensor] For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

aparam
[tf.Tensor] The parameters of the descriptor

**kwargs
[dict] The keyword arguments

Returns

feed_dict
[dict[str, tf.Tensor]] The output feed_dict of current descriptor

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[dict] The global data, containing the training section

local_jdata
[dict] The local data refer to the current class

class deepmd.tf.model.DOSModel(descriptor: dict, fitting_net: dict, type_embedding: dict | deepmd.tf.utils.type_embed.TypeEmbedNet | None = None, type_map: List[str] | None = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01, **kwargs)

Bases: *deepmd.tf.model.model.StandardModel*

DOS model.

Parameters

descriptor
: Descriptor

fitting_net
: Fitting net

type_embedding
: Type embedding net

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
: Number of frames used for data statistic

data_stat_protect
: Protect parameter for atomic energy regression

model_type = 'dos'

get_numb_dos()

Get the number of gridpoints in energy space.

get_rcut()

Get cutoff radius of the model.

get_ntypes()

Get the number of types.

get_type_map()

Get the type map.

get_numb_fparam() → int

Get the number of frame parameters.

get_numb_aparam() → int

Get the number of atomic parameters.

data_stat(data)

Data staticis.

_compute_input_stat(all_stat, protection=0.01, mixed_type=False)

_compute_output_stat(all_stat, mixed_type=False)

build(coord_, atype_, natoms, box, mesh, input_dict, frz_model=None, ckpt_meta: str | None = None, suffix='', reuse=None)

Build the model.

Parameters

coord_
: [tf.Tensor] The coordinates of atoms

atype_
[tf.Tensor] The atom types of atoms

natoms
[tf.Tensor] The number of atoms

box
[tf.Tensor] The box vectors

mesh
[tf.Tensor] The mesh vectors

input_dict
[dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

dict
The output dict

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None

Init the embedding net variables with the given frozen model.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

model_type
[str] the type of the model

suffix
[str] suffix to name scope

class deepmd.tf.model.EnerModel(descriptor: dict, fitting_net: dict, type_embedding: dict | deepmd.tf.utils.type_embed.TypeEmbedNet | None = None, type_map: List[str] | None = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01, use_srtab: str | None = None, smin_alpha: float | None = None, sw_rmin: float | None = None, sw_rmax: float | None = None, srtab_add_bias: bool = True, spin: deepmd.tf.utils.spin.Spin | None = None, data_bias_nsample: int = 10, **kwargs)

Bases: *deepmd.tf.model.model.StandardModel*

Energy model.

Parameters

descriptor
: Descriptor

fitting_net
: Fitting net

type_embedding
: Type embedding net

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
: Number of frames used for data statistic

data_stat_protect
: Protect parameter for atomic energy regression

use_srtab
: The table for the short-range pairwise interaction added on top of DP. The table is a text data file with (N_t + 1) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

smin_alpha
: The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when use_srtab is provided.

sw_rmin
: The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

sw_rmin
: The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

srtab_add_bias
: [bool] Whether add energy bias from the statistics of the data to short-range tabulated atomic energy. It only takes effect when use_srtab is provided.

spin
: spin

data_stat_nsample
: The number of training samples in a system to compute and change the energy bias.

model_type = 'ener'

get_rcut()

Get cutoff radius of the model.

get_ntypes()

Get the number of types.

get_type_map()

Get the type map.

get_numb_fparam() → int

Get the number of frame parameters.

get_numb_aparam() → int

Get the number of atomic parameters.

data_stat(data)

Data staticis.

_compute_input_stat(all_stat, protection=0.01, mixed_type=False)

_compute_output_stat(all_stat, mixed_type=False)

build(coord_, atype_, natoms, box, mesh, input_dict, frz_model=None, ckpt_meta: str | None = None, suffix='', reuse=None)

Build the model.

Parameters

coord_
[tf.Tensor] The coordinates of atoms

atype_
[tf.Tensor] The atom types of atoms

natoms
[tf.Tensor] The number of atoms

box
[tf.Tensor] The box vectors

mesh
[tf.Tensor] The mesh vectors

input_dict
[dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

dict
The output dict

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None

Init the embedding net variables with the given frozen model.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

model_type
[str] the type of the model

suffix
[str] suffix to name scope

natoms_match(force, natoms)

natoms_not_match(force, natoms, atype)

change_energy_bias(data: deepmd.tf.utils.data_system.DeepmdDataSystem, frozen_model: str, origin_type_map: list, full_type_map: str, bias_adjust_mode: str = 'change-by-statistic') → None

Change the energy bias according to the input data and the pretrained model.

Parameters

data
[DeepmdDataSystem] The training data.

frozen_model
[str] The path file of frozen model.

origin_type_map
[list] The original type_map in dataset, they are targets to change the energy bias.

full_type_map
[str] The full type_map in pretrained model

bias_adjust_mode
[str] The mode for changing energy bias : ['change-by-statistic', 'set-by-statistic']
'change-by-statistic' : perform predictions on energies of target dataset,

and do least sqaure on the errors to obtain the target shift as bias.

'set-by-statistic' : directly use the statistic energy bias in the target dataset.

class deepmd.tf.model.MultiModel(descriptor: dict, fitting_net_dict: dict, fitting_type_dict: dict | None = None, type_embedding=None, type_map: List[str] | None = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01, use_srtab: str | None = None, smin_alpha: float | None = None, sw_rmin: float | None = None, sw_rmax: float | None = None, **kwargs)

Bases: *deepmd.tf.model.model.Model*

Multi-task model.

Parameters

descriptor
Descriptor

fitting_net_dict
Dictionary of fitting nets

fitting_type_dict
deprecated argument

type_embedding
Type embedding net

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
: Number of frames used for data statistic

data_stat_protect
: Protect parameter for atomic energy regression

use_srtab
: The table for the short-range pairwise interaction added on top of DP. The table is a text data file with (N_t + 1) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

smin_alpha
: The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when use_srtab is provided.

sw_rmin
: The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

sw_rmin
: The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

model_type = 'multi_task'

get_rcut()
: Get cutoff radius of the model.

get_ntypes()
: Get the number of types.

get_type_map()
: Get the type map.

data_stat(data)
: Data staticis.

_compute_input_stat(all_stat, protection=0.01, mixed_type=False, fitting_key='')

_compute_output_stat(all_stat, mixed_type=False, fitting_key='')

build(coord_, atype_, natoms, box, mesh, input_dict, frz_model=None, ckpt_meta: str | None = None, suffix='', reuse=None)
: Build the model.

  Parameters

  coord_
  : [tf.Tensor] The coordinates of atoms

  atype_
  : [tf.Tensor] The atom types of atoms

natoms
[tf.Tensor] The number of atoms

box
[tf.Tensor] The box vectors

mesh
[tf.Tensor] The mesh vectors

input_dict
[dict] The input dict

frz_model
[str, optional] The path to the frozen model

ckpt_meta
[str, optional] The path prefix of the checkpoint and meta files

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

dict
The output dict

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, model_type: str = 'original_model', suffix: str = '') → None

Init the embedding net variables with the given frozen model.

Parameters

graph
[tf.Graph] The input frozen model graph

graph_def
[tf.GraphDef] The input frozen model graph_def

model_type
[str] the type of the model

suffix
[str] suffix to name scope

enable_mixed_precision(mixed_prec: dict)

Enable mixed precision for the model.

Parameters

mixed_prec
[dict] The mixed precision config

get_numb_fparam() → dict

Get the number of frame parameters.

get_numb_aparam() → dict

Get the number of atomic parameters.

get_numb_dos() → dict

Get the number of gridpoints in energy space.

`get_fitting()` → dict

Get the fitting(s).

`get_loss`(loss: dict, lr: dict) → Dict[str, deepmd.tf.loss.loss.Loss]

Get the loss function(s).

`classmethod update_sel`(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata
[`dict`] The global data, containing the training section

local_jdata
[`dict`] The local data refer to the current class

`class deepmd.tf.model.DipoleModel`(*args, **kwargs)

Bases: `TensorModel`

Tensor model.

Parameters

tensor_name
Name of the tensor.

descriptor
Descriptor

fitting_net
Fitting net

type_embedding
Type embedding net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
Number of frames used for data statistic

data_stat_protect
Protect parameter for atomic energy regression

`class deepmd.tf.model.GlobalPolarModel`(*args, **kwargs)

Bases: `TensorModel`

Tensor model.

Parameters

tensor_name
Name of the tensor.

descriptor
Descriptor

fitting_net
Fitting net

type_embedding
Type embedding net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
Number of frames used for data statistic

data_stat_protect
Protect parameter for atomic energy regression

class deepmd.tf.model.**PolarModel**(*args, **kwargs)

Bases: TensorModel

Tensor model.

Parameters

tensor_name
Name of the tensor.

descriptor
Descriptor

fitting_net
Fitting net

type_embedding
Type embedding net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
Number of frames used for data statistic

data_stat_protect
Protect parameter for atomic energy regression

class deepmd.tf.model.**WFCModel**(*args, **kwargs)

Bases: TensorModel

Tensor model.

Parameters

tensor_name
Name of the tensor.

descriptor
Descriptor

fitting_net
Fitting net

type_embedding
Type embedding net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
Number of frames used for data statistic

data_stat_protect
: Protect parameter for atomic energy regression

deepmd.tf.nvnmd

Subpackages

deepmd.tf.nvnmd.data

Submodules

deepmd.tf.nvnmd.data.data

Module Contents

deepmd.tf.nvnmd.data.data.jdata_sys

deepmd.tf.nvnmd.data.data.jdata_config_v0

deepmd.tf.nvnmd.data.data.jdata_config_v0_ni128

deepmd.tf.nvnmd.data.data.jdata_config_v0_ni256

deepmd.tf.nvnmd.data.data.jdata_config_v1

deepmd.tf.nvnmd.data.data.jdata_config_v1_ni128

deepmd.tf.nvnmd.data.data.jdata_config_v1_ni256

deepmd.tf.nvnmd.data.data.jdata_deepmd_input_v0

deepmd.tf.nvnmd.data.data.jdata_deepmd_input_v0_ni128

deepmd.tf.nvnmd.data.data.jdata_deepmd_input_v0_ni256

deepmd.tf.nvnmd.data.data.jdata_deepmd_input_v1

deepmd.tf.nvnmd.data.data.jdata_deepmd_input_v1_ni128

deepmd.tf.nvnmd.data.data.jdata_deepmd_input_v1_ni256

deepmd.tf.nvnmd.data.data.NVNMD_WELCOME = (' _ _ __ __ _ _ __ __ ____ ', '| \\ | | \\ \\
/ / | \\ | | | \\/ | | _ \\...

deepmd.tf.nvnmd.data.data.NVNMD_CITATION = ('Please read and cite:', 'Mo et al., npj
Comput Mater 8, 107 (2022)')

deepmd.tf.nvnmd.descriptor

Submodules

deepmd.tf.nvnmd.descriptor.se_a

Module Contents

Functions

build_davg_dstd()	Get the davg and dstd from the dictionary nvnmd_cfg.
check_switch_range(davg, dstd)	Check the range of switch, let it in range [-2, 14].
build_op_descriptor()	Replace se_a.py/DescrptSeA/build.
descrpt2r4(inputs, natoms)	Replace $r_{ji} \to r'_{ji}$
filter_lower_R42GR(type_i, type_input, inputs_i, ...)	Replace se_a.py/DescrptSeA/_filter_lower.
filter_GR2D(xyz_scatter_1)	Replace se_a.py/_filter.

Attributes

log

deepmd.tf.nvnmd.descriptor.se_a.**log**

deepmd.tf.nvnmd.descriptor.se_a.**build_davg_dstd**()

Get the davg and dstd from the dictionary nvnmd_cfg. The davg and dstd have been obtained by training CNN.

deepmd.tf.nvnmd.descriptor.se_a.**check_switch_range**(davg, dstd)

Check the range of switch, let it in range [-2, 14].

deepmd.tf.nvnmd.descriptor.se_a.**build_op_descriptor**()

Replace se_a.py/DescrptSeA/build.

deepmd.tf.nvnmd.descriptor.se_a.**descrpt2r4**(inputs, natoms)

Replace $r_{ji} \to r'_{ji}$ where $r_{ji} = (x_{ji}, y_{ji}, z_{ji})$ and $r'_{ji} = (s_{ji}, \frac{s_{ji}x_{ji}}{r_{ji}}, \frac{s_{ji}y_{ji}}{r_{ji}}, \frac{s_{ji}z_{ji}}{r_{ji}})$.

deepmd.tf.nvnmd.descriptor.se_a.**filter_lower_R42GR**(type_i, type_input, inputs_i, is_exclude, activation_fn, bavg, stddev, trainable, suffix, seed, seed_shift, uniform_seed, filter_neuron, filter_precision, filter_resnet_dt, embedding_net_variables)

Replace se_a.py/DescrptSeA/_filter_lower.

deepmd.tf.nvnmd.descriptor.se_a.**filter_GR2D**(xyz_scatter_1)

Replace se_a.py/_filter.

deepmd.tf.nvnmd.descriptor.se_atten

Module Contents

Functions

build_davg_dstd()	Get the davg and dstd from the dictionary nvnmd_cfg.
check_switch_range(davg, dstd)	Check the range of switch, let it in range [-2, 14].
build_op_descriptor()	Replace se_a.py/DescrptSeA/build.
descrpt2r4(inputs, atype)	Replace $r_{ji} \to r'_{ji}$
filter_lower_R42GR(inputs_i, atype, nei_type_vec)	Replace se_a.py/DescrptSeA/_filter_lower.
filter_GR2D(xyz_scatter_1)	Replace se_a.py/_filter.

Attributes

log

deepmd.tf.nvnmd.descriptor.se_atten.**log**

deepmd.tf.nvnmd.descriptor.se_atten.**build_davg_dstd**()

Get the davg and dstd from the dictionary nvnmd_cfg. The davg and dstd have been obtained by training CNN.

deepmd.tf.nvnmd.descriptor.se_atten.**check_switch_range**(davg, dstd)

Check the range of switch, let it in range [-2, 14].

deepmd.tf.nvnmd.descriptor.se_atten.**build_op_descriptor**()

Replace se_a.py/DescrptSeA/build.

deepmd.tf.nvnmd.descriptor.se_atten.**descrpt2r4**(inputs, atype)

Replace $r_{ji} \to r'_{ji}$ where $r_{ji} = (x_{ji}, y_{ji}, z_{ji})$ and $r'_{ji} = (s_{ji}, \frac{s_{ji}x_{ji}}{r_{ji}}, \frac{s_{ji}y_{ji}}{r_{ji}}, \frac{s_{ji}z_{ji}}{r_{ji}})$.

deepmd.tf.nvnmd.descriptor.se_atten.**filter_lower_R42GR**(inputs_i, atype, nei_type_vec)

Replace se_a.py/DescrptSeA/_filter_lower.

deepmd.tf.nvnmd.descriptor.se_atten.**filter_GR2D**(xyz_scatter_1)

Replace se_a.py/_filter.

deepmd.tf.nvnmd.entrypoints

Submodules

deepmd.tf.nvnmd.entrypoints.freeze

Module Contents

Functions

filter_tensorVariableList(→ dict)	Get the name of variable for NVNMD.
save_weight(sess[, file_name])	Save the dictionary of weight to a npy file.

deepmd.tf.nvnmd.entrypoints.freeze.**filter_tensorVariableList**(tensorVariableList) → dict

Get the name of variable for NVNMD.

```
train_attr/min_nbor_dist
descrpt_attr/t_avg:0
descrpt_attr/t_std:0
type_embed_net/matrix_{layer l}:0
type_embed_net/bias_{layer l}:0
```

version 0: | `filter_type_{atom i}/matrix_{layer l}_{atomj}:0`

```
filter_type_{atom i}/bias_{layer l}_{atomj}:0
layer_{layer l}_type_{atom i}/matrix:0
layer_{layer l}_type_{atom i}/bias:0
final_layer_type_{atom i}/matrix:0
final_layer_type_{atom i}/bias:0
```

version 1: | `filter_type_all/matrix_{layer l}:0`

```
filter_type_all/bias_{layer l}:0
filter_type_all/matrix_{layer l}_two_side_ebd:0
filter_type_all/bias_{layer l}_two_side_ebd:0
layer_{layer l}/matrix:0
layer_{layer l}/bias:0
final_layer/matrix:0
final_layer/bias:0
```

deepmd.tf.nvnmd.entrypoints.freeze.**save_weight**(sess, file_name: str = 'nvnmd/weight.npy')

Save the dictionary of weight to a npy file.

deepmd.tf.nvnmd.entrypoints.mapt

Module Contents

Classes

MapTable	Generate the mapping table describing the relastionship of

Functions

mapt(*[, nvnmd_config, nvnmd_weight, nvnmd_map])	

Attributes

log	

deepmd.tf.nvnmd.entrypoints.mapt.log

class deepmd.tf.nvnmd.entrypoints.mapt.**MapTable**(config_file: str, weight_file: str, map_file: str)

Generate the mapping table describing the relastionship of atomic distance, cutoff function, and embedding matrix.

three mapping table will be built:

$r_{ji}^2 \to s_{ji}$
$r_{ji}^2 \to h_{ji}$
$r_{ji}^2 \to \mathcal{G}_{ji}$

where s_{ji} is cut-off function, $h_{ji} = \frac{s(r_{ji})}{r_{ji}}$, and $\mathcal{G}_{ji}$ is embedding matrix.

The mapping funciton can be define as:

$y = f(x) = y_k + (x - x_k) * dy_k$
$y_k = f(x_k)$
$dy_k = \frac{f(x_{k+1}) - f(x_k)}{dx}$
$x_k \leq x < x_{k+1}$
$x_k = k * dx$

where dx is interpolation interval.

Parameters

config_file
: input file name an .npy file containing the configuration information of NVNMD model

weight_file
: input file name an .npy file containing the weights of NVNMD model

map_file
: output file name an .npy file containing the mapping tables of NVNMD model

References

DOI: 10.1038/s41524-022-00773-z

build_map()

mapping(x, dic_map, cfgs)
: Evaluate value by mapping table operation of tensorflow.

mapping2(x, dic_map, cfgs)
: Evaluate value by mapping table of numpy.

plot_lines(x, dic1, dic2=None)
: Plot lines to see accuracy.

build_map_coef(cfgs, x, ys, grads, grad_grads, Nr, Nc)
: Build mapping table coefficient cfgs: cfg list cfg = x0, x1, dx.

  coef4: a x^3 + b x^2 + c x + d = y: / d = y0 | c = y0' | b = (3 y1 - dx dy' - 2dx y0' - 3y0) / dx^2 a = (dx y1' - 2 y1 + dx y0' + 2 y0) / dx^3

cal_coef4(cfgs, x, y, dy)
: Build mapping table coefficient for one line coef4: a x^3 + b x^2 + c x + d = y: / d = y0 | c = y0' | b = (3 y1 - dx dy' - 2dx y0' - 3y0) / dx^2 a = (dx y1' - 2 y1 + dx y0' + 2 y0) / dx^3.

build_grad(x, y, Nr, Nc)
: : Build gradient of tensor y of x.

build_u2s(r2)
: Build tensor s, s=s(r2).

build_u2s_grad()
: Build gradient of s with respect to u (r^2).

run_u2s()
: Build u->s graph and run it to get value of mapping table.

build_s2g(s)
: Build s->G s is switch function G is embedding net output.

build_s2g_grad()
: Build gradient of G with respect to s.

run_s2g()
: Build s-> graph and run it to get value of mapping table.

build_t2g()

Build t->G t is chemical species of center atom and neighbor atom G is embedding net output of type.

run_t2g()

Build t-> graph and run it to get value of mapping table.

build_embedding_net(xx, wbs, activation_fn=tf.tanh)

build_davg_dstd()

deepmd.tf.nvnmd.entrypoints.mapt.**mapt**(*, nvnmd_config: str | None = 'nvnmd/config.npy', nvnmd_weight: str | None = 'nvnmd/weight.npy', nvnmd_map: str | None = 'nvnmd/map.npy', **kwargs)

deepmd.tf.nvnmd.entrypoints.train

Module Contents

Functions

normalized_input(fn, PATH_CNN, CONFIG_CNN)	Normalize a input script file for continuous neural network.
normalized_input_qnn(jdata, PATH_QNN, CONFIG_CNN, ...)	Normalize a input script file for quantize neural network.
train_nvnmd(*, INPUT, init_model, restart, step[, ...])	

Attributes

log	
jdata_cmd_train	
jdata_cmd_freeze	

deepmd.tf.nvnmd.entrypoints.train.**log**

deepmd.tf.nvnmd.entrypoints.train.**jdata_cmd_train**

deepmd.tf.nvnmd.entrypoints.train.**jdata_cmd_freeze**

deepmd.tf.nvnmd.entrypoints.train.**normalized_input**(fn, PATH_CNN, CONFIG_CNN)

Normalize a input script file for continuous neural network.

deepmd.tf.nvnmd.entrypoints.train.**normalized_input_qnn**(jdata, PATH_QNN, CONFIG_CNN, WEIGHT_CNN, MAP_CNN)

Normalize a input script file for quantize neural network.

deepmd.tf.nvnmd.entrypoints.train.train_nvnmd(*, INPUT: str, init_model: str | None, restart: str | None, step: str, skip_neighbor_stat: bool = False, **kwargs)

deepmd.tf.nvnmd.entrypoints.wrap

Module Contents

Classes

Wrap	Generate the binary model file (model.pb).

Functions

wrap(*[, nvnmd_config, nvnmd_weight, nvnmd_map, ...])	

Attributes

log

deepmd.tf.nvnmd.entrypoints.wrap.log

class deepmd.tf.nvnmd.entrypoints.wrap.Wrap(config_file: str, weight_file: str, map_file: str, model_file: str)

Generate the binary model file (model.pb).

the model file can be use to run the NVNMD with lammps the pair style need set as:

```
pair_style nvnmd model.pb
pair_coeff * *
```

Parameters

config_file
: input file name an .npy file containing the configuration information of NVNMD model

weight_file
: input file name an .npy file containing the weights of NVNMD model

map_file
: input file name an .npy file containing the mapping tables of NVNMD model

model_file
: output file name an .pb file containing the model using in the NVNMD

References

DOI: 10.1038/s41524-022-00773-z

wrap()

wrap_head(nhs, nws)

Wrap the head information.

version nhead nheight nwidth rcut cut-off radius ntype number of atomic species nnei number of neighbors atom_ener atom bias energy

wrap_dscp()

Wrap the configuration of descriptor.

version 0: [NBIT_IDX_S2G-1:0] SHIFT_IDX_S2G

[NBIT_NEIB*NTYPE-1:0] SELs [NBIT_FIXD*M1*NTYPE*NTYPE-1:0] GSs [NBIT_FLTE-1:0] NEXPO_DIV_NI

version 1:

[NBIT_FLTE-1:0] NEXPO_DIV_NI

wrap_fitn()

Wrap the weights of fitting net.

w weight b bias

wrap_bias(bias, NBIT_DATA, NBIT_DATA_FL)

wrap_weight(weight, NBIT_DISP, NBIT_WEIGHT)

weight: weights of fittingNet NBIT_DISP: nbits of exponent of weight max value NBIT_WEIGHT: nbits of mantissa of weights.

wrap_map()

Wrap the mapping table of embedding network.

wrap_lut()

Wrap the LUT.

deepmd.tf.nvnmd.entrypoints.wrap.wrap(*, nvnmd_config: str | None = 'nvnmd/config.npy', nvnmd_weight: str | None = 'nvnmd/weight.npy', nvnmd_map: str | None = 'nvnmd/map.npy', nvnmd_model: str | None = 'nvnmd/model.pb', **kwargs)

Package Contents

Classes

MapTable	Generate the mapping table describing the relastionship of
Wrap	Generate the binary model file (model.pb).

Functions

save_weight(sess[, file_name])	Save the dictionary of weight to a npy file.

deepmd.tf.nvnmd.entrypoints.**save_weight**(sess, file_name: str = 'nvnmd/weight.npy')

Save the dictionary of weight to a npy file.

class deepmd.tf.nvnmd.entrypoints.**MapTable**(config_file: str, weight_file: str, map_file: str)

Generate the mapping table describing the relastionship of atomic distance, cutoff function, and embedding matrix.

three mapping table will be built:

$r_{ji}^2 \to s_{ji}$
$r_{ji}^2 \to h_{ji}$
$r_{ji}^2 \to \mathcal{G}_{ji}$

where s_{ji} is cut-off function, $h_{ji} = \frac{s(r_{ji})}{r_{ji}}$, and $\mathcal{G}_{ji}$ is embedding matrix.

The mapping funciton can be define as:

$y = f(x) = y_k + (x - x_k) * dy_k$
$y_k = f(x_k)$
$dy_k = \frac{f(x_{k+1}) - f(x_k)}{dx}$
$x_k \le x < x_{k+1}$
$x_k = k * dx$

where dx is interpolation interval.

Parameters

config_file
: input file name an .npy file containing the configuration information of NVNMD model

weight_file
: input file name an .npy file containing the weights of NVNMD model

map_file
: output file name an .npy file containing the mapping tables of NVNMD model

References

DOI: 10.1038/s41524-022-00773-z

build_map()

mapping(x, dic_map, cfgs)

Evaluate value by mapping table operation of tensorflow.

mapping2(x, dic_map, cfgs)

Evaluate value by mapping table of numpy.

plot_lines(x, dic1, dic2=None)

Plot lines to see accuracy.

build_map_coef(cfgs, x, ys, grads, grad_grads, Nr, Nc)

Build mapping table coefficient cfgs: cfg list cfg = x0, x1, dx.

coef4: a x^3 + b x^2 + c x + d = y: / d = y0 | c = y0' | b = (3 y1 - dx dy' - 2dx y0' - 3y0) / dx^2 a = (dx y1' - 2 y1 + dx y0' + 2 y0) / dx^3

cal_coef4(cfgs, x, y, dy)

Build mapping table coefficient for one line coef4: a x^3 + b x^2 + c x + d = y: / d = y0 | c = y0' | b = (3 y1 - dx dy' - 2dx y0' - 3y0) / dx^2 a = (dx y1' - 2 y1 + dx y0' + 2 y0) / dx^3.

build_grad(x, y, Nr, Nc)

: Build gradient of tensor y of x.

build_u2s(r2)

Build tensor s, s=s(r2).

build_u2s_grad()

Build gradient of s with respect to u (r^2).

run_u2s()

Build u->s graph and run it to get value of mapping table.

build_s2g(s)

Build s->G s is switch function G is embedding net output.

build_s2g_grad()

Build gradient of G with respect to s.

run_s2g()

Build s-> graph and run it to get value of mapping table.

build_t2g()

Build t->G t is chemical species of center atom and neighbor atom G is embedding net output of type.

run_t2g()

Build t-> graph and run it to get value of mapping table.

build_embedding_net(xx, wbs, activation_fn=tf.tanh)

build_davg_dstd()

class deepmd.tf.nvnmd.entrypoints.Wrap(config_file: str, weight_file: str, map_file: str, model_file: str)

Generate the binary model file (model.pb).

the model file can be use to run the NVNMD with lammps the pair style need set as:

```
pair_style nvnmd model.pb
pair_coeff * *
```

Parameters

config_file
: input file name an .npy file containing the configuration information of NVNMD model

weight_file
: input file name an .npy file containing the weights of NVNMD model

map_file
: input file name an .npy file containing the mapping tables of NVNMD model

model_file
: output file name an .pb file containing the model using in the NVNMD

References

DOI: 10.1038/s41524-022-00773-z

wrap()

wrap_head(nhs, nws)

Wrap the head information.

version nhead nheight nwidth rcut cut-off radius ntype number of atomic species nnei number of neighbors atom_ener atom bias energy

wrap_dscp()

Wrap the configuration of descriptor.

version 0: [NBIT_IDX_S2G-1:0] SHIFT_IDX_S2G

[NBIT_NEIB*NTYPE-1:0] SELs [NBIT_FIXD*M1*NTYPE*NTYPE-1:0] GSs [NBIT_FLTE-1:0] NEXPO_DIV_NI

version 1:

[NBIT_FLTE-1:0] NEXPO_DIV_NI

wrap_fitn()

Wrap the weights of fitting net.

w weight b bias

wrap_bias(bias, NBIT_DATA, NBIT_DATA_FL)

wrap_weight(weight, NBIT_DISP, NBIT_WEIGHT)

weight: weights of fittingNet NBIT_DISP: nbits of exponent of weight max value NBIT_WEIGHT: nbits of mantissa of weights.

wrap_map()

Wrap the mapping table of embedding network.

wrap_lut()
Wrap the LUT.

deepmd.tf.nvnmd.fit

Submodules

deepmd.tf.nvnmd.fit.ener

Module Contents

Functions

one_layer_nvnmd(inputs, outputs_size[, activation_fn, ...])	Build one layer with continuous or quantized value.

Attributes

GLOBAL_TF_FLOAT_PRECISION
nvnmd_cfg

deepmd.tf.nvnmd.fit.ener.**GLOBAL_TF_FLOAT_PRECISION**

deepmd.tf.nvnmd.fit.ener.**nvnmd_cfg**

deepmd.tf.nvnmd.fit.ener.**one_layer_nvnmd**(inputs, outputs_size, activation_fn=tf.nn.tanh, precision=GLOBAL_TF_FLOAT_PRECISION, stddev=1.0, bavg=0.0, name='linear', reuse=None, seed=None, use_timestep=False, trainable=True, useBN=False, uniform_seed=False, initial_variables=None, mixed_prec=None, final_layer=False)

Build one layer with continuous or quantized value. Its weight and bias can be initialed with random or constant value.

deepmd.tf.nvnmd.utils

Submodules

deepmd.tf.nvnmd.utils.argcheck

Alias for backward compatibility.

Module Contents

Functions

nvnmd_args()	

deepmd.tf.nvnmd.utils.argcheck.**nvnmd_args**()

deepmd.tf.nvnmd.utils.config

Module Contents

Classes

NvnmdConfig	Configuration for NVNMD

Attributes

log	
nvnmd_cfg	

deepmd.tf.nvnmd.utils.config.**log**

class deepmd.tf.nvnmd.utils.config.**NvnmdConfig**(jdata: dict)

Configuration for NVNMD record the message of model such as size, using nvnmd or not.

Parameters

jdata
: a dictionary of input script

References

DOI: 10.1038/s41524-022-00773-z

init_from_jdata(jdata: dict = {})

Initialize this class with jdata loaded from input script.

init_value()

Initialize member with dict.

update_config()

Update config from dict.

init_train_mode(mod='cnn')
Configure for taining cnn or qnn.

init_from_config(jdata)
Initialize member element one by one.

init_config_by_version(version, max_nnei)
Initialize version-dependent parameters.

init_net_size()
Initialize net_size.

init_from_deepmd_input(jdata)
Initialize members with input script of deepmd.

init_dscp(jdata: dict, jdata_parent: dict = {}) → dict
Initialize members about descriptor.

init_fitn(jdata: dict, jdata_parent: dict = {}) → dict
Initialize members about fitting network.

init_dpin(jdata: dict, jdata_parent: dict = {}) → dict
Initialize members about other deepmd input.

init_size(jdata: dict, jdata_parent: dict = {}) → dict
Initialize members about ram capacity.

init_ctrl(jdata: dict, jdata_parent: dict = {}) → dict
Initialize members about control signal.

init_nbit(jdata: dict, jdata_parent: dict = {}) → dict
Initialize members about quantification precision.

save(file_name=None)
Save all configuration to file.

set_ntype(ntype)
Set the number of type.

get_s_range(davg, dstd)
Get the range of switch function.

get_dscp_jdata()
Generate model/descriptor in input script.

get_fitn_jdata()
Generate model/fitting_net in input script.

get_model_jdata()
Generate model in input script.

get_nvnmd_jdata()
Generate nvnmd in input script.

get_learning_rate_jdata()
Generate learning_rate in input script.

get_loss_jdata()
Generate loss in input script.

get_training_jdata()
Generate training in input script.

get_deepmd_jdata()
Generate input script with member element one by one.

get_dp_init_weights()
Build the weight dict for initialization of net.

disp_message()
Display the log of NVNMD.

deepmd.tf.nvnmd.utils.config.**nvnmd_cfg**

deepmd.tf.nvnmd.utils.encode

Module Contents

Classes

Encode	Encoding value as hex, bin, and dec format.

Attributes

log

deepmd.tf.nvnmd.utils.encode.**log**

class deepmd.tf.nvnmd.utils.encode.**Encode**
Encoding value as hex, bin, and dec format.

qr(v, nbit: int = 14)
Quantize value using round.

qf(v, nbit: int = 14)
Quantize value using floor.

qc(v, nbit: int = 14)
Quantize value using ceil.

split_expo_mant(v, min=-1000)

find_max_expo(v, expo_min=-1000)

norm_expo(v, nbit_frac=20, expo_min=-1000)

flt2bin_one(v, nbit_expo, nbit_frac)

flt2bin(data, nbit_expo, nbit_frac)

Convert float into binary string list.

byte2hex(bs, nbyte)

Convert byte into hex bs: low byte in the first hex: low byte in the right.

check_dec(idec, nbit, signed=False, name='')

Check whether the data (idec) is in the range range is $[0, 2^n bit - 1]$ for unsigned range is $[-2^{nbit-1}, 2^{nbit-1} - 1]$ for signed.

extend_list(slbin, nfull)

Extend the list (slbin) to the length (nfull) the attched element of list is 0.

such as, when

slbin = ['10010','10100'],
nfull = 4

extent it to

['10010','10100','00000','00000']

extend_bin(slbin, nfull)

Extend the element of list (slbin) to the length (nfull).

such as, when

slbin = ['10010','10100'],
nfull = 6

extent to

['010010','010100']

extend_hex(slhex, nfull)

Extend the element of list (slhex) to the length (nfull).

split_bin(sbin, nbit: int)

Split sbin into many segment with the length nbit.

reverse_bin(slbin, nreverse)

Reverse binary string list per nreverse value.

merge_bin(slbin, nmerge)

Merge binary string list per nmerge value.

dec2bin(idec, nbit=10, signed=False, name='')

Convert dec array to binary string list.

hex2bin_str(shex)

Convert hex string to binary string.

hex2bin(data)

Convert hex string list to binary string list.

bin2hex_str(sbin)

Convert binary string to hex string.

bin2hex(data)

Convert binary string list to hex string list.

deepmd.tf.nvnmd.utils.fio

Module Contents

Classes

Fio	Basic class for FIO.
FioDic	Input and output for dict class data
FioNpyDic	Input and output for .npy file containing dictionary.
FioJsonDic	Input and output for .json file containing dictionary.
FioBin	Input and output for binary file.
FioTxt	Input and output for .txt file with string.

Attributes

log

deepmd.tf.nvnmd.utils.fio.**log**

class deepmd.tf.nvnmd.utils.fio.**Fio**

Basic class for FIO.

exits(file_name='')

mkdir(path_name='')

create_file_path(file_name='')

is_path(path)

is_file(file_name)

get_file_list(path) → list

class deepmd.tf.nvnmd.utils.fio.**FioDic**

Input and output for dict class data the file can be .json or .npy file containing a dictionary.

load(file_name='', default_value={})

save(file_name='', dic={})

get(jdata, key, default_value)

update(jdata, jdata_o)

Update key-value pair is key in jdata_o.keys().

Parameters

jdata
: new jdata

jdata_o
: origin jdata

class deepmd.tf.nvnmd.utils.fio.**FioNpyDic**

Input and output for .npy file containing dictionary.

load(file_name='', default_value={})

save(file_name='', dic={})

class deepmd.tf.nvnmd.utils.fio.**FioJsonDic**

Input and output for .json file containing dictionary.

load(file_name='', default_value={})

Load .json file into dict.

save(file_name='', dic={})

Save dict into .json file.

class deepmd.tf.nvnmd.utils.fio.**FioBin**

Input and output for binary file.

load(file_name='', default_value='')

Load binary file into bytes value.

save(file_name: str, data: List[str])

Save hex string into binary file.

class deepmd.tf.nvnmd.utils.fio.**FioTxt**

Input and output for .txt file with string.

load(file_name='', default_value=[])

Load .txt file into string list.

save(file_name: str = '', data: list = [])

Save string list into .txt file.

deepmd.tf.nvnmd.utils.network

Module Contents

Functions

get_sess()	
matmul2_qq(a, b, nbit)	Quantized matmul operation for 2d tensor.
matmul3_qq(a, b, nbit)	Quantized matmul operation for 3d tensor.
qf(x, nbit)	Quantize and floor tensor x with quantification precision nbit.
qr(x, nbit)	Quantize and round tensor x with quantification precision nbit.
tanh4(x)	
one_layer_wb(shape, outputs_size, bavg, stddev, ...)	
one_layer_t(shape, outputs_size, bavg, stddev, ...)	
one_layer(inputs, outputs_size[, activation_fn, ...])	Build one layer with continuous or quantized value.

Attributes

log

deepmd.tf.nvnmd.utils.network.**log**

deepmd.tf.nvnmd.utils.network.**get_sess**()

deepmd.tf.nvnmd.utils.network.**matmul2_qq**(a, b, nbit)

Quantized matmul operation for 2d tensor. a and b is input tensor, nbit represent quantification precision.

deepmd.tf.nvnmd.utils.network.**matmul3_qq**(a, b, nbit)

Quantized matmul operation for 3d tensor. a and b is input tensor, nbit represent quantification precision.

deepmd.tf.nvnmd.utils.network.**qf**(x, nbit)

Quantize and floor tensor x with quantification precision nbit.

deepmd.tf.nvnmd.utils.network.**qr**(x, nbit)

Quantize and round tensor x with quantification precision nbit.

deepmd.tf.nvnmd.utils.network.**tanh4**(x)

deepmd.tf.nvnmd.utils.network.**one_layer_wb**(shape, outputs_size, bavg, stddev, precision, trainable, initial_variables, seed, uniform_seed, name)

deepmd.tf.nvnmd.utils.network.**one_layer_t**(shape, outputs_size, bavg, stddev, precision, trainable, initial_variables, seed, uniform_seed, name)

deepmd.tf.nvnmd.utils.network.**one_layer**(inputs, outputs_size, activation_fn=tf.nn.tanh, precision=GLOBAL_TF_FLOAT_PRECISION, stddev=1.0, bavg=0.0, name='linear', reuse=None, seed=None, use_timestep=False, trainable=True, useBN=False, uniform_seed=False, initial_variables=None, mixed_prec=None, final_layer=False)

Build one layer with continuous or quantized value. Its weight and bias can be initialed with random or constant value.

deepmd.tf.nvnmd.utils.op

Module Contents

Functions

r2s(r, rmin, rmax)	
map_nvnmd(x, map_y, map_dy, prec[, nbit])	Mapping function implemented by numpy.

deepmd.tf.nvnmd.utils.op.**r2s**(r, rmin, rmax)

deepmd.tf.nvnmd.utils.op.**map_nvnmd**(x, map_y, map_dy, prec, nbit=None)

Mapping function implemented by numpy.

deepmd.tf.nvnmd.utils.weight

Module Contents

Functions

get_weight(weights, key)	Get weight value according to key.
get_normalize(weights)	Get normalize parameter (avg and std) of s_{ji}.
get_type_embedding_weight(weights, layer_l)	Get weight and bias of type_embedding network.
get_filter_weight(weights, spe_j, layer_l)	Get weight and bias of embedding network.
get_filter_type_weight(weights, layer_l)	Get weight and bias of two_side_type_embedding network.
get_fitnet_weight(weights, spe_i, layer_l[, nlayer])	Get weight and bias of fitting network.
get_type_weight(weights, layer_l)	Get weight and bias of fitting network.
get_constant_initializer(weights, name)	Get initial value by name and create a initializer.

Attributes

log

deepmd.tf.nvnmd.utils.weight.**log**

deepmd.tf.nvnmd.utils.weight.**get_weight**(weights, key)

Get weight value according to key.

deepmd.tf.nvnmd.utils.weight.**get_normalize**(weights: dict)

Get normalize parameter (avg and std) of s_{ji}.

deepmd.tf.nvnmd.utils.weight.**get_type_embedding_weight**(weights: dict, layer_l: int)

Get weight and bias of type_embedding network.

Parameters

weights
[dict] weights

layer_l
layer order in embedding network 1~nlayer

deepmd.tf.nvnmd.utils.weight.**get_filter_weight**(weights: int, spe_j: int, layer_l: int)

Get weight and bias of embedding network.

Parameters

weights
[dict] weights

spe_j
[int] special order of neighbor atom j 0~ntype-1

layer_l
layer order in embedding network 1~nlayer

deepmd.tf.nvnmd.utils.weight.**get_filter_type_weight**(weights: dict, layer_l: int)

Get weight and bias of two_side_type_embedding network.

Parameters

weights
[dict] weights

layer_l
layer order in embedding network 1~nlayer

deepmd.tf.nvnmd.utils.weight.**get_fitnet_weight**(weights: dict, spe_i: int, layer_l: int, nlayer: int = 10)

Get weight and bias of fitting network.

Parameters

weights
[dict] weights

spe_i
[int] special order of central atom i 0~ntype-1

layer_l
[int] layer order in embedding network 0~nlayer-1

nlayer
[int] number of layers

deepmd.tf.nvnmd.utils.weight.**get_type_weight**(weights: dict, layer_l: int)

Get weight and bias of fitting network.

Parameters

weights
[dict] weights

layer_l
[int] layer order in embedding network 0~nlayer-1

deepmd.tf.nvnmd.utils.weight.**get_constant_initializer**(weights, name)

Get initial value by name and create a initializer.

Package Contents

Classes

Encode	Encoding value as hex, bin, and dec format.
FioBin	Input and output for binary file.
FioDic	Input and output for dict class data
FioTxt	Input and output for .txt file with string.

Functions

nvnmd_args()	
one_layer(inputs, outputs_size[, activation_fn, ...])	Build one layer with continuous or quantized value.
map_nvnmd(x, map_y, map_dy, prec[, nbit])	Mapping function implemented by numpy.
get_filter_weight(weights, spe_j, layer_l)	Get weight and bias of embedding network.
get_fitnet_weight(weights, spe_i, layer_l[, nlayer])	Get weight and bias of fitting network.

Attributes

nvnmd_cfg

deepmd.tf.nvnmd.utils.**nvnmd_args**()

deepmd.tf.nvnmd.utils.**nvnmd_cfg**

class deepmd.tf.nvnmd.utils.Encode

Encoding value as hex, bin, and dec format.

qr(v, nbit: int = 14)

Quantize value using round.

qf(v, nbit: int = 14)

Quantize value using floor.

qc(v, nbit: int = 14)

Quantize value using ceil.

split_expo_mant(v, min=-1000)

find_max_expo(v, expo_min=-1000)

norm_expo(v, nbit_frac=20, expo_min=-1000)

flt2bin_one(v, nbit_expo, nbit_frac)

flt2bin(data, nbit_expo, nbit_frac)

Convert float into binary string list.

byte2hex(bs, nbyte)

Convert byte into hex bs: low byte in the first hex: low byte in the right.

check_dec(idec, nbit, signed=False, name='')

Check whether the data (idec) is in the range range is $[0, 2^n bit - 1]$ for unsigned range is $[-2^{nbit-1}, 2^{nbit-1} - 1]$ for signed.

extend_list(slbin, nfull)

Extend the list (slbin) to the length (nfull) the attched element of list is 0.

such as, when

slbin = ['10010','10100'],
nfull = 4

extent it to

['10010','10100','00000','00000]

extend_bin(slbin, nfull)

Extend the element of list (slbin) to the length (nfull).

such as, when

slbin = ['10010','10100'],
nfull = 6

extent to

['010010','010100']

extend_hex(slhex, nfull)

Extend the element of list (slhex) to the length (nfull).

split_bin(sbin, nbit: int)

Split sbin into many segment with the length nbit.

reverse_bin(slbin, nreverse)

Reverse binary string list per nreverse value.

merge_bin(slbin, nmerge)

Merge binary string list per nmerge value.

dec2bin(idec, nbit=10, signed=False, name='')

Convert dec array to binary string list.

hex2bin_str(shex)

Convert hex string to binary string.

hex2bin(data)

Convert hex string list to binary string list.

bin2hex_str(sbin)

Convert binary string to hex string.

bin2hex(data)

Convert binary string list to hex string list.

class deepmd.tf.nvnmd.utils.FioBin

Input and output for binary file.

load(file_name='', default_value='')

Load binary file into bytes value.

save(file_name: str, data: List[str])

Save hex string into binary file.

class deepmd.tf.nvnmd.utils.FioDic

Input and output for dict class data the file can be .json or .npy file containing a dictionary.

load(file_name='', default_value={})

save(file_name='', dic={})

get(jdata, key, default_value)

update(jdata, jdata_o)

Update key-value pair is key in jdata_o.keys().

Parameters

jdata
: new jdata

jdata_o
: origin jdata

class deepmd.tf.nvnmd.utils.FioTxt

Input and output for .txt file with string.

load(file_name='', default_value=[])
Load .txt file into string list.

save(file_name: str = '', data: list = [])
Save string list into .txt file.

deepmd.tf.nvnmd.utils.**one_layer**(inputs, outputs_size, activation_fn=tf.nn.tanh, precision=GLOBAL_TF_FLOAT_PRECISION, stddev=1.0, bavg=0.0, name='linear', reuse=None, seed=None, use_timestep=False, trainable=True, useBN=False, uniform_seed=False, initial_variables=None, mixed_prec=None, final_layer=False)
Build one layer with continuous or quantized value. Its weight and bias can be initialed with random or constant value.

deepmd.tf.nvnmd.utils.**map_nvnmd**(x, map_y, map_dy, prec, nbit=None)
Mapping function implemented by numpy.

deepmd.tf.nvnmd.utils.**get_filter_weight**(weights: int, spe_j: int, layer_l: int)
Get weight and bias of embedding network.

Parameters

weights
[dict] weights

spe_j
[int] special order of neighbor atom j 0~ntype-1

layer_l
layer order in embedding network 1~nlayer

deepmd.tf.nvnmd.utils.**get_fitnet_weight**(weights: dict, spe_i: int, layer_l: int, nlayer: int = 10)
Get weight and bias of fitting network.

Parameters

weights
[dict] weights

spe_i
[int] special order of central atom i 0~ntype-1

layer_l
[int] layer order in embedding network 0~nlayer-1

nlayer
[int] number of layers

deepmd.tf.op

This module will house cust Tf OPs after CMake installation.

Submodules

deepmd.tf.op._add_flt_nvnmd_grad

Module Contents

Functions

_AddFltNvnmdGrad(op, grad)

deepmd.tf.op._add_flt_nvnmd_grad.**_AddFltNvnmdGrad**(op, grad)

deepmd.tf.op._copy_flt_nvnmd_grad

Module Contents

Functions

_CpoyFltNvnmdGrad(op, grad1, grad2)

deepmd.tf.op._copy_flt_nvnmd_grad.**_CpoyFltNvnmdGrad**(op, grad1, grad2)

deepmd.tf.op._dotmul_flt_nvnmd_grad

Module Contents

Functions

_DotmulFltNvnmdGrad(op, grad)

deepmd.tf.op._dotmul_flt_nvnmd_grad.**_DotmulFltNvnmdGrad**(op, grad)

deepmd.tf.op._flt_nvnmd_grad

Module Contents

Functions

_FltNvnmdGrad(op, grad)	

deepmd.tf.op._flt_nvnmd_grad.**_FltNvnmdGrad**(op, grad)

deepmd.tf.op._gelu

First-order derivatives and second-order derivatives for gelu function.

Module Contents

Functions

_gelu_custom_cc(op, dy)	
_gelu_grad_custom_cc(op, dy)	

Attributes

gelu	

deepmd.tf.op._gelu.**gelu**

deepmd.tf.op._gelu.**_gelu_custom_cc**(op, dy)

deepmd.tf.op._gelu.**_gelu_grad_custom_cc**(op, dy)

deepmd.tf.op._map_flt_nvnmd_grad

Module Contents

Functions

_MapFltNvnmdGrad(op, grad)

deepmd.tf.op._map_flt_nvnmd_grad.**_MapFltNvnmdGrad**(op, grad)

deepmd.tf.op._matmul_fitnet_nvnmd_grad

Module Contents

Functions

_MatmulFitnetNvnmdGrad(op, grad)

deepmd.tf.op._matmul_fitnet_nvnmd_grad.**_MatmulFitnetNvnmdGrad**(op, grad)

deepmd.tf.op._matmul_flt2fix_nvnmd

Module Contents

Functions

_MatmulFlt2fixNvnmdGrad(op, grad)

deepmd.tf.op._matmul_flt2fix_nvnmd.**_MatmulFlt2fixNvnmdGrad**(op, grad)

deepmd.tf.op._matmul_flt_nvnmd_grad

Module Contents

Functions

_MatmulFltNvnmdGrad(op, grad)

deepmd.tf.op._matmul_flt_nvnmd_grad.**_MatmulFltNvnmdGrad**(op, grad)

deepmd.tf.op._mul_flt_nvnmd_grad

Module Contents

Functions

_MulFltNvnmdGrad(op, grad)

deepmd.tf.op._mul_flt_nvnmd_grad.**_MulFltNvnmdGrad**(op, grad)

deepmd.tf.op._prod_force_grad

Gradients for prod force.

Module Contents

Functions

_prod_force_grad_cc(op, grad)

deepmd.tf.op._prod_force_grad.**_prod_force_grad_cc**(op, grad)

deepmd.tf.op._prod_force_se_a_grad

Gradients for prod force.

Module Contents

Functions

_prod_force_se_a_grad_cc(op, grad)

deepmd.tf.op._prod_force_se_a_grad.**_prod_force_se_a_grad_cc**(op, grad)

deepmd.tf.op._prod_force_se_a_mask_grad

Gradients for prod force se_a_mask.

Module Contents

Functions

_prod_force_se_a_mask_grad_cc(op, grad)

deepmd.tf.op._prod_force_se_a_mask_grad.**_prod_force_se_a_mask_grad_cc**(op, grad)

deepmd.tf.op._prod_force_se_r_grad

Gradients for prod force.

Module Contents

Functions

_prod_force_se_a_grad_cc(op, grad)

deepmd.tf.op._prod_force_se_r_grad.**_prod_force_se_a_grad_cc**(op, grad)

deepmd.tf.op._prod_virial_grad

Gradients for prod virial.

Module Contents

Functions

_prod_virial_grad_cc(op, grad, grad_atom)

deepmd.tf.op._prod_virial_grad.**_prod_virial_grad_cc**(op, grad, grad_atom)

deepmd.tf.op._prod_virial_se_a_grad

Gradients for prod virial.

Module Contents

Functions

_prod_virial_se_a_grad_cc(op, grad, grad_atom)	

deepmd.tf.op._prod_virial_se_a_grad.**_prod_virial_se_a_grad_cc**(op, grad, grad_atom)

deepmd.tf.op._prod_virial_se_r_grad

Gradients for prod virial.

Module Contents

Functions

_prod_virial_se_a_grad_cc(op, grad, grad_atom)	

deepmd.tf.op._prod_virial_se_r_grad.**_prod_virial_se_a_grad_cc**(op, grad, grad_atom)

deepmd.tf.op._quantize_nvnmd_grad

Module Contents

Functions

_QuantizeNvnmdGrad(op, grad)	

deepmd.tf.op._quantize_nvnmd_grad.**_QuantizeNvnmdGrad**(op, grad)

deepmd.tf.op._soft_min_force_grad

Gradients for soft min force.

Module Contents

Functions

_soft_min_force_grad_cc(op, grad)

deepmd.tf.op._soft_min_force_grad.**_soft_min_force_grad_cc**(op, grad)

deepmd.tf.op._soft_min_virial_grad

Gradients for soft min virial.

Module Contents

Functions

_soft_min_virial_grad_cc(op, grad, grad_atom)

deepmd.tf.op._soft_min_virial_grad.**_soft_min_virial_grad_cc**(op, grad, grad_atom)

deepmd.tf.op._tabulate_grad

Gradients for tabulate.

Module Contents

Functions

_tabulate_fusion_se_a_grad_cc(op, dy)	
_tabulate_fusion_se_a_grad_grad_cc(op, dy, dy_)	
_tabulate_fusion_se_atten_grad_cc(op, dy)	
_tabulate_fusion_se_atten_grad_grad_cc(op, dy, dy_, ...)	
_tabulate_fusion_se_t_grad_cc(op, dy)	
_tabulate_fusion_se_t_grad_grad_cc(op, dy, dy_)	
_tabulate_fusion_se_r_grad_cc(op, dy)	
_tabulate_fusion_se_r_grad_grad_cc(op, dy)	

deepmd.tf.op._tabulate_grad.**_tabulate_fusion_se_a_grad_cc**(op, dy)

deepmd.tf.op._tabulate_grad.**_tabulate_fusion_se_a_grad_grad_cc**(op, dy, dy_)

deepmd.tf.op._tabulate_grad.**_tabulate_fusion_se_atten_grad_cc**(op, dy)

deepmd.tf.op._tabulate_grad.**_tabulate_fusion_se_atten_grad_grad_cc**(op, dy, dy_, dy_dtwo)

deepmd.tf.op._tabulate_grad.**_tabulate_fusion_se_t_grad_cc**(op, dy)

deepmd.tf.op._tabulate_grad.**_tabulate_fusion_se_t_grad_grad_cc**(op, dy, dy_)

deepmd.tf.op._tabulate_grad.**_tabulate_fusion_se_r_grad_cc**(op, dy)

deepmd.tf.op._tabulate_grad.**_tabulate_fusion_se_r_grad_grad_cc**(op, dy)

deepmd.tf.op._tanh4_flt_nvnmd_grad

Module Contents

Functions

_Tanh4FltNvnmdGrad(op, grad)	

deepmd.tf.op._tanh4_flt_nvnmd_grad.**_Tanh4FltNvnmdGrad**(op, grad)

Package Contents

Functions

import_ops()	Import all custom TF ops that are present in this submodule.

Attributes

NOT_LOADABLE
PACKAGE_BASE
log

deepmd.tf.op.**NOT_LOADABLE** = ('__init__.py',)

deepmd.tf.op.**PACKAGE_BASE** = 'deepmd.tf.op'

deepmd.tf.op.**log**

deepmd.tf.op.**import_ops**()

Import all custom TF ops that are present in this submodule.

Notes

Initialy this subdir is unpopulated. CMake will install all the op module python files and shared libs.

deepmd.tf.train

Submodules

deepmd.tf.train.run_options

Module taking care of important package constants.

Module Contents

Classes

RunOptions	Class with info on how to run training (cluster, MPI and GPU config).

class deepmd.tf.train.run_options.RunOptions(init_model: str | None = None, init_frz_model: str | None = None, finetune: str | None = None, restart: str | None = None, log_path: str | None = None, log_level: int = 0, mpi_log: str = 'master')

Class with info on how to run training (cluster, MPI and GPU config).

Attributes

gpus: Optional[List[int]]
list of GPUs if any are present else None

is_chief: bool
in distribured training it is true for tha main MPI process in serail it is always true

world_size: int
total worker count

my_rank: int
index of the MPI task

nodename: str
name of the node

node_list_
[List[str]] the list of nodes of the current mpirun

my_device: str
deviice type - gpu or cpu

property is_chief

Whether my rank is 0.

gpus: List[int] | None

world_size: int

my_rank: int

nodename: str

nodelist: List[int]

my_device: str

_HVD: horovod.tensorflow | None

_log_handles_already_set: bool = False

print_resource_summary()

Print build and current running cluster configuration summary.

_setup_logger(log_path: pathlib.Path | None, log_level: int, mpi_log: str | None)

Set up package loggers.

Parameters

log_level
[int] logging level

log_path
[Optional[str]] path to log file, if None logs will be send only to console. If the parent directory does not exist it will be automatically created, by default None

mpi_log
[Optional[str], optional] mpi log type. Has three options. master will output logs to file and console only from rank==0. collect will write messages from all ranks to one file opened under rank==0 and to console. workers will open one log file for each worker designated by its rank, console behaviour is the same as for collect.

_try_init_distrib()

_init_distributed(HVD: RunOptions._init_distributed.HVD)
Initialize settings for distributed training.

Parameters

HVD
[HVD] horovod object

_init_serial()
Initialize setting for serial training.

deepmd.tf.train.trainer

Module Contents

Classes

DPTrainer	
DatasetLoader	Generate an OP that loads the training data from the given DeepmdDataSystem.

Functions

_is_subdir(path, directory)

Attributes

log

deepmd.tf.train.trainer.**log**

deepmd.tf.train.trainer.**_is_subdir**(path, directory)

class deepmd.tf.train.trainer.**DPTrainer**(jdata, run_opt, is_compress=False)

_init_param(jdata)

build(data=None, stop_batch=0, origin_type_map=None, suffix='')

_build_lr()

_build_loss()

_build_network(data, suffix='')

_build_optimizer(fitting_key=None)

_build_training()

_init_session()

train(train_data=None, valid_data=None)

save_checkpoint(cur_batch: int)

get_feed_dict(batch, is_training)

get_global_step()

valid_on_the_fly(fp, train_batches, valid_batches, print_header=False, fitting_key=None)

static print_header(fp, train_results, valid_results, multi_task_mode=False)

static print_on_training(fp, train_results, valid_results, cur_batch, cur_lr,
multi_task_mode=False, cur_lr_dict=None)

static eval_single_list(single_batch_list, loss, sess, get_feed_dict_func, prefix='')

get_evaluation_results(batch_list)

save_compressed()

Save the compressed graph.

_get_place_holders(data_dict)

_init_from_frz_model()

_init_from_ckpt(ckpt_meta: str)

_init_from_pretrained_model(data, origin_type_map=None,
bias_adjust_mode='change-by-statistic')

Init the embedding net variables with the given frozen model.

Parameters

data
[`DeepmdDataSystem`] The training data.

origin_type_map
[`list`] The original type_map in dataset, they are targets to change the energy bias.

bias_adjust_mode
[`str`] The mode for changing energy bias : ['change-by-statistic', 'set-by-statistic']
'change-by-statistic' : perform predictions on energies of target dataset,

and do least sqaure on the errors to obtain the target shift as bias.

'set-by-statistic' : directly use the statistic energy bias in the target dataset.

_change_energy_bias(data, frozen_model, origin_type_map, bias_adjust_mode='change-by-statistic')

class deepmd.tf.train.trainer.DatasetLoader(train_data: deepmd.tf.utils.data_system.DeepmdDataSystem)

Generate an OP that loads the training data from the given DeepmdDataSystem.

It can be used to load the training data in the training process, so there is no waiting time between training steps.

Parameters

train_data
[DeepmdDataSystem] The training data.

Examples

```
>>> loader = DatasetLoader(train_data)
>>> data_op = loader.build()
>>> with tf.Session() as sess:
>>>     data_list = sess.run(data_op)
>>> data_dict = loader.get_data_dict(data_list)
```

build() → List[deepmd.tf.env.tf.Tensor]

Build the OP that loads the training data.

Returns

List[tf.Tensor]
Tensor of the loaded data.

get_data_dict(batch_list: List[numpy.ndarray]) → Dict[str, numpy.ndarray]

Generate a dict of the loaded data.

Parameters

batch_list
[List[np.ndarray]] The loaded data.

Returns

Dict[str, np.ndarray]
The dict of the loaded data.

deepmd.tf.utils

Submodules

deepmd.tf.utils.argcheck

Alias for backward compatibility.

Module Contents

Functions

gen_args(→ List[dargs.Argument])	
gen_doc(*[, make_anchor, make_link])	
gen_json(**kwargs)	
list_to_doc(xx)	
normalize(data)	
type_embedding_args()	

deepmd.tf.utils.argcheck.**gen_args**(**kwargs) → List[dargs.Argument]

deepmd.tf.utils.argcheck.**gen_doc**(*, make_anchor=True, make_link=True, **kwargs)

deepmd.tf.utils.argcheck.**gen_json**(**kwargs)

deepmd.tf.utils.argcheck.**list_to_doc**(xx)

deepmd.tf.utils.argcheck.**normalize**(data)

deepmd.tf.utils.argcheck.**type_embedding_args**()

deepmd.tf.utils.batch_size

Module Contents

Classes

AutoBatchSize	This class allows DeePMD-kit to automatically decide the maximum

class deepmd.tf.utils.batch_size.**AutoBatchSize**(initial_batch_size: int = 1024, factor: float = 2.0)

Bases: *deepmd.utils.batch_size.AutoBatchSize*

This class allows DeePMD-kit to automatically decide the maximum batch size that will not cause an OOM error.

Parameters

initial_batch_size
: [int, default: 1024] initial batch size (number of total atoms) when DP_INFER_BATCH_SIZE is not set

factor
: [float, default: 2.] increased factor

Notes

In some CPU environments, the program may be directly killed when OOM. In this case, by default the batch size will not be increased for CPUs. The environment variable DP_INFER_BATCH_SIZE can be set as the batch size.

In other cases, we assume all OOM error will raise `OutOfMemoryError`.

Attributes

current_batch_size
: [`int`] current batch size (number of total atoms)

maximum_working_batch_size
: [`int`] maximum working batch size

minimal_not_working_batch_size
: [`int`] minimal not working batch size

is_gpu_available() → bool

Check if GPU is available.

Returns

bool
: True if GPU is available

is_oom_error(e: Exception) → bool

Check if the exception is an OOM error.

Parameters

e
: [`Exception`] Exception

deepmd.tf.utils.compat

Alias for backward compatibility.

Module Contents

Functions

convert_input_v0_v1(→ Dict[str, Any])	Convert input from v0 format to v1.
convert_input_v1_v2(→ Dict[str, Any])	
deprecate_numb_test(→ Dict[str, Any])	Deprecate numb_test since v2.1. It has taken no effect since v2.0.
update_deepmd_input(→ Dict[str, Any])	

deepmd.tf.utils.compat.**convert_input_v0_v1**(jdata: Dict[str, Any], warning: bool = True, dump: str | pathlib.Path | None = None) → Dict[str, Any]

Convert input from v0 format to v1.

Parameters

jdata
: [`Dict`[`str`, *`Any`*]] loaded json/yaml file

warning
: [bool, `optional`] whether to show deprecation warning, by default True

dump
: [`Optional`[`Union`[`str`, `Path`]], `optional`] whether to dump converted file, by default None

Returns

`Dict`[`str`, *`Any`*]
: converted output

deepmd.tf.utils.compat.convert_input_v1_v2(jdata: Dict[str, Any], warning: bool = True, dump: str | pathlib.Path | None = None) → Dict[str, Any]

deepmd.tf.utils.compat.deprecate_numb_test(jdata: Dict[str, Any], warning: bool = True, dump: str | pathlib.Path | None = None) → Dict[str, Any]

Deprecate numb_test since v2.1. It has taken no effect since v2.0.

See #1243.

Parameters

jdata
: [`Dict`[`str`, *`Any`*]] loaded json/yaml file

warning
: [bool, `optional`] whether to show deprecation warning, by default True

dump
: [`Optional`[`Union`[`str`, `Path`]], `optional`] whether to dump converted file, by default None

Returns

`Dict`[`str`, *`Any`*]
: converted output

deepmd.tf.utils.compat.update_deepmd_input(jdata: Dict[str, Any], warning: bool = True, dump: str | pathlib.Path | None = None) → Dict[str, Any]

deepmd.tf.utils.compress

Module Contents

Functions

get_type_embedding(self, graph)	
get_two_side_type_embedding(self, graph)	
get_extra_side_embedding_net_variable(self, graph_def, ...)	
_layer_0(self, x, w, b)	
_layer_1(self, x, w, b)	
make_data(self, xx)	

Attributes

log	

deepmd.tf.utils.compress.**log**

deepmd.tf.utils.compress.**get_type_embedding**(self, graph)

deepmd.tf.utils.compress.**get_two_side_type_embedding**(self, graph)

deepmd.tf.utils.compress.**get_extra_side_embedding_net_variable**(self, graph_def, type_side_suffix, varialbe_name, suffix)

deepmd.tf.utils.compress.**_layer_0**(self, x, w, b)

deepmd.tf.utils.compress.**_layer_1**(self, x, w, b)

deepmd.tf.utils.compress.**make_data**(self, xx)

deepmd.tf.utils.convert

Module Contents

Functions

detect_model_version(input_model)	Detect DP graph version.
convert_to_21(input_model, output_model[, version])	Convert DP graph to 2.1 graph.
convert_13_to_21(input_model, output_model)	Convert DP 1.3 graph to 2.1 graph.
convert_12_to_21(input_model, output_model)	Convert DP 1.2 graph to 2.1 graph.
convert_10_to_21(input_model, output_model)	Convert DP 1.0 graph to 2.1 graph.
convert_012_to_21(input_model, output_model)	Convert DP 0.12 graph to 2.1 graph.
convert_20_to_21(input_model, output_model)	Convert DP 2.0 graph to 2.1 graph.
convert_pb_to_pbtxt(pbfile, pbtxtfile)	Convert DP graph to graph text.
convert_pbtxt_to_pb(pbtxtfile, pbfile)	Convert DP graph text to graph.
convert_dp012_to_dp10(file)	Convert DP 0.12 graph text to 1.0 graph text.
convert_dp10_to_dp11(file)	Convert DP 1.0 graph text to 1.1 graph text.
convert_dp12_to_dp13(file)	Convert DP 1.2 graph text to 1.3 graph text.
convert_dp13_to_dp20(fname)	Convert DP 1.3 graph text to 2.0 graph text.
convert_dp20_to_dp21(fname)	

Attributes

log

deepmd.tf.utils.convert.**log**

deepmd.tf.utils.convert.**detect_model_version**(input_model: str)

Detect DP graph version.

Parameters

input_model
[str] filename of the input graph

deepmd.tf.utils.convert.**convert_to_21**(input_model: str, output_model: str, version: str | None = None)

Convert DP graph to 2.1 graph.

Parameters

input_model
[str] filename of the input graph

output_model
[str] filename of the output graph

version
[str] version of the input graph, if not specified, it will be detected automatically

deepmd.tf.utils.convert.**convert_13_to_21**(input_model: str, output_model: str)

Convert DP 1.3 graph to 2.1 graph.

Parameters

input_model
[str] filename of the input graph

output_model
[str] filename of the output graph

deepmd.tf.utils.convert.convert_12_to_21(input_model: str, output_model: str)
Convert DP 1.2 graph to 2.1 graph.

Parameters

input_model
[str] filename of the input graph

output_model
[str] filename of the output graph

deepmd.tf.utils.convert.convert_10_to_21(input_model: str, output_model: str)
Convert DP 1.0 graph to 2.1 graph.

Parameters

input_model
[str] filename of the input graph

output_model
[str] filename of the output graph

deepmd.tf.utils.convert.convert_012_to_21(input_model: str, output_model: str)
Convert DP 0.12 graph to 2.1 graph.

Parameters

input_model
[str] filename of the input graph

output_model
[str] filename of the output graph

deepmd.tf.utils.convert.convert_20_to_21(input_model: str, output_model: str)
Convert DP 2.0 graph to 2.1 graph.

Parameters

input_model
[str] filename of the input graph

output_model
[str] filename of the output graph

deepmd.tf.utils.convert.convert_pb_to_pbtxt(pbfile: str, pbtxtfile: str)
Convert DP graph to graph text.

Parameters

pbfile
[str] filename of the input graph

pbtxtfile
[str] filename of the output graph text

deepmd.tf.utils.convert.convert_pbtxt_to_pb(pbtxtfile: str, pbfile: str)

Convert DP graph text to graph.

Parameters

pbtxtfile
[str] filename of the input graph text

pbfile
[str] filename of the output graph

deepmd.tf.utils.convert.convert_dp012_to_dp10(file: str)

Convert DP 0.12 graph text to 1.0 graph text.

Parameters

file
[str] filename of the graph text

deepmd.tf.utils.convert.convert_dp10_to_dp11(file: str)

Convert DP 1.0 graph text to 1.1 graph text.

Parameters

file
[str] filename of the graph text

deepmd.tf.utils.convert.convert_dp12_to_dp13(file: str)

Convert DP 1.2 graph text to 1.3 graph text.

Parameters

file
[str] filename of the graph text

deepmd.tf.utils.convert.convert_dp13_to_dp20(fname: str)

Convert DP 1.3 graph text to 2.0 graph text.

Parameters

fname
[str] filename of the graph text

deepmd.tf.utils.convert.convert_dp20_to_dp21(fname: str)

deepmd.tf.utils.data

Alias for backward compatibility.

Module Contents

Classes

DeepmdData	Class for a data system.

class deepmd.tf.utils.data.DeepmdData(sys_path: str, set_prefix: str = 'set', shuffle_test: bool = True, type_map: List[str] | None = None, optional_type_map: bool = True, modifier=None, trn_all_set: bool = False, sort_atoms: bool = True)

Class for a data system.

It loads data from hard disk, and mantains the data as a data_dict

Parameters

sys_path
Path to the data system

set_prefix
Prefix for the directories of different sets

shuffle_test
If the test data are shuffled

type_map
Gives the name of different atom types

optional_type_map
If the type_map.raw in each system is optional

modifier
Data modifier that has the method modify_data

trn_all_set
Use all sets as training dataset. Otherwise, if the number of sets is more than 1, the last set is left for test.

sort_atoms
[bool] Sort atoms by atom types. Required to enable when the data is directly feeded to descriptors except mixed types.

add(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: List[int] | None = None, repeat: int = 1, default: float = 0.0, dtype: numpy.dtype | None = None, output_natoms_for_type_sel: bool = False)

Add a data item that to be loaded.

Parameters

key
The key of the item. The corresponding data is stored in sys_path/set.*/key.npy

ndof
The number of dof

atomic
The item is an atomic property. If False, the size of the data should be nframes x ndof If True, the size of data should be nframes x natoms x ndof

must
The data file sys_path/set.*/key.npy must exist. If must is False and the data file does not exist, the data_dict[find_key] is set to 0.0

high_prec
Load the data and store in float64, otherwise in float32

type_sel
Select certain type of atoms

repeat
The data will be repeated repeat times.

default
[float, default=0.] default value of data

dtype
[np.dtype, optional] the dtype of data, overwrites high_prec if provided

output_natoms_for_type_sel
[bool, optional] if True and type_sel is True, the atomic dimension will be natoms instead of nsel

reduce(key_out: str, key_in: str)

Generate a new item from the reduction of another atom.

Parameters

key_out
The name of the reduced item

key_in
The name of the data item to be reduced

get_data_dict() → dict

Get the data_dict.

check_batch_size(batch_size)

Check if the system can get a batch of data with batch_size frames.

check_test_size(test_size)

Check if the system can get a test dataset with test_size frames.

get_item_torch(index: int) → dict

Get a single frame data . The frame is picked from the data system by index. The index is coded across all the sets.

Parameters

index
index of the frame

get_batch(batch_size: int) → dict

Get a batch of data with batch_size frames. The frames are randomly picked from the data system.

Parameters

batch_size
size of the batch

get_test(ntests: int = -1) → dict

Get the test data with ntests frames.

Parameters

ntests
Size of the test data set. If ntests is -1, all test data will be get.

get_ntypes() → int

Number of atom types in the system.

get_type_map() → List[str]

Get the type map.

get_atom_type() → List[int]

Get atom types.

get_numb_set() → int

Get number of training sets.

get_numb_batch(batch_size: int, set_idx: int) → int

Get the number of batches in a set.

get_sys_numb_batch(batch_size: int) → int

Get the number of batches in the data system.

get_natoms()

Get number of atoms.

get_natoms_vec(ntypes: int)

Get number of atoms and number of atoms in different types.

Parameters

ntypes
Number of types (may be larger than the actual number of types in the system).

Returns

natoms
natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

avg(key)

Return the average value of an item.

_idx_map_sel(atom_type, type_sel)

_get_natoms_2(ntypes)

_get_subdata(data, idx=None)

_load_batch_set(set_name: deepmd.utils.path.DPPath)

reset_get_batch()

_load_test_set(set_name: deepmd.utils.path.DPPath, shuffle_test)

_shuffle_data(data)

_get_nframes(set_name: deepmd.utils.path.DPPath)

reformat_data_torch(data)

Modify the data format for the requirements of Torch backend.

Parameters

data
original data

_load_set(set_name: deepmd.utils.path.DPPath)

_load_data(set_name, key, nframes, ndof_, atomic=False, must=True, repeat=1, high_prec=False, type_sel=None, default: float = 0.0, dtype: numpy.dtype | None = None, output_natoms_for_type_sel: bool = False)

_load_type(sys_path: deepmd.utils.path.DPPath)

_load_type_mix(set_name: deepmd.utils.path.DPPath)

_make_idx_map(atom_type)

_load_type_map(sys_path: deepmd.utils.path.DPPath)

_check_pbc(sys_path: deepmd.utils.path.DPPath)

_check_mode(set_path: deepmd.utils.path.DPPath)

deepmd.tf.utils.data_system

Alias for backward compatibility.

Module Contents

Classes

DeepmdDataSystem	Class for manipulating many data systems.

Functions

prob_sys_size_ext(keywords, nsystems, nbatch)	
process_sys_probs(sys_probs, nbatch)	

class deepmd.tf.utils.data_system.DeepmdDataSystem(systems: List[str], batch_size: int, test_size: int, rcut: float | None = None, set_prefix: str = 'set', shuffle_test: bool = True, type_map: List[str] | None = None, optional_type_map: bool = True, modifier=None, trn_all_set=False, sys_probs=None, auto_prob_style='prob_sys_size', sort_atoms: bool = True)

Class for manipulating many data systems.

It is implemented with the help of DeepmdData

property default_mesh: List[numpy.ndarray]

Mesh for each system.

_load_test(ntests=-1)

`compute_energy_shift`(rcond=None, key='energy')

`add_dict`(adict: dict) → None

Add items to the data system by a dict. adict should have items like .. code-block:: python.

adict[key] = {
"ndof": ndof, "atomic": atomic, "must": must, "high_prec": high_prec, "type_sel": type_sel, "repeat": repeat,

}

For the explaination of the keys see add

`add`(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: List[int] | None = None, repeat: int = 1, default: float = 0.0, dtype: numpy.dtype | None = None, output_natoms_for_type_sel: bool = False)

Add a data item that to be loaded.

Parameters

key
The key of the item. The corresponding data is stored in sys_path/set.*/key.npy

ndof
The number of dof

atomic
The item is an atomic property. If False, the size of the data should be nframes x ndof If True, the size of data should be nframes x natoms x ndof

must
The data file sys_path/set.*/key.npy must exist. If must is False and the data file does not exist, the data_dict[find_key] is set to 0.0

high_prec
Load the data and store in float64, otherwise in float32

type_sel
Select certain type of atoms

repeat
The data will be repeated repeat times.

default, default=0.
Default value of data

dtype
The dtype of data, overwrites high_prec if provided

output_natoms_for_type_sel
[bool] If True and type_sel is True, the atomic dimension will be natoms instead of nsel

`reduce`(key_out, key_in)

Generate a new item from the reduction of another atom.

Parameters

key_out
The name of the reduced item

key_in
The name of the data item to be reduced

get_data_dict(ii: int = 0) → dict

set_sys_probs(sys_probs=None, auto_prob_style: str = 'prob_sys_size')

get_batch(sys_idx: int | None = None) → dict

Get a batch of data from the data systems.

Parameters

sys_idx
: [int] The index of system from which the batch is get. If sys_idx is not None, sys_probs and auto_prob_style are ignored If sys_idx is None, automatically determine the system according to sys_probs or auto_prob_style, see the following. This option does not work for mixed systems.

Returns

dict
: The batch data

get_batch_standard(sys_idx: int | None = None) → dict

Get a batch of data from the data systems in the standard way.

Parameters

sys_idx
: [int] The index of system from which the batch is get. If sys_idx is not None, sys_probs and auto_prob_style are ignored If sys_idx is None, automatically determine the system according to sys_probs or auto_prob_style, see the following.

Returns

dict
: The batch data

get_batch_mixed() → dict

Get a batch of data from the data systems in the mixed way.

Returns

dict
: The batch data

_merge_batch_data(batch_data: List[dict]) → dict

Merge batch data from different systems.

Parameters

batch_data
: [list of dict] A list of batch data from different systems.

Returns

dict
: The merged batch data.

get_test(sys_idx: int | None = None, n_test: int = -1)

Get test data from the the data systems.

Parameters

sys_idx
: The test dat of system with index sys_idx will be returned. If is None, the currently selected system will be returned.

n_test
Number of test data. If set to -1 all test data will be get.

get_sys_ntest(sys_idx=None)
Get number of tests for the currently selected system, or one defined by sys_idx.

get_type_map() → List[str]
Get the type map.

get_nbatches() → int
Get the total number of batches.

get_ntypes() → int
Get the number of types.

get_nsystems() → int
Get the number of data systems.

get_sys(idx: int) → deepmd.utils.data.DeepmdData
Get a certain data system.

get_batch_size() → int
Get the batch size.

print_summary(name: str)

_make_auto_bs(rule)

_make_auto_ts(percent)

_check_type_map_consistency(type_map_list)

deepmd.tf.utils.data_system.**prob_sys_size_ext**(keywords, nsystems, nbatch)

deepmd.tf.utils.data_system.**process_sys_probs**(sys_probs, nbatch)

deepmd.tf.utils.errors

Module Contents

exception deepmd.tf.utils.errors.**OutOfMemoryError**
Bases: Exception

This error is caused by out-of-memory (OOM).

exception deepmd.tf.utils.errors.**GraphTooLargeError**
Bases: Exception

The graph is too large, exceeding protobuf's hard limit of 2GB.

exception deepmd.tf.utils.errors.**GraphWithoutTensorError**
Bases: Exception

Common base class for all non-exit exceptions.

deepmd.tf.utils.finetune

Module Contents

Functions

replace_model_params_with_pretrained_model(j... ...)	Replace the model params in input script according to pretrained model.

Attributes

log

deepmd.tf.utils.finetune.log

deepmd.tf.utils.finetune.**replace_model_params_with_pretrained_model**(jdata: Dict[str, Any], pretrained_model: str)

Replace the model params in input script according to pretrained model.

Parameters

jdata
: [Dict[str, *Any*]] input script

pretrained_model
: [str] filename of the pretrained model

deepmd.tf.utils.graph

Module Contents

Functions

load_graph_def(→ Tuple[deepmd.tf.env.tf.Graph, ...)	Load graph as well as the graph_def from the frozen model(model_file).
get_tensor_by_name_from_graph(→ deepmd.tf.env.tf.Tensor)	Load tensor value from the given tf.Graph object.
get_tensor_by_name(→ deepmd.tf.env.tf.Tensor)	Load tensor value from the frozen model(model_file).
get_pattern_nodes_from_graph_def(→ Dict)	Get the pattern nodes with the given tf.GraphDef object.
get_embedding_net_nodes_from_graph_def(→ Dict)	Get the embedding net nodes with the given tf.GraphDef object.
get_embedding_net_nodes(→ Dict)	Get the embedding net nodes with the given frozen model(model_file).
get_embedding_net_variables_from_graph_def(→ Dict)	Get the embedding net variables with the given tf.GraphDef object.
get_extra_embedding_net_suffix(type_one_side)	Get the extra embedding net suffix according to the value of type_one_side.
get_extra_embedding_net_nodes_from_graph_def(→ Dict)	Get the extra embedding net nodes with the given tf.GraphDef object.
get_extra_embedding_net_variables_from_graph_def(→ Dict)	Get the embedding net variables with the given tf.GraphDef object.
get_embedding_net_variables(→ Dict)	Get the embedding net variables with the given frozen model(model_file).
get_fitting_net_nodes_from_graph_def(→ Dict)	Get the fitting net nodes with the given tf.GraphDef object.
get_fitting_net_nodes(→ Dict)	Get the fitting net nodes with the given frozen model(model_file).
get_fitting_net_variables_from_graph_def(→ Dict)	Get the fitting net variables with the given tf.GraphDef object.
get_fitting_net_variables(→ Dict)	Get the fitting net variables with the given frozen model(model_file).
get_type_embedding_net_nodes_from_graph_def(→ Dict)	Get the type embedding net nodes with the given tf.GraphDef object.
get_type_embedding_net_variables_from_graph_def(→ Dict)	Get the type embedding net variables with the given tf.GraphDef object.
get_attention_layer_nodes_from_graph_def(→ Dict)	Get the attention layer nodes with the given tf.GraphDef object.
get_attention_layer_variables_from_graph_def(→ Dict)	Get the attention layer variables with the given tf.GraphDef object.
convert_tensor_to_ndarray_in_dict(→ Dict[str, ...)	Convert tensor to ndarray in dict.

deepmd.tf.utils.graph.**load_graph_def**(model_file: str) → Tuple[deepmd.tf.env.tf.Graph, deepmd.tf.env.tf.GraphDef]

Load graph as well as the graph_def from the frozen model(model_file).

Parameters

model_file
[str] The input frozen model path

Returns

tf.Graph
The graph loaded from the frozen model

tf.GraphDef
The graph_def loaded from the frozen model

deepmd.tf.utils.graph.**get_tensor_by_name_from_graph**(graph: deepmd.tf.env.tf.Graph, tensor_name: str) → deepmd.tf.env.tf.Tensor

Load tensor value from the given tf.Graph object.

Parameters

graph
[tf.Graph] The input TensorFlow graph

tensor_name
[str] Indicates which tensor which will be loaded from the frozen model

Returns

tf.Tensor
The tensor which was loaded from the frozen model

Raises

GraphWithoutTensorError
Whether the tensor_name is within the frozen model

deepmd.tf.utils.graph.**get_tensor_by_name**(model_file: str, tensor_name: str) → deepmd.tf.env.tf.Tensor

Load tensor value from the frozen model(model_file).

Parameters

model_file
[str] The input frozen model path

tensor_name
[str] Indicates which tensor which will be loaded from the frozen model

Returns

tf.Tensor
The tensor which was loaded from the frozen model

Raises

GraphWithoutTensorError
Whether the tensor_name is within the frozen model

deepmd.tf.utils.graph.**get_pattern_nodes_from_graph_def**(graph_def: deepmd.tf.env.tf.GraphDef, pattern: str) → Dict

Get the pattern nodes with the given tf.GraphDef object.

Parameters

graph_def
The input tf.GraphDef object

pattern
The node pattern within the graph_def

Returns

Dict
The fitting net nodes within the given tf.GraphDef object

deepmd.tf.utils.graph.**get_embedding_net_nodes_from_graph_def**(graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → Dict

Get the embedding net nodes with the given tf.GraphDef object.

Parameters

graph_def
The input tf.GraphDef object

suffix
[str, optional] The scope suffix

Returns

Dict
The embedding net nodes within the given tf.GraphDef object

deepmd.tf.utils.graph.**get_embedding_net_nodes**(model_file: str, suffix: str = '') → Dict

Get the embedding net nodes with the given frozen model(model_file).

Parameters

model_file
The input frozen model path

suffix
[str, optional] The suffix of the scope

Returns

Dict
The embedding net nodes with the given frozen model

deepmd.tf.utils.graph.**get_embedding_net_variables_from_graph_def**(graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → Dict

Get the embedding net variables with the given tf.GraphDef object.

Parameters

graph_def
The input tf.GraphDef object

suffix
[str, optional] The suffix of the scope

Returns

Dict
The embedding net variables within the given tf.GraphDef object

deepmd.tf.utils.graph.**get_extra_embedding_net_suffix**(type_one_side: bool)

Get the extra embedding net suffix according to the value of type_one_side.

Parameters

type_one_side
The value of type_one_side

Returns

str
The extra embedding net suffix

deepmd.tf.utils.graph.**get_extra_embedding_net_nodes_from_graph_def**(graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '', extra_suffix: str = '') → Dict

Get the extra embedding net nodes with the given tf.GraphDef object.

Parameters

graph_def
The input tf.GraphDef object

suffix
[str, optional] The scope suffix

extra_suffix
[str] The extra scope suffix

Returns

Dict
The embedding net nodes within the given tf.GraphDef object

deepmd.tf.utils.graph.**get_extra_embedding_net_variables_from_graph_def**(graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '', extra_suffix: str = '') → Dict

Get the embedding net variables with the given tf.GraphDef object.

Parameters

graph_def
The input tf.GraphDef object

suffix
[str, optional] The suffix of the scope

extra_suffix
The extra scope suffix

Returns

Dict
The embedding net variables within the given tf.GraphDef object

deepmd.tf.utils.graph.**get_embedding_net_variables**(model_file: str, suffix: str = '') → Dict

Get the embedding net variables with the given frozen model(model_file).

Parameters

model_file
The input frozen model path

suffix
[str, optional] The suffix of the scope

Returns

Dict
The embedding net variables within the given frozen model

deepmd.tf.utils.graph.**get_fitting_net_nodes_from_graph_def**(graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → Dict

Get the fitting net nodes with the given tf.GraphDef object.

Parameters

graph_def
The input tf.GraphDef object

suffix
suffix of the scope

Returns

Dict
The fitting net nodes within the given tf.GraphDef object

deepmd.tf.utils.graph.**get_fitting_net_nodes**(model_file: str) → Dict

Get the fitting net nodes with the given frozen model(model_file).

Parameters

model_file
The input frozen model path

Returns

Dict
The fitting net nodes with the given frozen model

deepmd.tf.utils.graph.**get_fitting_net_variables_from_graph_def**(graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → Dict

Get the fitting net variables with the given tf.GraphDef object.

Parameters

graph_def
The input tf.GraphDef object

suffix
suffix of the scope

Returns

Dict
The fitting net variables within the given tf.GraphDef object

deepmd.tf.utils.graph.**get_fitting_net_variables**(model_file: str, suffix: str = '') → Dict

Get the fitting net variables with the given frozen model(model_file).

Parameters

model_file
The input frozen model path

suffix
suffix of the scope

Returns

Dict
The fitting net variables within the given frozen model

deepmd.tf.utils.graph.**get_type_embedding_net_nodes_from_graph_def**(graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → Dict

Get the type embedding net nodes with the given tf.GraphDef object.

Parameters

graph_def
The input tf.GraphDef object

suffix
[str, optional] The scope suffix

Returns

Dict
The type embedding net nodes within the given tf.GraphDef object

deepmd.tf.utils.graph.**get_type_embedding_net_variables_from_graph_def**(graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → Dict

Get the type embedding net variables with the given tf.GraphDef object.

Parameters

graph_def
[tf.GraphDef] The input tf.GraphDef object

suffix
[str, optional] The suffix of the scope

Returns

Dict
The embedding net variables within the given tf.GraphDef object

deepmd.tf.utils.graph.**get_attention_layer_nodes_from_graph_def**(graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → Dict

Get the attention layer nodes with the given tf.GraphDef object.

Parameters

graph_def
The input tf.GraphDef object

suffix
[str, optional] The scope suffix

Returns

Dict
The attention layer nodes within the given tf.GraphDef object

deepmd.tf.utils.graph.**get_attention_layer_variables_from_graph_def**(graph_def: deepmd.tf.env.tf.GraphDef, suffix: str = '') → Dict

Get the attention layer variables with the given tf.GraphDef object.

Parameters

graph_def
[`tf.GraphDef`] The input tf.GraphDef object

suffix
[`str`, `optional`] The suffix of the scope

Returns

`Dict`
The attention layer variables within the given tf.GraphDef object

deepmd.tf.utils.graph.**convert_tensor_to_ndarray_in_dict**(tensor_dict: Dict[str, deepmd.tf.env.tf.Tensor]) → Dict[str, numpy.ndarray]

Convert tensor to ndarray in dict.

Parameters

tensor_dict
[`Dict`[`str`, `tf.Tensor`]] The input tensor dict

Returns

`Dict`[`str`, `np.ndarray`]
The converted tensor dict

deepmd.tf.utils.learning_rate

Module Contents

Classes

LearningRateExp	The exponentially decaying learning rate.

class deepmd.tf.utils.learning_rate.**LearningRateExp**(start_lr: float, stop_lr: float = 5e-08, decay_steps: int = 5000, decay_rate: float = 0.95)

The exponentially decaying learning rate.

The learning rate at step t is given by

$$\alpha(t) = \alpha_0 \lambda^{t/\tau}$$

where α is the learning rate, α_0 is the starting learning rate, λ is the decay rate, and τ is the decay steps.

Parameters

start_lr
Starting learning rate α_0

stop_lr
: Stop learning rate α_1

decay_steps
: Learning rate decay every this number of steps τ

decay_rate
: The decay rate λ. If stop_step is provided in build, then it will be determined automatically and overwritten.

build(global_step: deepmd.tf.env.tf.Tensor, stop_step: int | None = None) → deepmd.tf.env.tf.Tensor

Build the learning rate.

Parameters

global_step
: The tf Tensor prividing the global training step

stop_step
: The stop step. If provided, the decay_rate will be determined automatically and overwritten.

Returns

learning_rate
: The learning rate

start_lr() → float

Get the start lr.

value(step: int) → float

Get the lr at a certain step.

deepmd.tf.utils.multi_init

Module Contents

Functions

replace_model_params_with_frz_multi_model(jd ...)	Replace the model params in input script according to pretrained frozen multi-task united model.
_change_sub_config(jdata, src_jdata, sub_key)	

Attributes

log

deepmd.tf.utils.multi_init.**log**

deepmd.tf.utils.multi_init.**replace_model_params_with_frz_multi_model**(jdata: Dict[str, Any], pretrained_model: str)

Replace the model params in input script according to pretrained frozen multi-task united model.

Parameters

jdata
: [Dict[str, *Any*]] input script

pretrained_model
: [str] filename of the pretrained frozen multi-task united model

deepmd.tf.utils.multi_init.**_change_sub_config**(jdata: Dict[str, Any], src_jdata: Dict[str, Any], sub_key: str)

deepmd.tf.utils.neighbor_stat

Module Contents

Classes

NeighborStatOP	Class for getting neighbor statics data information.
NeighborStat	Class for getting training data information.

Attributes

log

deepmd.tf.utils.neighbor_stat.**log**

class deepmd.tf.utils.neighbor_stat.**NeighborStatOP**(ntypes: int, rcut: float, mixed_types: bool)

Class for getting neighbor statics data information.

Parameters

ntypes
: The num of atom types

rcut
: The cut-off radius

mixed_types
[bool, optional] If True, treat neighbors of all types as a single type.

build(coord: deepmd.tf.env.tf.Tensor, atype: deepmd.tf.env.tf.Tensor, cell: deepmd.tf.env.tf.Tensor, pbc: deepmd.tf.env.tf.Tensor) → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Calculate the nearest neighbor distance between atoms, maximum nbor size of atoms and the output data range of the environment matrix.

Parameters

coord
The coordinates of atoms.

atype
The atom types.

cell
The cell.

Returns

tf.Tensor
The minimal squared distance between two atoms, in the shape of (nframes,)

tf.Tensor
The maximal number of neighbors

class deepmd.tf.utils.neighbor_stat.NeighborStat(ntypes: int, rcut: float, mixed_type: bool = False)

Bases: *deepmd.utils.neighbor_stat.NeighborStat*

Class for getting training data information.

It loads data from DeepmdData object, and measures the data info, including neareest nbor distance between atoms, max nbor size of atoms and the output data range of the environment matrix.

Parameters

ntypes
The num of atom types

rcut
The cut-off radius

mixed_type
[bool, optional, default=False] Treat all types as a single type.

build() → Tuple[deepmd.tf.env.tf.Tensor, deepmd.tf.env.tf.Tensor]

Build the graph.

Returns

tf.Tensor
The minimal squared distance between two atoms, in the shape of (nframes,)

tf.Tensor
The maximal number of neighbors

iterator(data: deepmd.tf.utils.data_system.DeepmdDataSystem) → Iterator[Tuple[numpy.ndarray, float, str]]

Produce data.

Parameters

data
The data system

Yields

np.ndarray
The maximal number of neighbors

float
The squared minimal distance between two atoms

str
The directory of the data system

_execute(coord: numpy.ndarray, atype: numpy.ndarray, box: numpy.ndarray | None, pbc: bool)

Execute the operation.

Parameters

coord
The coordinates of atoms.

atype
The atom types.

box
The box.

pbc
Whether the box is periodic.

deepmd.tf.utils.network

Module Contents

Functions

one_layer_rand_seed_shift()	
one_layer(inputs, outputs_size[, activation_fn, ...])	
layer_norm_tf(x, shape[, weight, bias, eps])	Layer normalization implementation in TensorFlow.
layernorm(inputs, outputs_size[, precision, name, ...])	
embedding_net_rand_seed_shift(network_size)	
embedding_net(xx, network_size, precision[, ...])	The embedding network.
variable_summaries(var, name)	Attach a lot of summaries to a Tensor (for TensorBoard visualization).

deepmd.tf.utils.network.**one_layer_rand_seed_shift**()

deepmd.tf.utils.network.**one_layer**(inputs, outputs_size, activation_fn=tf.nn.tanh, precision=GLOBAL_TF_FLOAT_PRECISION, stddev=1.0, bavg=0.0, name='linear', scope='', reuse=None, seed=None, use_timestep=False, trainable=True, useBN=False, uniform_seed=False, initial_variables=None, mixed_prec=None, final_layer=False)

deepmd.tf.utils.network.**layer_norm_tf**(x, shape, weight=None, bias=None, eps=1e-05)

Layer normalization implementation in TensorFlow.

Parameters

x
[tf.Tensor] The input tensor.

shape
[tuple] The shape of the weight and bias tensors.

weight
[tf.Tensor] The weight tensor.

bias
[tf.Tensor] The bias tensor.

eps
[float] A small value added to prevent division by zero.

Returns

tf.Tensor
The normalized output tensor.

deepmd.tf.utils.network.**layernorm**(inputs, outputs_size, precision=GLOBAL_TF_FLOAT_PRECISION, name='linear', scope='', reuse=None, seed=None, uniform_seed=False, uni_init=True, eps=1e-05, trainable=True, initial_variables=None)

deepmd.tf.utils.network.**embedding_net_rand_seed_shift**(network_size)

deepmd.tf.utils.network.**embedding_net**(xx, network_size, precision, activation_fn=tf.nn.tanh, resnet_dt=False, name_suffix='', stddev=1.0, bavg=0.0, seed=None, trainable=True, uniform_seed=False, initial_variables=None, mixed_prec=None)

The embedding network.

The embedding network function $\mathcal{N}$ is constructed by is the composition of multiple layers $\mathcal{L}^{(i)}$:

$$\mathcal{N} = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \cdots \circ \mathcal{L}^{(1)}$$

A layer $\mathcal{L}$ is given by one of the following forms, depending on the number of nodes: [1]

$$\mathrm{y} = \mathcal{L}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \begin{cases} \boldsymbol{\phi}(\mathrm{x}^T \mathrm{w} + \mathrm{b}) + \mathrm{x}, & N_2 = N_1 \\ \boldsymbol{\phi}(\mathrm{x}^T \mathrm{w} + \mathrm{b}) + (\mathrm{x}, \mathrm{x}), & N_2 = 2N_1 \\ \boldsymbol{\phi}(\mathrm{x}^T \mathrm{w} + \mathrm{b}), & \text{otherwise} \end{cases}$$

where $\mathrm{x} \in \mathbb{R}^{N_1}$ is the input vector and $\mathrm{y} \in \mathbb{R}^{N_2}$ is the output vector. $\mathrm{w} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathrm{b} \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable is True. $\boldsymbol{\phi}$ is the activation function.

Parameters

xx
: [Tensor] Input tensor x of shape [-1,1]

network_size
: [list of int] Size of the embedding network. For example [16,32,64]

precision:
: Precision of network weights. For example, tf.float64

activation_fn:
: Activation function ϕ

resnet_dt
: [bool] Using time-step in the ResNet construction

name_suffix
: [str] The name suffix append to each variable.

stddev
: [float] Standard deviation of initializing network parameters

bavg
: [float] Mean of network intial bias

seed
: [int] Random seed for initializing network parameters

trainable
: [bool] If the network is trainable

uniform_seed
: [bool] Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

initial_variables
: [dict] The input dict which stores the embedding net variables

mixed_prec
: The input dict which stores the mixed precision setting for the embedding net

References

[1]

deepmd.tf.utils.network.**variable_summaries**(var: deepmd.tf.env.tf.Variable, name: str)

Attach a lot of summaries to a Tensor (for TensorBoard visualization).

Parameters

var
: [tf.Variable] [description]

name
: [str] variable name

deepmd.tf.utils.nlist

Module Contents

Functions

extend_coord_with_ghosts(coord, atype, cell, rcut, pbc)	Extend the coordinates of the atoms by appending peridoc images.

deepmd.tf.utils.nlist.**extend_coord_with_ghosts**(coord: deepmd.tf.env.tf.Tensor, atype: deepmd.tf.env.tf.Tensor, cell: deepmd.tf.env.tf.Tensor, rcut: float, pbc: deepmd.tf.env.tf.Tensor)

Extend the coordinates of the atoms by appending peridoc images. The number of images is large enough to ensure all the neighbors within rcut are appended.

Parameters

coord
: [tf.Tensor] original coordinates of shape [-1, nloc*3].

atype
: [tf.Tensor] atom type of shape [-1, nloc].

cell
: [tf.Tensor] simulation cell tensor of shape [-1, 9].

rcut
: [float] the cutoff radius

pbc
: [tf.Tensor] whether the simulation cell is periodic or not

Returns

extended_coord: tf.Tensor
: extended coordinates of shape [-1, nall*3].

extended_atype: tf.Tensor
: extended atom type of shape [-1, nall].

index_mapping: tf.Tensor
: maping extended index to the local index

deepmd.tf.utils.pair_tab

Alias for backward compatibility.

Module Contents

Classes

PairTab	Pairwise tabulated potential.

class deepmd.tf.utils.pair_tab.PairTab(filename: str, rcut: float | None = None)

Pairwise tabulated potential.

Parameters

filename
: File name for the short-range tabulated potential. The table is a text data file with (N_t + 1) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

reinit(filename: str, rcut: float | None = None) → None

Initialize the tabulated interaction.

Parameters

filename
: File name for the short-range tabulated potential. The table is a text data file with (N_t + 1) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

serialize() → dict

classmethod deserialize(data) → PairTab

_check_table_upper_boundary() → None

Update User Provided Table Based on rcut.

This function checks the upper boundary provided in the table against rcut. If the table upper boundary values decay to zero before rcut, padding zeros will be added to the table to cover rcut; if the table upper boundary values do not decay to zero before ruct, extrapolation will be performed till rcut.

Examples

table = [[0.005 1. 2. 3.]
[0.01 0.8 1.6 2.4] [0.015 0. 1. 1.5]]

rcut = 0.022

new_table = [[0.005 1. 2. 3.]
[0.01 0.8 1.6 2.4] [0.015 0. 1. 1.5] [0.02 0. 0. 0.]

table = [[0.005 1. 2. 3.]
[0.01 0.8 1.6 2.4] [0.015 0.5 1. 1.5] [0.02 0.25 0.4 0.75] [0.025 0. 0.1 0.] [0.03 0. 0. 0.]]

rcut = 0.031

new_table = [[0.005 1. 2. 3.]
[0.01 0.8 1.6 2.4] [0.015 0.5 1. 1.5] [0.02 0.25 0.4 0.75] [0.025 0. 0.1 0.] [0.03 0. 0. 0.] [0.035 0. 0. 0.]]

get() → Tuple[numpy.array, numpy.array]

Get the serialized table.

_extrapolate_table(pad_extrapolation: numpy.array) → numpy.array

Soomth extrapolation between table upper boundary and rcut.

This method should only be used when the table upper boundary rmax is smaller than rcut, and the table upper boundary values are not zeros. To simplify the problem, we use a single cubic spline between rmax and rcut for each pair of atom types. One can substitute this extrapolation to higher order polynomials if needed.

There are two scenarios:

1.ruct - rmax >= hh:
Set values at the grid point right before rcut to 0, and perform exterapolation between the grid point and rmax, this allows smooth decay to 0 at rcut.

2.rcut - rmax < hh:
Set values at rmax + hh to 0, and perform extrapolation between rmax and rmax + hh.

Parameters

pad_extrapolation
[**np.array**] The emepty grid that holds the extrapolation values.

Returns

np.array
The cubic spline extrapolation.

_make_data()

deepmd.tf.utils.parallel_op

Module Contents

Classes

ParallelOp	Run an op with data parallelism.

class deepmd.tf.utils.parallel_op.**ParallelOp**(builder: Callable[Ellipsis, Tuple[Dict[str, deepmd.tf.env.tf.Tensor], Tuple[deepmd.tf.env.tf.Tensor]]], nthreads: int | None = None, config: deepmd.tf.env.tf.ConfigProto | None = None)

Run an op with data parallelism.

Parameters

builder
: [`Callable`[..., `Tuple`[`Dict`[`str`, `tf.Tensor`], `Tuple`[`tf.Tensor`]]]] returns two objects: a dict which stores placeholders by key, and a tuple with the final op(s)

nthreads
: [`int`, `optional`] the number of threads

config
: [`tf.ConfigProto`, `optional`] tf.ConfigProto

Examples

```
>>> from deepmd.tf.env import tf
>>> from deepmd.tf.utils.parallel_op import ParallelOp
>>> def builder():
...     x = tf.placeholder(tf.int32, [1])
...     return {"x": x}, (x + 1)
>>> p = ParallelOp(builder, nthreads=4)
>>> def feed():
...     for ii in range(10):
...         yield {"x": [ii]}
>>> print(*p.generate(tf.Session(), feed()))
[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
```

generate(sess: deepmd.tf.env.tf.Session, feed: Generator[Dict[str, Any], None, None]) → Generator[Tuple, None, None]

Returns a generator.

Parameters

sess
: [`tf.Session`] TensorFlow session

feed
: [`Generator`[`dict`, `None`, `None`]] generator which yields feed_dict

Yields

`Generator`[`Tuple`, `None`, `None`]
: generator which yields session returns

deepmd.tf.utils.path

Alias for backward compatibility.

Module Contents

Classes

DPH5Path	The path class to data system (DeepmdData) for HDF5 files.
DPOSPath	The OS path class to data system (DeepmdData) for real directories.
DPPath	The path class to data system (DeepmdData).

class deepmd.tf.utils.path.DPH5Path(path: str, mode: str = 'r')

Bases: *DPPath*

The path class to data system (DeepmdData) for HDF5 files.

Parameters

path
[str] path

mode
[str, optional] mode, by default "r"

Notes

OS - HDF5 relationship:
directory - Group file - Dataset

property _keys: List[str]
Walk all groups and dataset.

property _new_keys
New keys that haven't been cached.

property name: str
Name of the path.

__file_new_keys: ClassVar[Dict[h5py.File, List[str]]]

classmethod _load_h5py(path: str, mode: str = 'r') → h5py.File
Load hdf5 file.

Parameters

path
[str] path to hdf5 file

mode
[str, optional] mode, by default 'r'

load_numpy() → numpy.ndarray
Load NumPy array.

Returns

np.ndarray
loaded NumPy array

load_txt(dtype: numpy.dtype | None = None, **kwargs) → numpy.ndarray

Load NumPy array from text.

Returns

np.ndarray
loaded NumPy array

save_numpy(arr: numpy.ndarray) → None

Save NumPy array.

Parameters

arr
[np.ndarray] NumPy array

glob(pattern: str) → List[DPPath]

Search path using the glob pattern.

Parameters

pattern
[str] glob pattern

Returns

List[*DPPath*]
list of paths

rglob(pattern: str) → List[DPPath]

This is like calling *DPPath.glob()* with **/ added in front of the given relative pattern.

Parameters

pattern
[str] glob pattern

Returns

List[*DPPath*]
list of paths

classmethod _file_keys(file: h5py.File) → List[str]

Walk all groups and dataset.

is_file() → bool

Check if self is file.

is_dir() → bool

Check if self is directory.

__truediv__(key: str) → DPPath

Used for / operator.

_connect_path(path: str) → str

Connect self with path.

__lt__(other: DPH5Path) → bool

Whether this DPPath is less than other for sorting.

__str__() → str

Returns path of self.

mkdir(parents: bool = False, exist_ok: bool = False) → None

Make directory.

Parameters

parents
[bool, optional] If true, any missing parents of this directory are created as well.

exist_ok
[bool, optional] If true, no error will be raised if the target directory already exists.

class deepmd.tf.utils.path.DPOSPath(path: str, mode: str = 'r')

Bases: *DPPath*

The OS path class to data system (DeepmdData) for real directories.

Parameters

path
[str] path

mode
[str, optional] mode, by default "r"

property name: str

Name of the path.

load_numpy() → numpy.ndarray

Load NumPy array.

Returns

np.ndarray
loaded NumPy array

load_txt(**kwargs) → numpy.ndarray

Load NumPy array from text.

Returns

np.ndarray
loaded NumPy array

save_numpy(arr: numpy.ndarray) → None

Save NumPy array.

Parameters

arr
[np.ndarray] NumPy array

glob(pattern: str) → List[DPPath]

Search path using the glob pattern.

Parameters

pattern
[str] glob pattern

Returns

List[*DPPath*]
list of paths

rglob(pattern: str) → List[DPPath]

This is like calling *DPPath.glob()* with **/ added in front of the given relative pattern.

Parameters

pattern
[str] glob pattern

Returns

List[*DPPath*]
list of paths

is_file() → bool

Check if self is file.

is_dir() → bool

Check if self is directory.

__truediv__(key: str) → DPPath

Used for / operator.

__lt__(other: DPOSPath) → bool

Whether this DPPath is less than other for sorting.

__str__() → str

Represent string.

mkdir(parents: bool = False, exist_ok: bool = False) → None

Make directory.

Parameters

parents
[bool, optional] If true, any missing parents of this directory are created as well.

exist_ok
[bool, optional] If true, no error will be raised if the target directory already exists.

class deepmd.tf.utils.path.DPPath

Bases: abc.ABC

The path class to data system (DeepmdData).

Parameters

path
[str] path

mode
[str, optional] mode, by default "r"

abstract property name: str

Name of the path.

abstract load_numpy() → numpy.ndarray

Load NumPy array.

Returns

np.ndarray
loaded NumPy array

abstract load_txt(**kwargs) → numpy.ndarray

Load NumPy array from text.

Returns

np.ndarray
loaded NumPy array

abstract save_numpy(arr: numpy.ndarray) → None

Save NumPy array.

Parameters

arr
[np.ndarray] NumPy array

abstract glob(pattern: str) → List[DPPath]

Search path using the glob pattern.

Parameters

pattern
[str] glob pattern

Returns

List[*DPPath*]
list of paths

abstract rglob(pattern: str) → List[DPPath]

This is like calling *DPPath.glob()* with **/ added in front of the given relative pattern.

Parameters

pattern
[str] glob pattern

Returns

List[*DPPath*]
list of paths

abstract is_file() → bool

Check if self is file.

abstract is_dir() → bool

Check if self is directory.

abstract __truediv__(key: str) → DPPath

Used for / operator.

abstract __lt__(other: DPPath) → bool

Whether this DPPath is less than other for sorting.

abstract __str__() → str

Represent string.

__repr__() → str

Return repr(self).

__eq__(other) → bool

Return self==value.

__hash__()
Return hash(self).

abstract mkdir(parents: bool = False, exist_ok: bool = False) → None
Make directory.

Parameters

parents
[bool, optional] If true, any missing parents of this directory are created as well.

exist_ok
[bool, optional] If true, no error will be raised if the target directory already exists.

deepmd.tf.utils.plugin

Alias for backward compatibility.

Module Contents

Classes

Plugin	A class to register and restore plugins.
PluginVariant	A class to remove type from input arguments.
VariantABCMeta	Metaclass for defining Abstract Base Classes (ABCs).
VariantMeta	

class deepmd.tf.utils.plugin.Plugin
A class to register and restore plugins.

Examples

```
>>> plugin = Plugin()
>>> @plugin.register("xx")
    def xxx():
        pass
>>> print(plugin.plugins["xx"])
```

Attributes

plugins
[Dict[str, object]] plugins

__add__(other) → Plugin

register(key: str) → Callable[[object], object]
Register a plugin.

Parameters

key
[str] key of the plugin

Returns

Callable[[object], object]
decorator

get_plugin(key) → object

Visit a plugin by key.

Parameters

key
[str] key of the plugin

Returns

object
the plugin

class deepmd.tf.utils.plugin.PluginVariant

A class to remove type from input arguments.

class deepmd.tf.utils.plugin.VariantABCMeta

Bases: *VariantMeta*, abc.ABCMeta

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

class deepmd.tf.utils.plugin.VariantMeta

__call__(*args, **kwargs)

Remove type and keys that starts with underline.

deepmd.tf.utils.random

Alias for backward compatibility.

Module Contents

Functions

choice(a[, size, replace, p])	Generates a random sample from a given 1-D array.
random([size])	Return random floats in the half-open interval [0.0, 1.0).
seed([val])	Seed the generator.
shuffle(x)	Modify a sequence in-place by shuffling its contents.

deepmd.tf.utils.random.choice(a: numpy.ndarray | int, size: int | Tuple[int, Ellipsis] | None = None, replace: bool = True, p: numpy.ndarray | None = None)

Generates a random sample from a given 1-D array.

Parameters

a
: [1-D array_like or int] If an ndarray, a random sample is generated from its elements. If an int, the random sample is generated as if it were np.arange(a)

size
: [int or tuple of ints, optional] Output shape. If the given shape is, e.g., (m, n, k), then m * n * k samples are drawn. Default is None, in which case a single value is returned.

replace
: [bool, optional] Whether the sample is with or without replacement. Default is True, meaning that a value of a can be selected multiple times.

p
: [1-D array_like, optional] The probabilities associated with each entry in a. If not given, the sample assumes a uniform distribution over all entries in a.

Returns

np.ndarray
: arrays with results and their shapes

deepmd.tf.utils.random.random(size=None)

Return random floats in the half-open interval [0.0, 1.0).

Parameters

size
: Output shape.

Returns

np.ndarray
: Arrays with results and their shapes.

deepmd.tf.utils.random.seed(val: int | None = None)

Seed the generator.

Parameters

val
: [int] Seed.

deepmd.tf.utils.random.shuffle(x: numpy.ndarray)

Modify a sequence in-place by shuffling its contents.

Parameters

x
: [np.ndarray] The array or list to be shuffled.

deepmd.tf.utils.region

Module Contents

Functions

to_face_distance(cell)	Compute the to-face-distance of the simulation cell.
b_to_face_distance(cell)	

deepmd.tf.utils.region.**to_face_distance**(cell)

Compute the to-face-distance of the simulation cell.

Parameters

cell
[tf.Tensor] simulation cell tensor of shape [*, 3, 3].

Returns

dist: tf.Tensor
the to face distances of shape [*, 3]

deepmd.tf.utils.region.**b_to_face_distance**(cell)

deepmd.tf.utils.serialization

Module Contents

Functions

serialize_from_file(→ dict)	Serialize the model file to a dictionary.
deserialize_to_file(→ None)	Deserialize the dictionary to a model file.

deepmd.tf.utils.serialization.**serialize_from_file**(model_file: str) → dict

Serialize the model file to a dictionary.

Parameters

model_file
[str] The model file to be serialized.

Returns

dict
The serialized model data.

deepmd.tf.utils.serialization.**deserialize_to_file**(model_file: str, data: dict) → None

Deserialize the dictionary to a model file.

Parameters

model_file
: [str] The model file to be saved.

data
: [dict] The dictionary to be deserialized.

deepmd.tf.utils.sess

Module Contents

Functions

run_sess(sess, *args, **kwargs)	Run session with erorrs caught.

deepmd.tf.utils.sess.**run_sess**(sess: deepmd.tf.env.tf.Session, *args, **kwargs)

Run session with erorrs caught.

Parameters

sess
: [tf.Session] TensorFlow Session

*args
: Variable length argument list.

**kwargs
: Arbitrary keyword arguments.

Returns

Any
: the result of sess.run()

deepmd.tf.utils.spin

Module Contents

Classes

Spin	Class for spin.

class deepmd.tf.utils.spin.**Spin**(use_spin: List[bool] | None = None, spin_norm: List[float] | None = None, virtual_len: List[float] | None = None)

Class for spin.

Parameters

use_spin
: Whether to use atomic spin model for each atom type

spin_norm
: The magnitude of atomic spin for each atom type with spin

virtual_len
The distance between virtual atom representing spin and its corresponding real atom for each atom type with spin

build(reuse=None, suffix='')
Build the computational graph for the spin.

Parameters

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

embedded_types
The computational graph for embedded types

get_ntypes_spin() → int
Returns the number of atom types which contain spin.

get_use_spin() → List[bool]
Returns the list of whether to use spin for each atom type.

get_spin_norm() → List[float]
Returns the list of magnitude of atomic spin for each atom type.

get_virtual_len() → List[float]
Returns the list of distance between real atom and virtual atom for each atom type.

deepmd.tf.utils.tabulate

Module Contents

Classes

DPTabulate	Class for tabulation.

Attributes

log

deepmd.tf.utils.tabulate.log

class deepmd.tf.utils.tabulate.**DPTabulate**(descrpt: deepmd.tf.descriptor.Descriptor, neuron: List[int], graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, type_one_side: bool = False, exclude_types: List[List[int]] = [], activation_fn: Callable[[deepmd.tf.env.tf.Tensor], deepmd.tf.env.tf.Tensor] = tf.nn.tanh, suffix: str = '')

Class for tabulation.

Compress a model, which including tabulating the embedding-net. The table is composed of fifth-order polynomial coefficients and is assembled from two sub-tables. The first table takes the stride(parameter) as it's uniform stride, while the second table takes 10 * stride as it's uniform stride The range of the first table is automatically detected by deepmd-kit, while the second table ranges from the first table's upper boundary(upper) to the extrapolate(parameter) * upper.

Parameters

descrpt
: Descriptor of the original model

neuron
: Number of neurons in each hidden layers of the embedding net
mathcalN

graph
: [`tf.Graph`] The graph of the original model

graph_def
: [`tf.GraphDef`] The graph_def of the original model

type_one_side
: Try to build N_types tables. Otherwise, building N_types^2 tables

exclude_types
: [`List[List[int]]`] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

activation_function
: The activation function in the embedding net. Supported options are {"tanh","gelu"} in common.ACTIVATION_FN_DICT.

suffix
: [`str`, `optional`] The suffix of the scope

property _n_all_excluded: int

Then number of types excluding all types.

build(min_nbor_dist: float, extrapolate: float, stride0: float, stride1: float) → Tuple[Dict[str, int], Dict[str, int]]

Build the tables for model compression.

Parameters

min_nbor_dist
: The nearest distance between neighbor atoms

extrapolate
: The scale of model extrapolation

stride0
: The uniform stride of the first table

stride1
: The uniform stride of the second table

Returns

lower
: [`dict[str, int]`] The lower boundary of environment matrix by net

upper
[dict[str, int]] The upper boundary of environment matrix by net

_build_lower(net, xx, idx, upper, lower, stride0, stride1, extrapolate, nspline)

_load_sub_graph()

_get_bias()

_get_matrix()

_make_data(xx, idx)

_layer_0(x, w, b)

_layer_1(x, w, b)

_get_env_mat_range(min_nbor_dist)

_spline5_switch(xx, rmin, rmax)

_get_layer_size()

_all_excluded(ii: int) → bool
Check if type ii excluds all types.

Parameters

ii
[int] type index

Returns

bool
if type ii excluds all types

_get_table_size()

_get_data_type()

_get_last_layer_size()

_convert_numpy_to_tensor()
Convert self.data from np.ndarray to tf.Tensor.

deepmd.tf.utils.type_embed

Module Contents

Classes

TypeEmbedNet	Type embedding network.

Functions

embed_atom_type(ntypes, natoms, type_embedding)	Make the embedded type for the atoms in system.

deepmd.tf.utils.type_embed.**embed_atom_type**(ntypes: int, natoms: deepmd.tf.env.tf.Tensor, type_embedding: deepmd.tf.env.tf.Tensor)

Make the embedded type for the atoms in system. The atoms are assumed to be sorted according to the type, thus their types are described by a tf.Tensor natoms, see explanation below.

Parameters

ntypes:
Number of types.

natoms:
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

type_embedding:
The type embedding. It has the shape of [ntypes, embedding_dim]

Returns

atom_embedding
The embedded type of each atom. It has the shape of [numb_atoms, embedding_dim]

class deepmd.tf.utils.type_embed.**TypeEmbedNet**(*, ntypes: int, neuron: List[int], resnet_dt: bool = False, activation_function: str | None = 'tanh', precision: str = 'default', trainable: bool = True, seed: int | None = None, uniform_seed: bool = False, padding: bool = False, **kwargs)

Type embedding network.

Parameters

ntypes
[int] Number of atom types

neuron
[list[int]] Number of neurons in each hidden layers of the embedding net

resnet_dt
Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

activation_function
The activation function in the embedding net. Supported options are “tanh”, “gelu”, “none”, “linear”, “sigmoid”, “softplus”, “relu”, “relu6”, “gelu_tf”.

precision
The precision of the embedding net parameters. Supported options are “default”, “float64”, “float16”, “float32”.

trainable
If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

uniform_seed
: Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

padding
: Concat the zero padding to the output, as the default embedding of empty type.

build(ntypes: int, reuse=None, suffix='')

Build the computational graph for the descriptor.

Parameters

ntypes
: Number of atom types.

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

embedded_types
: The computational graph for embedded types

init_variables(graph: deepmd.tf.env.tf.Graph, graph_def: deepmd.tf.env.tf.GraphDef, suffix='', model_type='original_model') → None

Init the type embedding net variables with the given dict.

Parameters

graph
: [tf.Graph] The input frozen model graph

graph_def
: [tf.GraphDef] The input frozen model graph_def

suffix
: Name suffix to identify this descriptor

model_type
: Indicator of whether this model is a compressed model

classmethod deserialize(data: dict, suffix: str = '')

Deserialize the model.

Parameters

data
: [dict] The serialized data

suffix
: [str, optional] The suffix of the scope

Returns

Model
: The deserialized model

serialize(suffix: str = '') → dict

Serialize the model.

Parameters

suffix
[str, optional] The suffix of the scope

Returns

dict
The serialized data

deepmd.tf.utils.update_sel

Module Contents

Classes

UpdateSel	Update the sel field in the descriptor.

class deepmd.tf.utils.update_sel.**UpdateSel**

Bases: *deepmd.utils.update_sel.BaseUpdateSel*

Update the sel field in the descriptor.

property neighbor_stat: Type[*deepmd.tf.utils.neighbor_stat.NeighborStat*]

hook(min_nbor_dist, max_nbor_size)

deepmd.tf.utils.weight_avg

Alias for backward compatibility.

Module Contents

Functions

weighted_average(→ Dict)	Compute wighted average of prediction errors (MAE or RMSE) for model.

deepmd.tf.utils.weight_avg.**weighted_average**(errors: List[Dict[str, Tuple[float, float]]]) → Dict

Compute wighted average of prediction errors (MAE or RMSE) for model.

Parameters

errors
[List[Dict[str, Tuple[float, float]]]] List: the error of systems Dict: the error of quantities, name given by the key str: the name of the quantity, must starts with 'mae' or 'rmse' Tuple: (error, weight)

Returns

Dict
weighted averages

Package Contents

Classes

DeepmdData	Class for a data system.
DeepmdDataSystem	Class for manipulating many data systems.
LearningRateExp	The exponentially decaying learning rate.
PairTab	Pairwise tabulated potential.
Plugin	A class to register and restore plugins.
PluginVariant	A class to remove type from input arguments.

class deepmd.tf.utils.DeepmdData(sys_path: str, set_prefix: str = 'set', shuffle_test: bool = True, type_map: List[str] | None = None, optional_type_map: bool = True, modifier=None, trn_all_set: bool = False, sort_atoms: bool = True)

Class for a data system.

It loads data from hard disk, and mantains the data as a data_dict

Parameters

sys_path
Path to the data system

set_prefix
Prefix for the directories of different sets

shuffle_test
If the test data are shuffled

type_map
Gives the name of different atom types

optional_type_map
If the type_map.raw in each system is optional

modifier
Data modifier that has the method modify_data

trn_all_set
Use all sets as training dataset. Otherwise, if the number of sets is more than 1, the last set is left for test.

sort_atoms
[bool] Sort atoms by atom types. Required to enable when the data is directly feeded to descriptors except mixed types.

add(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: List[int] | None = None, repeat: int = 1, default: float = 0.0, dtype: numpy.dtype | None = None, output_natoms_for_type_sel: bool = False)

Add a data item that to be loaded.

Parameters

key
The key of the item. The corresponding data is stored in sys_path/set.*/key.npy

ndof
The number of dof

atomic
The item is an atomic property. If False, the size of the data should be nframes x ndof If True, the size of data should be nframes x natoms x ndof

must
The data file sys_path/set.*/key.npy must exist. If must is False and the data file does not exist, the data_dict[find_key] is set to 0.0

high_prec
Load the data and store in float64, otherwise in float32

type_sel
Select certain type of atoms

repeat
The data will be repeated repeat times.

default
[`float`, default=0.] default value of data

dtype
[`np.dtype`, `optional`] the dtype of data, overwrites high_prec if provided

output_natoms_for_type_sel
[bool, `optional`] if True and type_sel is True, the atomic dimension will be natoms instead of nsel

reduce(key_out: str, key_in: str)

Generate a new item from the reduction of another atom.

Parameters

key_out
The name of the reduced item

key_in
The name of the data item to be reduced

get_data_dict() → dict

Get the data_dict.

check_batch_size(batch_size)

Check if the system can get a batch of data with batch_size frames.

check_test_size(test_size)

Check if the system can get a test dataset with test_size frames.

get_item_torch(index: int) → dict

Get a single frame data . The frame is picked from the data system by index. The index is coded across all the sets.

Parameters

index
index of the frame

get_batch(batch_size: int) → dict

Get a batch of data with batch_size frames. The frames are randomly picked from the data system.

Parameters

batch_size
size of the batch

get_test(ntests: int = -1) → dict

Get the test data with ntests frames.

Parameters

ntests
Size of the test data set. If ntests is -1, all test data will be get.

get_ntypes() → int

Number of atom types in the system.

get_type_map() → List[str]

Get the type map.

get_atom_type() → List[int]

Get atom types.

get_numb_set() → int

Get number of training sets.

get_numb_batch(batch_size: int, set_idx: int) → int

Get the number of batches in a set.

get_sys_numb_batch(batch_size: int) → int

Get the number of batches in the data system.

get_natoms()

Get number of atoms.

get_natoms_vec(ntypes: int)

Get number of atoms and number of atoms in different types.

Parameters

ntypes
Number of types (may be larger than the actual number of types in the system).

Returns

natoms
natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

avg(key)

Return the average value of an item.

_idx_map_sel(atom_type, type_sel)

_get_natoms_2(ntypes)

_get_subdata(data, idx=None)

_load_batch_set(set_name: deepmd.utils.path.DPPath)

reset_get_batch()

_load_test_set(set_name: deepmd.utils.path.DPPath, shuffle_test)

_shuffle_data(data)

_get_nframes(set_name: deepmd.utils.path.DPPath)

reformat_data_torch(data)

Modify the data format for the requirements of Torch backend.

Parameters

data
original data

_load_set(set_name: deepmd.utils.path.DPPath)

_load_data(set_name, key, nframes, ndof_, atomic=False, must=True, repeat=1, high_prec=False, type_sel=None, default: float = 0.0, dtype: numpy.dtype | None = None, output_natoms_for_type_sel: bool = False)

_load_type(sys_path: deepmd.utils.path.DPPath)

_load_type_mix(set_name: deepmd.utils.path.DPPath)

_make_idx_map(atom_type)

_load_type_map(sys_path: deepmd.utils.path.DPPath)

_check_pbc(sys_path: deepmd.utils.path.DPPath)

_check_mode(set_path: deepmd.utils.path.DPPath)

class deepmd.tf.utils.DeepmdDataSystem(systems: List[str], batch_size: int, test_size: int, rcut: float | None = None, set_prefix: str = 'set', shuffle_test: bool = True, type_map: List[str] | None = None, optional_type_map: bool = True, modifier=None, trn_all_set=False, sys_probs=None, auto_prob_style='prob_sys_size', sort_atoms: bool = True)

Class for manipulating many data systems.

It is implemented with the help of DeepmdData

property default_mesh: List[numpy.ndarray]

Mesh for each system.

_load_test(ntests=-1)

compute_energy_shift(rcond=None, key='energy')

add_dict(adict: dict) → None

Add items to the data system by a dict. adict should have items like .. code-block:: python.

adict[key] = {
"ndof": ndof, "atomic": atomic, "must": must, "high_prec": high_prec, "type_sel": type_sel, "repeat": repeat,

}

For the explaination of the keys see add

add(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: List[int] | None = None, repeat: int = 1, default: float = 0.0, dtype: numpy.dtype | None = None, output_natoms_for_type_sel: bool = False)

Add a data item that to be loaded.

Parameters

key
The key of the item. The corresponding data is stored in sys_path/set.*/key.npy

ndof
The number of dof

atomic
The item is an atomic property. If False, the size of the data should be nframes x ndof If True, the size of data should be nframes x natoms x ndof

must
The data file sys_path/set.*/key.npy must exist. If must is False and the data file does not exist, the data_dict[find_key] is set to 0.0

high_prec
Load the data and store in float64, otherwise in float32

type_sel
Select certain type of atoms

repeat
The data will be repeated repeat times.

default, default=0.
Default value of data

dtype
The dtype of data, overwrites high_prec if provided

output_natoms_for_type_sel
[bool] If True and type_sel is True, the atomic dimension will be natoms instead of nsel

reduce(key_out, key_in)

Generate a new item from the reduction of another atom.

Parameters

key_out
The name of the reduced item

key_in
The name of the data item to be reduced

get_data_dict(ii: int = 0) → dict

set_sys_probs(sys_probs=None, auto_prob_style: str = 'prob_sys_size')

get_batch(sys_idx: int | None = None) → dict

Get a batch of data from the data systems.

Parameters

sys_idx
[int] The index of system from which the batch is get. If sys_idx is not None,

sys_probs and auto_prob_style are ignored If sys_idx is None, automatically determine the system according to sys_probs or auto_prob_style, see the following. This option does not work for mixed systems.

Returns

dict
The batch data

get_batch_standard(sys_idx: int | None = None) → dict

Get a batch of data from the data systems in the standard way.

Parameters

sys_idx
[int] The index of system from which the batch is get. If sys_idx is not None, sys_probs and auto_prob_style are ignored If sys_idx is None, automatically determine the system according to sys_probs or auto_prob_style, see the following.

Returns

dict
The batch data

get_batch_mixed() → dict

Get a batch of data from the data systems in the mixed way.

Returns

dict
The batch data

_merge_batch_data(batch_data: List[dict]) → dict

Merge batch data from different systems.

Parameters

batch_data
[list of dict] A list of batch data from different systems.

Returns

dict
The merged batch data.

get_test(sys_idx: int | None = None, n_test: int = -1)

Get test data from the the data systems.

Parameters

sys_idx
The test dat of system with index sys_idx will be returned. If is None, the currently selected system will be returned.

n_test
Number of test data. If set to -1 all test data will be get.

get_sys_ntest(sys_idx=None)

Get number of tests for the currently selected system, or one defined by sys_idx.

get_type_map() → List[str]

Get the type map.

get_nbatches() → int

Get the total number of batches.

get_ntypes() → int

Get the number of types.

get_nsystems() → int

Get the number of data systems.

get_sys(idx: int) → deepmd.utils.data.DeepmdData

Get a certain data system.

get_batch_size() → int

Get the batch size.

print_summary(name: str)

_make_auto_bs(rule)

_make_auto_ts(percent)

_check_type_map_consistency(type_map_list)

class deepmd.tf.utils.LearningRateExp(start_lr: float, stop_lr: float = 5e-08, decay_steps: int = 5000, decay_rate: float = 0.95)

The exponentially decaying learning rate.

The learning rate at step t is given by

$$\alpha(t) = \alpha_0 \lambda^{t/\tau}$$

where α is the learning rate, α_0 is the starting learning rate, λ is the decay rate, and τ is the decay steps.

Parameters

start_lr
Starting learning rate α_0

stop_lr
Stop learning rate α_1

decay_steps
Learning rate decay every this number of steps τ

decay_rate
The decay rate λ. If stop_step is provided in build, then it will be determined automatically and overwritten.

build(global_step: deepmd.tf.env.tf.Tensor, stop_step: int | None = None) → deepmd.tf.env.tf.Tensor

Build the learning rate.

Parameters

global_step
The tf Tensor prividing the global training step

stop_step
The stop step. If provided, the decay_rate will be determined automatically and overwritten.

Returns

learning_rate
The learning rate

start_lr() → float

Get the start lr.

value(step: int) → float

Get the lr at a certain step.

class deepmd.tf.utils.PairTab(filename: str, rcut: float | None = None)

Pairwise tabulated potential.

Parameters

filename
File name for the short-range tabulated potential. The table is a text data file with (N_t + 1) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

reinit(filename: str, rcut: float | None = None) → None

Initialize the tabulated interaction.

Parameters

filename
File name for the short-range tabulated potential. The table is a text data file with (N_t + 1) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

serialize() → dict

classmethod deserialize(data) → PairTab

_check_table_upper_boundary() → None

Update User Provided Table Based on rcut.

This function checks the upper boundary provided in the table against rcut. If the table upper boundary values decay to zero before rcut, padding zeros will be added to the table to cover rcut; if the table upper boundary values do not decay to zero before ruct, extrapolation will be performed till rcut.

Examples

table = [[0.005 1. 2. 3.]
[0.01 0.8 1.6 2.4] [0.015 0. 1. 1.5]]

rcut = 0.022

new_table = [[0.005 1. 2. 3.]
[0.01 0.8 1.6 2.4] [0.015 0. 1. 1.5] [0.02 0. 0. 0.]

table = [[0.005 1. 2. 3.]
[0.01 0.8 1.6 2.4] [0.015 0.5 1. 1.5] [0.02 0.25 0.4 0.75] [0.025 0. 0.1 0.] [0.03 0. 0. 0.]]

rcut = 0.031

new_table = [[0.005 1. 2. 3.]
[0.01 0.8 1.6 2.4] [0.015 0.5 1. 1.5] [0.02 0.25 0.4 0.75] [0.025 0. 0.1 0.] [0.03 0. 0. 0.] [0.035 0. 0. 0.]]

get() → Tuple[numpy.array, numpy.array]

Get the serialized table.

_extrapolate_table(pad_extrapolation: numpy.array) → numpy.array

Soomth extrapolation between table upper boundary and rcut.

This method should only be used when the table upper boundary rmax is smaller than rcut, and the table upper boundary values are not zeros. To simplify the problem, we use a single cubic spline between rmax and rcut for each pair of atom types. One can substitute this extrapolation to higher order polynomials if needed.

There are two scenarios:

1.ruct - rmax >= hh:
Set values at the grid point right before rcut to 0, and perform exterapolation between the grid point and rmax, this allows smooth decay to 0 at rcut.

2.rcut - rmax < hh:
Set values at rmax + hh to 0, and perform extrapolation between rmax and rmax + hh.

Parameters

pad_extrapolation
[**np.array**] The emepty grid that holds the extrapolation values.

Returns

np.array
The cubic spline extrapolation.

_make_data()

class deepmd.tf.utils.Plugin

A class to register and restore plugins.

Examples

```
>>> plugin = Plugin()
>>> @plugin.register("xx")
    def xxx():
        pass
>>> print(plugin.plugins["xx"])
```

Attributes

plugins
[Dict[str, object]] plugins

__add__(other) → Plugin

register(key: str) → Callable[[object], object]

Register a plugin.

Parameters

key
[str] key of the plugin

Returns

Callable[[object], object]
decorator

get_plugin(key) → object

Visit a plugin by key.

Parameters

key
[str] key of the plugin

Returns

object
the plugin

class deepmd.tf.utils.PluginVariant

A class to remove type from input arguments.

Submodules

deepmd.tf.__about__

Module Contents

deepmd.tf.__about__.__version__ = 'unknown'

deepmd.tf.__main__

Package dp entry point.

deepmd.tf.calculator

Module Contents

Classes

DP	Implementation of ASE deepmd calculator.

class deepmd.tf.calculator.DP(model: str | pathlib.Path, label: str = 'DP', type_dict: Dict[str, int] | None = None, neighbor_list=None, **kwargs)

Bases: ase.calculators.calculator.Calculator

Implementation of ASE deepmd calculator.

Implemented propertie are energy, forces and stress

Parameters

model
[Union[str, Path]] path to the model

label
[str, optional] calculator label, by default "DP"

type_dict
[Dict[str, int], optional] mapping of element types and their numbers, best left None and the calculator will infer this information from model, by default None

neighbor_list
[ase.neighborlist.NeighborList, optional] The neighbor list object. If None, then build the native neighbor list.

Examples

Compute potential energy

```
>>> from ase import Atoms
>>> from deepmd.tf.calculator import DP
>>> water = Atoms('H2O',
>>>               positions=[(0.7601, 1.9270, 1),
>>>                          (1.9575, 1, 1),
>>>                          (1., 1., 1.)],
>>>               cell=[100, 100, 100],
>>>               calculator=DP(model="frozen_model.pb"))
>>> print(water.get_potential_energy())
>>> print(water.get_forces())
```

Run BFGS structure optimization

```
>>> from ase.optimize import BFGS
>>> dyn = BFGS(water)
>>> dyn.run(fmax=1e-6)
>>> print(water.get_positions())
```

name = 'DP'

implemented_properties: ClassVar[List[str]] = ['energy', 'free_energy', 'forces', 'virial', 'stress']

calculate(atoms: ase.Atoms | None = None, properties: List[str] = ['energy', 'forces', 'virial'], system_changes: List[str] = all_changes)

Run calculation with deepmd model.

Parameters

atoms
: [Optional[Atoms], optional] atoms object to run the calculation on, by default None

properties
: [List[str], optional] unused, only for function signature compatibility, by default ["energy", "forces", "stress"]

system_changes
: [List[str], optional] unused, only for function signature compatibility, by default all_changes

deepmd.tf.common

Collection of functions and classes used throughout the whole package.

Module Contents

Functions

add_data_requirement(key, ndof[, atomic, must, ...])	Specify data requirements for training.
expand_sys_str(→ List[str])	Recursively iterate over directories taking those that contain type.raw file.
get_np_precision(→ numpy.dtype)	Get numpy precision constant from string.
j_loader(→ Dict[str, Any])	Load yaml or json settings file.
j_must_have(→ _DICT_VAL)	Assert that supplied dictionary conaines specified key.
make_default_mesh(→ numpy.ndarray)	Make mesh.
select_idx_map(→ numpy.ndarray)	Build map of indices for element supplied element types from all atoms list.
gelu(→ deepmd.tf.env.tf.Tensor)	Gaussian Error Linear Unit.
gelu_tf(→ deepmd.tf.env.tf.Tensor)	Gaussian Error Linear Unit.
get_activation_func(...)	Get activation function callable based on string name.
get_precision(→ Any)	Convert str to TF DType constant.
safe_cast_tensor(→ deepmd.tf.env.tf.Tensor)	Convert a Tensor from a precision to another precision.
cast_precision(→ Callable)	A decorator that casts and casts back the input
clear_session()	Reset all state generated by DeePMD-kit.

Attributes

data_requirement
PRECISION_DICT
ACTIVATION_FN_DICT

deepmd.tf.common.**add_data_requirement**(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: bool | None = None, repeat: int = 1, default: float = 0.0, dtype: numpy.dtype | None = None, output_natoms_for_type_sel: bool = False)

Specify data requirements for training.

Parameters

key
: [str] type of data stored in corresponding *.npy file e.g. forces or energy

ndof
: [int] number of the degrees of freedom, this is tied to atomic parameter e.g. forces have atomic=True and ndof=3

atomic
: [bool, optional] specifies whwther the ndof keyworrd applies to per atom quantity or not, by default False

must
: [bool, optional] specifi if the *.npy data file must exist, by default False

high_prec
: [bool, optional] if true load data to np.float64 else np.float32, by default False

type_sel
: [bool, optional] select only certain type of atoms, by default None

repeat
: [int, optional] if specify repaeat data repeat times, by default 1

default
: [float, optional, default=0.] default value of data

dtype
: [np.dtype, optional] the dtype of data, overwrites high_prec if provided

output_natoms_for_type_sel
: [bool, optional] if True and type_sel is True, the atomic dimension will be natoms instead of nsel

deepmd.tf.common.**data_requirement**

deepmd.tf.common.**expand_sys_str**(root_dir: str | pathlib.Path) → List[str]

Recursively iterate over directories taking those that contain type.raw file.

Parameters

root_dir
: [Union[str, Path]] starting directory

Returns

List[str]
list of string pointing to system directories

deepmd.tf.common.get_np_precision(precision: _PRECISION) → numpy.dtype

Get numpy precision constant from string.

Parameters

precision
[_PRECISION] string name of numpy constant or default

Returns

np.dtype
numpy presicion constant

Raises

RuntimeError
if string is invalid

deepmd.tf.common.j_loader(filename: str | pathlib.Path) → Dict[str, Any]

Load yaml or json settings file.

Parameters

filename
[Union[str, Path]] path to file

Returns

Dict[str, Any]
loaded dictionary

Raises

TypeError
if the supplied file is of unsupported type

deepmd.tf.common.j_must_have(jdata: Dict[str, _DICT_VAL], key: str, deprecated_key: List[str] = []) → _DICT_VAL

Assert that supplied dictionary conaines specified key.

Returns

_DICT_VAL
value that was store unde supplied key

Raises

RuntimeError
if the key is not present

deepmd.tf.common.make_default_mesh(pbc: bool, mixed_type: bool) → numpy.ndarray

Make mesh.

Only the size of mesh matters, not the values: * 6 for PBC, no mixed types * 0 for no PBC, no mixed types * 7 for PBC, mixed types * 1 for no PBC, mixed types

Parameters

pbc
[bool] if True, the mesh will be made for periodic boundary conditions

mixed_type
[bool] if True, the mesh will be made for mixed types

Returns

np.ndarray
mesh

deepmd.tf.common.select_idx_map(atom_types: numpy.ndarray, select_types: numpy.ndarray) → numpy.ndarray

Build map of indices for element supplied element types from all atoms list.

Parameters

atom_types
[np.ndarray] array specifing type for each atoms as integer

select_types
[np.ndarray] types of atoms you want to find indices for

Returns

np.ndarray
indices of types of atoms defined by select_types in atom_types array

Warning: select_types array will be sorted before finding indices in atom_types

deepmd.tf.common.PRECISION_DICT

deepmd.tf.common.gelu(x: deepmd.tf.env.tf.Tensor) → deepmd.tf.env.tf.Tensor

Gaussian Error Linear Unit.

This is a smoother version of the RELU, implemented by custom operator.

Parameters

x
[tf.Tensor] float Tensor to perform activation

Returns

tf.Tensor
x with the GELU activation applied

References

Original paper https://arxiv.org/abs/1606.08415

deepmd.tf.common.gelu_tf(x: deepmd.tf.env.tf.Tensor) → deepmd.tf.env.tf.Tensor

Gaussian Error Linear Unit.

This is a smoother version of the RELU, implemented by TF.

Parameters

x
[tf.Tensor] float Tensor to perform activation

Returns

tf.Tensor
x with the GELU activation applied

References

Original paper https://arxiv.org/abs/1606.08415

deepmd.tf.common.**ACTIVATION_FN_DICT**

deepmd.tf.common.**get_activation_func**(activation_fn: deepmd.common._ACTIVATION | None) → Callable[[deepmd.tf.env.tf.Tensor], deepmd.tf.env.tf.Tensor]

Get activation function callable based on string name.

Parameters

activation_fn
[_ACTIVATION] one of the defined activation functions

Returns

Callable[[tf.Tensor], tf.Tensor]
correspondingg TF callable

Raises

RuntimeError
if unknown activation function is specified

deepmd.tf.common.**get_precision**(precision: deepmd.common._PRECISION) → Any

Convert str to TF DType constant.

Parameters

precision
[_PRECISION] one of the allowed precisions

Returns

tf.python.framework.dtypes.DType
appropriate TF constant

Raises

RuntimeError
if supplied precision string does not have acorresponding TF constant

deepmd.tf.common.**safe_cast_tensor**(input: deepmd.tf.env.tf.Tensor, from_precision: deepmd.tf.env.tf.DType, to_precision: deepmd.tf.env.tf.DType) → deepmd.tf.env.tf.Tensor

Convert a Tensor from a precision to another precision.

If input is not a Tensor or without the specific precision, the method will not cast it.

Parameters

input
[tf.Tensor] input tensor

from_precision
[tf.DType] Tensor data type that is casted from

to_precision
[tf.DType] Tensor data type that casts to

Returns

tf.Tensor
casted Tensor

deepmd.tf.common.cast_precision(func: Callable) → Callable

A decorator that casts and casts back the input and output tensor of a method.

The decorator should be used in a classmethod.

The decorator will do the following thing: (1) It casts input Tensors from GLOBAL_TF_FLOAT_PRECISION to precision defined by property precision. (2) It casts output Tensors from precision to GLOBAL_TF_FLOAT_PRECISION. (3) It checks inputs and outputs and only casts when input or output is a Tensor and its dtype matches GLOBAL_TF_FLOAT_PRECISION and precision, respectively. If it does not match (e.g. it is an integer), the decorator will do nothing on it.

Returns

Callable
a decorator that casts and casts back the input and output tensor of a method

Examples

```
>>> class A:
...     @property
...     def precision(self):
...         return tf.float32
...
...     @cast_precision
...     def f(x: tf.Tensor, y: tf.Tensor) -> tf.Tensor:
...         return x**2 + y
```

deepmd.tf.common.clear_session()

Reset all state generated by DeePMD-kit.

deepmd.tf.env

Module that sets tensorflow working environment and exports inportant constants.

Module Contents

Functions

reset_default_tf_session_config(cpu_only)	Limit tensorflow session to CPU or not.
global_cvt_2_tf_float(→ tensorflow.compat.v1.Tensor)	Cast tensor to globally set TF precision.
global_cvt_2_ener_float(→ tensorflow.compat.v1.Tensor)	Cast tensor to globally set energy precision.

Attributes

GLOBAL_CONFIG
GLOBAL_NP_FLOAT_PRECISION
SHARED_LIB_DIR
SHARED_LIB_MODULE
tf_py_version
EMBEDDING_NET_PATTERN
FITTING_NET_PATTERN
TYPE_EMBEDDING_PATTERN
ATTENTION_LAYER_PATTERN
TRANSFER_PATTERN
REMOVE_SUFFIX_DICT
default_tf_session_config
MODEL_VERSION
TF_VERSION
op_module
op_grads_module
GLOBAL_TF_FLOAT_PRECISION

deepmd.tf.env.**GLOBAL_CONFIG**

deepmd.tf.env.**GLOBAL_NP_FLOAT_PRECISION**

deepmd.tf.env.**SHARED_LIB_DIR**

deepmd.tf.env.**SHARED_LIB_MODULE** = 'lib'

deepmd.tf.env.**tf_py_version**

deepmd.tf.env.**EMBEDDING_NET_PATTERN**

deepmd.tf.env.**FITTING_NET_PATTERN**

deepmd.tf.env.**TYPE_EMBEDDING_PATTERN**

deepmd.tf.env.**ATTENTION_LAYER_PATTERN**

deepmd.tf.env.**TRANSFER_PATTERN**

deepmd.tf.env.**REMOVE_SUFFIX_DICT**

deepmd.tf.env.**default_tf_session_config**

deepmd.tf.env.**reset_default_tf_session_config**(cpu_only: bool)

Limit tensorflow session to CPU or not.

Parameters

cpu_only
[bool] If enabled, no GPU device is visible to the TensorFlow Session.

deepmd.tf.env.**MODEL_VERSION**

deepmd.tf.env.**TF_VERSION**

deepmd.tf.env.**op_module**

deepmd.tf.env.**op_grads_module**

deepmd.tf.env.**GLOBAL_TF_FLOAT_PRECISION**

deepmd.tf.env.**global_cvt_2_tf_float**(xx: tensorflow.compat.v1.Tensor) → tensorflow.compat.v1.Tensor

Cast tensor to globally set TF precision.

Parameters

xx
[tf.Tensor] input tensor

Returns

tf.Tensor
output tensor cast to GLOBAL_TF_FLOAT_PRECISION

deepmd.tf.env.**global_cvt_2_ener_float**(xx: tensorflow.compat.v1.Tensor) → tensorflow.compat.v1.Tensor

Cast tensor to globally set energy precision.

Parameters

xx
[tf.Tensor] input tensor

Returns

tf.Tensor
output tensor cast to GLOBAL_ENER_FLOAT_PRECISION

deepmd.tf.lmp

Register entry points for lammps-wheel.

Module Contents

Functions

get_env(→ str)	Get the environment variable from given paths.
get_library_path(→ List[str])	Get library path from a module.
get_op_dir(→ str)	Get the directory of the deepmd-kit OP library.

Attributes

find_libpython
lib_env
preload_env
tf_dir
op_dir
cuda_library_paths
libpython

deepmd.tf.lmp.**find_libpython**

deepmd.tf.lmp.**get_env**(paths: List[str | None]) → str

Get the environment variable from given paths.

deepmd.tf.lmp.**get_library_path**(module: str, filename: str) → List[str]

Get library path from a module.

Parameters

module
[str] The module name.

filename
[str] The library filename pattern.

Returns

list[str]
The library path.

deepmd.tf.lmp.**lib_env** = 'LD_LIBRARY_PATH'

deepmd.tf.lmp.**preload_env = 'LD_PRELOAD'**

deepmd.tf.lmp.**tf_dir**

deepmd.tf.lmp.**op_dir**

deepmd.tf.lmp.**cuda_library_paths = []**

deepmd.tf.lmp.**libpython**

deepmd.tf.lmp.**get_op_dir**() → str

Get the directory of the deepmd-kit OP library.

Package Contents

Classes

DeepEval	TensorFlow backend implementation for DeepEval.
DipoleChargeModifier	Parameters

Functions

DeepPotential(→ deep_eval.DeepEval)	Factory function that forwards to DeepEval (for compatbility).

class deepmd.tf.DeepEval(model_file: pathlib.Path, output_def: deepmd.dpmodel.output_def.ModelOutputDef, *args: list, load_prefix: str = 'load', default_tf_graph: bool = False, auto_batch_size: bool | int | deepmd.tf.utils.batch_size.AutoBatchSize = False, input_map: dict | None = None, neighbor_list=None, **kwargs: dict)

Bases: *deepmd.infer.deep_eval.DeepEvalBackend*

TensorFlow backend implementation for DeepEval.

Parameters

model_file
[Path] The name of the frozen model file.

output_def
[ModelOutputDef] The output definition of the model.

*args
[list] Positional arguments.

load_prefix: str
The prefix in the load computational graph

default_tf_graph
[bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

auto_batch_size
: [bool or `int` or `AutomaticBatchSize`, default: `False`] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

input_map
: [`dict`, `optional`] The input map for tf.import_graph_def. Only work with default tf graph

neighbor_list
: [`ase.neighborlist.NewPrimitiveNeighborList`, `optional`] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

**kwargs
: [`dict`] Keyword arguments.

property model_type: *deepmd.infer.deep_eval.DeepEval*

Get type of model.

:type:str

property model_version: `str`

Get version of model.

Returns

`str`
: version of model

property sess: `deepmd.tf.env.tf.Session`

Get TF session.

_init_tensors()

_init_attr()

_graph_compatable() → bool

Check the model compatability.

Returns

bool
: If the model stored in the graph file is compatable with the current code

_get_tensor(tensor_name: str) → deepmd.tf.env.tf.Tensor

Get TF graph tensor.

Parameters

tensor_name
: [`str`] name of tensor to get

Returns

`tf.Tensor`
: loaded tensor

static _load_graph(frozen_graph_filename: pathlib.Path, prefix: str = 'load', default_tf_graph: bool = False, input_map: dict | None = None)

static sort_input(coord: numpy.ndarray, atom_type: numpy.ndarray, sel_atoms: List[int] | None = None)

Sort atoms in the system according their types.

Parameters

coord
The coordinates of atoms. Should be of shape [nframes, natoms, 3]

atom_type
The type of atoms Should be of shape [natoms]

sel_atoms
The selected atoms by type

Returns

coord_out
The coordinates after sorting

atom_type_out
The atom types after sorting

idx_map
The index mapping from the input to the output. For example coord_out = coord[:,idx_map,:]

sel_atom_type
Only output if sel_atoms is not None The sorted selected atom types

sel_idx_map
Only output if sel_atoms is not None The index mapping from the selected atoms to sorted selected atoms.

static reverse_map(vec: numpy.ndarray, imap: List[int]) → numpy.ndarray

Reverse mapping of a vector according to the index map.

Parameters

vec
Input vector. Be of shape [nframes, natoms, -1]

imap
Index map. Be of shape [natoms]

Returns

vec_out
Reverse mapped vector.

make_natoms_vec(atom_types: numpy.ndarray) → numpy.ndarray

Make the natom vector used by deepmd-kit.

Parameters

atom_types
The type of atoms

Returns

natoms
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

eval_typeebd() → numpy.ndarray

Evaluate output of type embedding network by using this model.

Returns

np.ndarray
The output of type embedding network. The shape is [ntypes, o_size], where ntypes is the number of types, and o_size is the number of nodes in the output layer.

Raises

KeyError
If the model does not enable type embedding.

See also:

deepmd.tf.utils.type_embed.TypeEmbedNet
The type embedding network.

Examples

Get the output of type embedding network of graph.pb:

```
>>> from deepmd.tf.infer import DeepPotential
>>> dp = DeepPotential("graph.pb")
>>> dp.eval_typeebd()
```

build_neighbor_list(coords: numpy.ndarray, cell: numpy.ndarray | None, atype: numpy.ndarray, imap: numpy.ndarray, neighbor_list)

Make the mesh with neighbor list for a single frame.

Parameters

coords
[np.ndarray] The coordinates of atoms. Should be of shape [natoms, 3]

cell
[Optional[np.ndarray]] The cell of the system. Should be of shape [3, 3]

atype
[np.ndarray] The type of atoms. Should be of shape [natoms]

imap
[np.ndarray] The index map of atoms. Should be of shape [natoms]

neighbor_list
[ase.neighborlist.NewPrimitiveNeighborList] ASE neighbor list. The following method or attribute will be used/set: bothways, self_interaction, update, build, first_neigh, pair_second, offset_vec.

Returns

natoms_vec
[np.ndarray] The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: nloc natoms[1]: nall natoms[i]: 2 <= i < Ntypes+2, number of type i atoms for nloc

coords
[np.ndarray] The coordinates of atoms, including ghost atoms. Should be of shape [nframes, nall, 3]

atype
[np.ndarray] The type of atoms, including ghost atoms. Should be of shape [nall]

mesh
[np.ndarray] The mesh in nei_mode=4.

imap
[np.ndarray] The index map of atoms. Should be of shape [nall]

ghost_map
[np.ndarray] The index map of ghost atoms. Should be of shape [nghost]

get_ntypes() → int
Get the number of atom types of this model.

get_ntypes_spin() → int
Get the number of spin atom types of this model.

get_rcut() → float
Get the cut-off radius of this model.

get_type_map() → List[str]
Get the type map (element name of the atom types) of this model.

get_sel_type() → numpy.ndarray | None
Get the selected atom types of this model.

Only atoms with selected atom types have atomic contribution to the result of the model. If returning an empty list, all atom types are selected.

get_dim_fparam() → int
Get the number (dimension) of frame parameters of this DP.

get_dim_aparam() → int
Get the number (dimension) of atomic parameters of this DP.

_eval_func(inner_func: Callable, numb_test: int, natoms: int) → Callable
Wrapper method with auto batch size.

Parameters

inner_func
[Callable] the method to be wrapped

numb_test
[int] number of tests

natoms
[int] number of atoms

Returns

Callable
the wrapper

_get_natoms_and_nframes(coords: numpy.ndarray, atom_types: List[int] | numpy.ndarray) → Tuple[int, int]

eval(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: numpy.ndarray, atomic: bool = False, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, efield: numpy.ndarray | None = None, **kwargs: Dict[str, Any]) → Dict[str, numpy.ndarray]
Evaluate the energy, force and virial by using this DP.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

atomic
Calculate the atomic energy and virial

fparam
The frame parameter. The array can be of size : - nframes x dim_fparam. - dim_fparam. Then all frames are assumed to be provided with the same fparam.

aparam
The atomic parameter The array can be of size : - nframes x natoms x dim_aparam. - natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. - dim_aparam. Then all frames and atoms are provided with the same aparam.

efield
The external field on atoms. The array should be of size nframes x natoms x 3

**kwargs
Other parameters

Returns

output_dict
[dict] The output of the evaluation. The keys are the names of the output variables, and the values are the corresponding output arrays.

_prepare_feed_dict(coords, cells, atom_types, fparam=None, aparam=None, efield=None)

_eval_inner(coords, cells, atom_types, fparam=None, aparam=None, efield=None, **kwargs)

_get_output_shape(odef, nframes, natoms)

eval_descriptor(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: numpy.ndarray, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, efield: numpy.ndarray | None = None) → numpy.ndarray

Evaluate descriptors by using this DP.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

fparam
The frame parameter. The array can be of size : - nframes x dim_fparam. - dim_fparam. Then all frames are assumed to be provided with the same fparam.

aparam
: The atomic parameter The array can be of size : - nframes x natoms x dim_aparam. - natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. - dim_aparam. Then all frames and atoms are provided with the same aparam.

efield
: The external field on atoms. The array should be of size nframes x natoms x 3

Returns

descriptor
: Descriptors.

_eval_descriptor_inner(coords: numpy.ndarray, cells: numpy.ndarray, atom_types: numpy.ndarray, fparam: numpy.ndarray | None = None, aparam: numpy.ndarray | None = None, efield: numpy.ndarray | None = None) → numpy.ndarray

get_numb_dos() → int
: Get the number of DOS.

get_has_efield() → bool
: Check if the model has efield.

deepmd.tf.**DeepPotential**(*args, **kwargs) → deep_eval.DeepEval
: Factory function that forwards to DeepEval (for compatbility).

Parameters

*args
: positional arguments

**kwargs
: keyword arguments

Returns

DeepEval
: potentials

class deepmd.tf.**DipoleChargeModifier**(model_name: str, model_charge_map: List[float], sys_charge_map: List[float], ewald_h: float = 1, ewald_beta: float = 1)

Bases: deepmd.tf.infer.deep_dipole.DeepDipoleOld

Parameters

model_name
: The model file for the DeepDipole model

model_charge_map
: Gives the amount of charge for the wfcc

sys_charge_map
: Gives the amount of charge for the real atoms

ewald_h
: Grid spacing of the reciprocal part of Ewald sum. Unit: A

ewald_beta
: Splitting parameter of the Ewald sum. Unit: A^{-1}

build_fv_graph() → deepmd.tf.env.tf.Tensor

Build the computational graph for the force and virial inference.

_build_fv_graph_inner()

_enrich(dipole, dof=3)

_slice_descrpt_deriv(deriv)

eval(coord: numpy.ndarray, box: numpy.ndarray, atype: numpy.ndarray, eval_fv: bool = True) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Evaluate the modification.

Parameters

coord
The coordinates of atoms

box
The simulation region. PBC is assumed

atype
The atom types

eval_fv
Evaluate force and virial

Returns

tot_e
The energy modification

tot_f
The force modification

tot_v
The virial modification

_eval_fv(coords, cells, atom_types, ext_f)

_extend_system(coord, box, atype, charge)

modify_data(data: dict, data_sys: deepmd.tf.utils.data.DeepmdData) → None

Modify data.

Parameters

data
Internal data of DeepmdData. Be a dict, has the following keys - coord coordinates - box simulation box - type atom types - find_energy tells if data has energy - find_force tells if data has force - find_virial tells if data has virial - energy energy - force force - virial virial

data_sys
[**DeepmdData**] The data system.

20.1.8 deepmd.utils

Submodules

deepmd.utils.argcheck

Module Contents

Classes

ArgsPlugin	

Functions

list_to_doc(xx)	
make_link(content, ref_key)	
deprecate_argument_extra_check(→ Callable[[dict], bool])	Generate an extra check to deprecate an argument in sub fields.
type_embedding_args()	
spin_args()	
descrpt_local_frame_args()	
descrpt_se_a_args()	
descrpt_se_t_args()	
descrpt_se_a_tpe_args()	
descrpt_se_r_args()	
descrpt_hybrid_args()	
descrpt_se_atten_common_args()	
descrpt_se_atten_args()	
descrpt_se_atten_v2_args()	
descrpt_dpa2_args()	
descrpt_se_a_ebd_v2_args()	

continues on next page

Table 2 – continued from previous page

descrpt_se_a_mask_args()
descrpt_variant_type_args(→ dargs.Variant)
fitting_ener()
fitting_dos()
fitting_polar()
fitting_dipole()
fitting_variant_type_args()
modifier_dipole_charge()
modifier_variant_type_args()
model_compression()
model_compression_type_args()
model_args([exclude_hybrid])
standard_model_args(→ dargs.Argument)
multi_model_args(→ dargs.Argument)
pairwise_dprc(→ dargs.Argument)
frozen_model_args(→ dargs.Argument)
pairtab_model_args(→ dargs.Argument)
linear_ener_model_args(→ dargs.Argument)
learning_rate_exp()
learning_rate_variant_type_args()
learning_rate_args()
learning_rate_dict_args()
start_pref(item[, label, abbr])
limit_pref(item)
loss_ener()
loss_ener_spin()

continues on next page

Table 2 – continued from previous page

loss_dos()	
loss_tensor()	
loss_variant_type_args()	
loss_args()	
loss_dict_args()	
training_data_args()	
validation_data_args()	
mixed_precision_args()	
training_args()	
make_index(keys)	
gen_doc(*[, make_anchor, make_link])	
gen_json(**kwargs)	
gen_args(→ List[dargs.Argument])	
normalize_multi_task(data)	
normalize_fitting_net_dict(fitting_net_dict)	
normalize_data_dict(data_dict)	
normalize_loss_dict(fitting_keys, loss_dict)	
normalize_learning_rate_dict(fitting_keys, ...)	
normalize_learning_rate_dict_with_single_le	
normalize_fitting_weight(fitting_keys, data_keys[, ...])	
normalize(data)	

Attributes

log
ACTIVATION_FN_DICT
PRECISION_DICT
doc_only_tf_supported
doc_only_pt_supported
descrpt_args_plugin
fitting_args_plugin
loss_args_plugin

deepmd.utils.argcheck.**log**

deepmd.utils.argcheck.**ACTIVATION_FN_DICT**

deepmd.utils.argcheck.**PRECISION_DICT**

deepmd.utils.argcheck.**doc_only_tf_supported = '(Supported Backend: TensorFlow) '**

deepmd.utils.argcheck.**doc_only_pt_supported = '(Supported Backend: PyTorch) '**

deepmd.utils.argcheck.**list_to_doc**(xx)

deepmd.utils.argcheck.**make_link**(content, ref_key)

deepmd.utils.argcheck.**deprecate_argument_extra_check**(key: str) → Callable[[dict], bool]

Generate an extra check to deprecate an argument in sub fields.

Parameters

key
[str] The name of the deprecated argument.

deepmd.utils.argcheck.**type_embedding_args**()

deepmd.utils.argcheck.**spin_args**()

class deepmd.utils.argcheck.**ArgsPlugin**

register(name: str, alias: List[str] | None = None, doc: str = '') → Callable[[], List[dargs.Argument]]

Register a descriptor argument plugin.

Parameters

name
[str] the name of a descriptor

alias
[List[str], optional] the list of aliases of this descriptor

Returns

Callable[[], List[Argument]]
the registered descriptor argument method

Examples

```
>>> some_plugin = ArgsPlugin()
>>> @some_plugin.register("some_descrpt")
    def descrpt_some_descrpt_args():
        return []
```

get_all_argument(exclude_hybrid: bool = False) → List[dargs.Argument]

Get all arguments.

Parameters

exclude_hybrid
[bool] exclude hybrid descriptor to prevent circular calls

Returns

List[Argument]
all arguments

deepmd.utils.argcheck.**descrpt_args_plugin**

deepmd.utils.argcheck.**descrpt_local_frame_args**()

deepmd.utils.argcheck.**descrpt_se_a_args**()

deepmd.utils.argcheck.**descrpt_se_t_args**()

deepmd.utils.argcheck.**descrpt_se_a_tpe_args**()

deepmd.utils.argcheck.**descrpt_se_r_args**()

deepmd.utils.argcheck.**descrpt_hybrid_args**()

deepmd.utils.argcheck.**descrpt_se_atten_common_args**()

deepmd.utils.argcheck.**descrpt_se_atten_args**()

deepmd.utils.argcheck.**descrpt_se_atten_v2_args**()

deepmd.utils.argcheck.**descrpt_dpa2_args**()

deepmd.utils.argcheck.**descrpt_se_a_ebd_v2_args**()

deepmd.utils.argcheck.**descrpt_se_a_mask_args**()

deepmd.utils.argcheck.**descrpt_variant_type_args**(exclude_hybrid: bool = False) → dargs.Variant

deepmd.utils.argcheck.**fitting_args_plugin**

deepmd.utils.argcheck.**fitting_ener**()

deepmd.utils.argcheck.**fitting_dos**()

deepmd.utils.argcheck.**fitting_polar**()

deepmd.utils.argcheck.**fitting_dipole**()

deepmd.utils.argcheck.**fitting_variant_type_args**()

deepmd.utils.argcheck.**modifier_dipole_charge**()

deepmd.utils.argcheck.**modifier_variant_type_args**()

deepmd.utils.argcheck.**model_compression**()

deepmd.utils.argcheck.**model_compression_type_args**()

deepmd.utils.argcheck.**model_args**(exclude_hybrid=False)

deepmd.utils.argcheck.**standard_model_args**() → dargs.Argument

deepmd.utils.argcheck.**multi_model_args**() → dargs.Argument

deepmd.utils.argcheck.**pairwise_dprc**() → dargs.Argument

deepmd.utils.argcheck.**frozen_model_args**() → dargs.Argument

deepmd.utils.argcheck.**pairtab_model_args**() → dargs.Argument

deepmd.utils.argcheck.**linear_ener_model_args**() → dargs.Argument

deepmd.utils.argcheck.**learning_rate_exp**()

deepmd.utils.argcheck.**learning_rate_variant_type_args**()

deepmd.utils.argcheck.**learning_rate_args**()

deepmd.utils.argcheck.**learning_rate_dict_args**()

deepmd.utils.argcheck.**start_pref**(item, label=None, abbr=None)

deepmd.utils.argcheck.**limit_pref**(item)

deepmd.utils.argcheck.**loss_args_plugin**

deepmd.utils.argcheck.**loss_ener**()

deepmd.utils.argcheck.**loss_ener_spin**()

deepmd.utils.argcheck.**loss_dos**()

deepmd.utils.argcheck.**loss_tensor**()

deepmd.utils.argcheck.**loss_variant_type_args**()

deepmd.utils.argcheck.**loss_args**()

deepmd.utils.argcheck.**loss_dict_args**()

deepmd.utils.argcheck.**training_data_args**()

deepmd.utils.argcheck.**validation_data_args**()

deepmd.utils.argcheck.**mixed_precision_args**()

deepmd.utils.argcheck.**training_args**()

deepmd.utils.argcheck.**make_index**(keys)

deepmd.utils.argcheck.**gen_doc**(*, make_anchor=True, make_link=True, **kwargs)

deepmd.utils.argcheck.**gen_json**(**kwargs)

deepmd.utils.argcheck.**gen_args**(**kwargs) → List[dargs.Argument]

deepmd.utils.argcheck.**normalize_multi_task**(data)

deepmd.utils.argcheck.**normalize_fitting_net_dict**(fitting_net_dict)

deepmd.utils.argcheck.**normalize_data_dict**(data_dict)

deepmd.utils.argcheck.**normalize_loss_dict**(fitting_keys, loss_dict)

deepmd.utils.argcheck.**normalize_learning_rate_dict**(fitting_keys, learning_rate_dict)

deepmd.utils.argcheck.**normalize_learning_rate_dict_with_single_learning_rate**(fitting_keys, learning_rate)

deepmd.utils.argcheck.**normalize_fitting_weight**(fitting_keys, data_keys, fitting_weight=None)

deepmd.utils.argcheck.**normalize**(data)

deepmd.utils.argcheck_nvnmd

Module Contents

Functions

nvnmd_args()	

deepmd.utils.argcheck_nvnmd.**nvnmd_args**()

deepmd.utils.batch_size

Module Contents

Classes

AutoBatchSize	This class allows DeePMD-kit to automatically decide the maximum

Attributes

log

deepmd.utils.batch_size.log

class deepmd.utils.batch_size.**AutoBatchSize**(initial_batch_size: int = 1024, factor: float = 2.0)

Bases: abc.ABC

This class allows DeePMD-kit to automatically decide the maximum batch size that will not cause an OOM error.

Parameters

initial_batch_size
: [int, default: 1024] initial batch size (number of total atoms) when DP_INFER_BATCH_SIZE is not set

factor
: [float, default: 2.] increased factor

Notes

In some CPU environments, the program may be directly killed when OOM. In this case, by default the batch size will not be increased for CPUs. The environment variable DP_INFER_BATCH_SIZE can be set as the batch size.

In other cases, we assume all OOM error will raise `OutOfMemoryError`.

Attributes

current_batch_size
: [int] current batch size (number of total atoms)

maximum_working_batch_size
: [int] maximum working batch size

minimal_not_working_batch_size
: [int] minimal not working batch size

execute(callable: Callable, start_index: int, natoms: int) → Tuple[int, tuple]

Excuate a method with given batch size.

Parameters

callable
: [Callable] The method should accept the batch size and start_index as parameters, and returns executed batch size and data.

start_index
: [int] start index

natoms
: [int] natoms

Returns

int
executed batch size * number of atoms

tuple
result from callable, None if failing to execute

Raises

OutOfMemoryError
OOM when batch size is 1

_adjust_batch_size(factor: float)

execute_all(callable: Callable, total_size: int, natoms: int, *args, **kwargs) → Tuple[numpy.ndarray]

Excuate a method with all given data.

Parameters

callable
[Callable] The method should accept *args and **kwargs as input and return the similiar array.

total_size
[int] Total size

natoms
[int] The number of atoms

*args
Variable length argument list.

**kwargs
If 2D np.ndarray, assume the first axis is batch; otherwise do nothing.

abstract is_gpu_available() → bool

Check if GPU is available.

Returns

bool
True if GPU is available

abstract is_oom_error(e: Exception) → bool

Check if the exception is an OOM error.

Parameters

e
[Exception] Exception

Returns

bool
True if the exception is an OOM error

deepmd.utils.compat

Module providing compatibility between 0.x.x and 1.x.x input versions.

Module Contents

Functions

convert_input_v0_v1(→ Dict[str, Any])	Convert input from v0 format to v1.
_warning_input_v0_v1(fname)	
_model(→ Dict[str, Dict[str, Any]])	Convert data to v1 input for non-smooth model.
_nonsmth_descriptor(→ Dict[str, Any])	Convert data to v1 input for non-smooth descriptor.
_smth_descriptor(→ Dict[str, Any])	Convert data to v1 input for smooth descriptor.
_fitting_net(→ Dict[str, Any])	Convert data to v1 input for fitting net.
_learning_rate(→ Dict[str, Any])	Convert data to v1 input for learning rate section.
_loss(→ Dict[str, Any])	Convert data to v1 input for loss function.
_training(→ Dict[str, Any])	Convert data to v1 input for training.
_jcopy(src, dst, keys)	Copy specified keys from one dict to another.
remove_decay_rate(jdata)	Convert decay_rate to stop_lr.
convert_input_v1_v2(→ Dict[str, Any])	
_warning_input_v1_v2(fname)	
deprecate_numb_test(→ Dict[str, Any])	Deprecate numb_test since v2.1. It has taken no effect since v2.0.
update_deepmd_input(→ Dict[str, Any])	

deepmd.utils.compat.**convert_input_v0_v1**(jdata: Dict[str, Any], warning: bool = True, dump: str | pathlib.Path | None = None) → Dict[str, Any]

Convert input from v0 format to v1.

Parameters

jdata
: [Dict[str, *Any*]] loaded json/yaml file

warning
: [bool, optional] whether to show deprecation warning, by default True

dump
: [Optional[Union[str, Path]], optional] whether to dump converted file, by default None

Returns

Dict[str, *Any*]
: converted output

deepmd.utils.compat.**_warning_input_v0_v1**(fname: str | pathlib.Path | None)

deepmd.utils.compat._model(jdata: Dict[str, Any], smooth: bool) → Dict[str, Dict[str, Any]]

Convert data to v1 input for non-smooth model.

Parameters

jdata
[Dict[str, Any]] parsed input json/yaml data

smooth
[bool] whether to use smooth or non-smooth descriptor version

Returns

Dict[str, Dict[str, Any]]
dictionary with model input parameters and sub-dictionaries for descriptor and fitting net

deepmd.utils.compat._nonsmth_descriptor(jdata: Dict[str, Any]) → Dict[str, Any]

Convert data to v1 input for non-smooth descriptor.

Parameters

jdata
[Dict[str, Any]] parsed input json/yaml data

Returns

Dict[str, Any]
dict with descriptor parameters

deepmd.utils.compat._smth_descriptor(jdata: Dict[str, Any]) → Dict[str, Any]

Convert data to v1 input for smooth descriptor.

Parameters

jdata
[Dict[str, Any]] parsed input json/yaml data

Returns

Dict[str, Any]
dict with descriptor parameters

deepmd.utils.compat._fitting_net(jdata: Dict[str, Any]) → Dict[str, Any]

Convert data to v1 input for fitting net.

Parameters

jdata
[Dict[str, Any]] parsed input json/yaml data

Returns

Dict[str, Any]
dict with fitting net parameters

deepmd.utils.compat._learning_rate(jdata: Dict[str, Any]) → Dict[str, Any]

Convert data to v1 input for learning rate section.

Parameters

jdata
[Dict[str, Any]] parsed input json/yaml data

Returns

Dict[str, Any]
dict with learning rate parameters

deepmd.utils.compat._loss(jdata: Dict[str, Any]) → Dict[str, Any]

Convert data to v1 input for loss function.

Parameters

jdata
[Dict[str, Any]] parsed input json/yaml data

Returns

Dict[str, Any]
dict with loss function parameters

deepmd.utils.compat._training(jdata: Dict[str, Any]) → Dict[str, Any]

Convert data to v1 input for training.

Parameters

jdata
[Dict[str, Any]] parsed input json/yaml data

Returns

Dict[str, Any]
dict with training parameters

deepmd.utils.compat._jcopy(src: Dict[str, Any], dst: Dict[str, Any], keys: Sequence[str])

Copy specified keys from one dict to another.

Parameters

src
[Dict[str, Any]] source dictionary

dst
[Dict[str, Any]] destination dictionary, will be modified in place

keys
[Sequence[str]] list of keys to copy

deepmd.utils.compat.remove_decay_rate(jdata: Dict[str, Any])

Convert decay_rate to stop_lr.

Parameters

jdata
[Dict[str, Any]] input data

deepmd.utils.compat.convert_input_v1_v2(jdata: Dict[str, Any], warning: bool = True, dump: str | pathlib.Path | None = None) → Dict[str, Any]

deepmd.utils.compat._warning_input_v1_v2(fname: str | pathlib.Path | None)

deepmd.utils.compat.deprecate_numb_test(jdata: Dict[str, Any], warning: bool = True, dump: str | pathlib.Path | None = None) → Dict[str, Any]

Deprecate numb_test since v2.1. It has taken no effect since v2.0.

See #1243.

Parameters

jdata
: [Dict[str, *Any*]] loaded json/yaml file

warning
: [bool, optional] whether to show deprecation warning, by default True

dump
: [Optional[Union[str, Path]], optional] whether to dump converted file, by default None

Returns

Dict[str, *Any*]
: converted output

deepmd.utils.compat.**update_deepmd_input**(jdata: Dict[str, Any], warning: bool = True, dump: str | pathlib.Path | None = None) → Dict[str, Any]

deepmd.utils.data

Module Contents

Classes

DeepmdData	Class for a data system.
DataRequirementItem	A class to store the data requirement for data systems.

Attributes

log

deepmd.utils.data.log

class deepmd.utils.data.**DeepmdData**(sys_path: str, set_prefix: str = 'set', shuffle_test: bool = True, type_map: List[str] | None = None, optional_type_map: bool = True, modifier=None, trn_all_set: bool = False, sort_atoms: bool = True)

Class for a data system.

It loads data from hard disk, and mantains the data as a data_dict

Parameters

sys_path
: Path to the data system

set_prefix
: Prefix for the directories of different sets

shuffle_test
: If the test data are shuffled

type_map
: Gives the name of different atom types

optional_type_map
: If the type_map.raw in each system is optional

modifier
: Data modifier that has the method modify_data

trn_all_set
: Use all sets as training dataset. Otherwise, if the number of sets is more than 1, the last set is left for test.

sort_atoms
: [bool] Sort atoms by atom types. Required to enable when the data is directly feeded to descriptors except mixed types.

add(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: List[int] | None = None, repeat: int = 1, default: float = 0.0, dtype: numpy.dtype | None = None, output_natoms_for_type_sel: bool = False)

Add a data item that to be loaded.

Parameters

key
: The key of the item. The corresponding data is stored in sys_path/set.*/key.npy

ndof
: The number of dof

atomic
: The item is an atomic property. If False, the size of the data should be nframes x ndof If True, the size of data should be nframes x natoms x ndof

must
: The data file sys_path/set.*/key.npy must exist. If must is False and the data file does not exist, the data_dict[find_key] is set to 0.0

high_prec
: Load the data and store in float64, otherwise in float32

type_sel
: Select certain type of atoms

repeat
: The data will be repeated repeat times.

default
: [`float`, default=0.] default value of data

dtype
: [`np.dtype`, `optional`] the dtype of data, overwrites high_prec if provided

output_natoms_for_type_sel
: [bool, `optional`] if True and type_sel is True, the atomic dimension will be natoms instead of nsel

reduce(key_out: str, key_in: str)

Generate a new item from the reduction of another atom.

Parameters

key_out
The name of the reduced item

key_in
The name of the data item to be reduced

get_data_dict() → dict

Get the data_dict.

check_batch_size(batch_size)

Check if the system can get a batch of data with batch_size frames.

check_test_size(test_size)

Check if the system can get a test dataset with test_size frames.

get_item_torch(index: int) → dict

Get a single frame data . The frame is picked from the data system by index. The index is coded across all the sets.

Parameters

index
index of the frame

get_batch(batch_size: int) → dict

Get a batch of data with batch_size frames. The frames are randomly picked from the data system.

Parameters

batch_size
size of the batch

get_test(ntests: int = -1) → dict

Get the test data with ntests frames.

Parameters

ntests
Size of the test data set. If ntests is -1, all test data will be get.

get_ntypes() → int

Number of atom types in the system.

get_type_map() → List[str]

Get the type map.

get_atom_type() → List[int]

Get atom types.

get_numb_set() → int

Get number of training sets.

get_numb_batch(batch_size: int, set_idx: int) → int

Get the number of batches in a set.

get_sys_numb_batch(batch_size: int) → int

Get the number of batches in the data system.

get_natoms()

Get number of atoms.

get_natoms_vec(ntypes: int)

Get number of atoms and number of atoms in different types.

Parameters

ntypes
Number of types (may be larger than the actual number of types in the system).

Returns

natoms
natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

avg(key)

Return the average value of an item.

_idx_map_sel(atom_type, type_sel)

_get_natoms_2(ntypes)

_get_subdata(data, idx=None)

_load_batch_set(set_name: deepmd.utils.path.DPPath)

reset_get_batch()

_load_test_set(set_name: deepmd.utils.path.DPPath, shuffle_test)

_shuffle_data(data)

_get_nframes(set_name: deepmd.utils.path.DPPath)

reformat_data_torch(data)

Modify the data format for the requirements of Torch backend.

Parameters

data
original data

_load_set(set_name: deepmd.utils.path.DPPath)

_load_data(set_name, key, nframes, ndof_, atomic=False, must=True, repeat=1, high_prec=False, type_sel=None, default: float = 0.0, dtype: numpy.dtype | None = None, output_natoms_for_type_sel: bool = False)

_load_type(sys_path: deepmd.utils.path.DPPath)

_load_type_mix(set_name: deepmd.utils.path.DPPath)

_make_idx_map(atom_type)

_load_type_map(sys_path: deepmd.utils.path.DPPath)

_check_pbc(sys_path: deepmd.utils.path.DPPath)

_check_mode(set_path: deepmd.utils.path.DPPath)

class deepmd.utils.data.DataRequirementItem(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: List[int] | None = None, repeat: int = 1, default: float = 0.0, dtype: numpy.dtype | None = None, output_natoms_for_type_sel: bool = False)

A class to store the data requirement for data systems.

Parameters

key
The key of the item. The corresponding data is stored in sys_path/set.*/key.npy

ndof
The number of dof

atomic
The item is an atomic property. If False, the size of the data should be nframes x ndof If True, the size of data should be nframes x natoms x ndof

must
The data file sys_path/set.*/key.npy must exist. If must is False and the data file does not exist, the data_dict[find_key] is set to 0.0

high_prec
Load the data and store in float64, otherwise in float32

type_sel
Select certain type of atoms

repeat
The data will be repeated repeat times.

default
[float, default=0.] default value of data

dtype
[np.dtype, optional] the dtype of data, overwrites high_prec if provided

output_natoms_for_type_sel
[bool, optional] if True and type_sel is True, the atomic dimension will be natoms instead of nsel

to_dict() → dict

__getitem__(key: str)

deepmd.utils.data_system

Module Contents

Classes

DeepmdDataSystem	Class for manipulating many data systems.

Functions

_format_name_length(name, width)	
print_summary(name, nsystems, system_dirs, natoms, ...)	Print summary of systems.
process_sys_probs(sys_probs, nbatch)	
prob_sys_size_ext(keywords, nsystems, nbatch)	
process_systems(→ List[str])	Process the user-input systems.
get_data(→ DeepmdDataSystem)	Get the data system.

Attributes

log

deepmd.utils.data_system.log

class deepmd.utils.data_system.**DeepmdDataSystem**(systems: List[str], batch_size: int, test_size: int, rcut: float | None = None, set_prefix: str = 'set', shuffle_test: bool = True, type_map: List[str] | None = None, optional_type_map: bool = True, modifier=None, trn_all_set=False, sys_probs=None, auto_prob_style='prob_sys_size', sort_atoms: bool = True)

Class for manipulating many data systems.

It is implemented with the help of DeepmdData

property default_mesh: List[numpy.ndarray]

Mesh for each system.

_load_test(ntests=-1)

compute_energy_shift(rcond=None, key='energy')

add_dict(adict: dict) → None

Add items to the data system by a dict. adict should have items like .. code-block:: python.

adict[key] = {
"ndof": ndof, "atomic": atomic, "must": must, "high_prec": high_prec, "type_sel": type_sel, "repeat": repeat,

}

For the explaination of the keys see add

add(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: List[int] | None = None, repeat: int = 1, default: float = 0.0, dtype: numpy.dtype | None = None, output_natoms_for_type_sel: bool = False)

Add a data item that to be loaded.

Parameters

key
The key of the item. The corresponding data is stored in sys_path/set.*/key.npy

ndof
The number of dof

atomic
The item is an atomic property. If False, the size of the data should be nframes x ndof If True, the size of data should be nframes x natoms x ndof

must
The data file sys_path/set.*/key.npy must exist. If must is False and the data file does not exist, the data_dict[find_key] is set to 0.0

high_prec
Load the data and store in float64, otherwise in float32

type_sel
Select certain type of atoms

repeat
The data will be repeated repeat times.

default, default=0.
Default value of data

dtype
The dtype of data, overwrites high_prec if provided

output_natoms_for_type_sel
[bool] If True and type_sel is True, the atomic dimension will be natoms instead of nsel

reduce(key_out, key_in)

Generate a new item from the reduction of another atom.

Parameters

key_out
The name of the reduced item

key_in
The name of the data item to be reduced

get_data_dict(ii: int = 0) → dict

set_sys_probs(sys_probs=None, auto_prob_style: str = 'prob_sys_size')

get_batch(sys_idx: int | None = None) → dict

Get a batch of data from the data systems.

Parameters

sys_idx
[int] The index of system from which the batch is get. If sys_idx is not None, sys_probs and auto_prob_style are ignored If sys_idx is None, automatically determine the system according to sys_probs or auto_prob_style, see the following. This option does not work for mixed systems.

Returns

dict
The batch data

get_batch_standard(sys_idx: int | None = None) → dict

Get a batch of data from the data systems in the standard way.

Parameters

sys_idx
[int] The index of system from which the batch is get. If sys_idx is not None, sys_probs and auto_prob_style are ignored If sys_idx is None, automatically determine the system according to sys_probs or auto_prob_style, see the following.

Returns

dict
The batch data

get_batch_mixed() → dict

Get a batch of data from the data systems in the mixed way.

Returns

dict
The batch data

_merge_batch_data(batch_data: List[dict]) → dict

Merge batch data from different systems.

Parameters

batch_data
[list of dict] A list of batch data from different systems.

Returns

dict
The merged batch data.

get_test(sys_idx: int | None = None, n_test: int = -1)

Get test data from the the data systems.

Parameters

sys_idx
The test dat of system with index sys_idx will be returned. If is None, the currently selected system will be returned.

n_test
Number of test data. If set to -1 all test data will be get.

get_sys_ntest(sys_idx=None)

Get number of tests for the currently selected system, or one defined by sys_idx.

get_type_map() → List[str]

Get the type map.

get_nbatches() → int

Get the total number of batches.

get_ntypes() → int

Get the number of types.

get_nsystems() → int

Get the number of data systems.

get_sys(idx: int) → deepmd.utils.data.DeepmdData

Get a certain data system.

get_batch_size() → int

Get the batch size.

print_summary(name: str)

_make_auto_bs(rule)

_make_auto_ts(percent)

_check_type_map_consistency(type_map_list)

deepmd.utils.data_system.**_format_name_length**(name, width)

deepmd.utils.data_system.**print_summary**(name: str, nsystems: int, system_dirs: List[str], natoms: List[int], batch_size: List[int], nbatches: List[int], sys_probs: List[float], pbc: List[bool])

Print summary of systems.

Parameters

name
[str] The name of the system

nsystems
[int] The number of systems

system_dirs
[list of str] The directories of the systems

natoms
[list of int] The number of atoms

batch_size
[list of int] The batch size

nbatches
[list of int] The number of batches

sys_probs
[list of float] The probabilities

pbc
[list of bool] The periodic boundary conditions

deepmd.utils.data_system.**process_sys_probs**(sys_probs, nbatch)

deepmd.utils.data_system.**prob_sys_size_ext**(keywords, nsystems, nbatch)

deepmd.utils.data_system.**process_systems**(systems: str | List[str]) → List[str]

Process the user-input systems.

If it is a single directory, search for all the systems in the directory. Check if the systems are valid.

Parameters

systems
: [str or list of str] The user-input systems

Returns

list of str
: The valid systems

deepmd.utils.data_system.get_data(jdata: Dict[str, Any], rcut, type_map, modifier, multi_task_mode=False) → DeepmdDataSystem

Get the data system.

Parameters

jdata
: The json data

rcut
: The cut-off radius, not used

type_map
: The type map

modifier
: The data modifier

multi_task_mode
: If in multi task mode

Returns

DeepmdDataSystem
: The data system

deepmd.utils.econf_embd

Module Contents

Functions

make_empty_list_vec()	
flatten_list_vec(lv)	
make_element_embedding_list_vec(→ numpy.ndarray)	Compute the embedding of one element.
make_econf_embedding(types[, flatten])	
print_econf_embedding(res)	

Attributes

electronic_configuration_embedding
ln_to_lett
lett_to_ln
conf_keys
maxn
maxl
maxm
type_map

deepmd.utils.econf_embd.**electronic_configuration_embedding**

deepmd.utils.econf_embd.**ln_to_lett**

deepmd.utils.econf_embd.**lett_to_ln**

deepmd.utils.econf_embd.**conf_keys = [(1, 's'), (2, 's'), (2, 'p'), (3, 's'), (3, 'p'), (3, 'd'), (4, 's'), (4, 'p'), (4, 'd'), (4,...**

deepmd.utils.econf_embd.**maxn = 7**

deepmd.utils.econf_embd.**maxl**

deepmd.utils.econf_embd.**maxm**

deepmd.utils.econf_embd.**type_map**

deepmd.utils.econf_embd.**make_empty_list_vec**()

deepmd.utils.econf_embd.**flatten_list_vec**(lv)

deepmd.utils.econf_embd.**make_element_embedding_list_vec**(ename: str) → numpy.ndarray

Compute the embedding of one element.

deepmd.utils.econf_embd.**make_econf_embedding**(types, flatten=True)

deepmd.utils.econf_embd.**print_econf_embedding**(res)

deepmd.utils.env_mat_stat

Module Contents

Classes

StatItem	A class to store the statistics of the environment matrix.
EnvMatStat	A base class to store and calculate the statistics of the environment matrix.

Attributes

log

deepmd.utils.env_mat_stat.log

class deepmd.utils.env_mat_stat.**StatItem**(number: int = 0, sum: float = 0, squared_sum: float = 0)

A class to store the statistics of the environment matrix.

Parameters

number
[int] The total size of given array.

sum
[float] The sum value of the matrix.

squared_sum
[float] The sum squared value of the matrix.

__add__(other: StatItem) → StatItem

compute_avg(default: float = 0) → float

Compute the average of the environment matrix.

Parameters

default
[float, optional] The default value of the average, by default 0.

Returns

float
The average of the environment matrix.

compute_std(default: float = 0.1, protection: float = 0.01) → float

Compute the standard deviation of the environment matrix.

Parameters

default
[float, optional] The default value of the standard deviation, by default 1e-1.

protection
[float, optional] The protection value for the standard deviation, by default 1e-2.

Returns

float
The standard deviation of the environment matrix.

class deepmd.utils.env_mat_stat.EnvMatStat

Bases: abc.ABC

A base class to store and calculate the statistics of the environment matrix.

compute_stats(data: List[Dict[str, numpy.ndarray]]) → None

Compute the statistics of the environment matrix.

Parameters

data
[List[Dict[str, np.ndarray]]] The environment matrix.

abstract iter(data: List[Dict[str, numpy.ndarray]]) → Iterator[Dict[str, StatItem]]

Get the iterator of the environment matrix.

Parameters

data
[List[Dict[str, np.ndarray]]] The environment matrix.

Yields

Dict[str, *StatItem*]
The statistics of the environment matrix.

save_stats(path: deepmd.utils.path.DPPath) → None

Save the statistics of the environment matrix.

Parameters

path
[DPPath] The path to save the statistics of the environment matrix.

load_stats(path: deepmd.utils.path.DPPath) → None

Load the statistics of the environment matrix.

Parameters

path
[DPPath] The path to load the statistics of the environment matrix.

load_or_compute_stats(data: List[Dict[str, numpy.ndarray]], path: deepmd.utils.path.DPPath | None = None) → None

Load the statistics of the environment matrix if it exists, otherwise compute and save it.

Parameters

path
[DPPath] The path to load the statistics of the environment matrix.

data
[List[Dict[str, np.ndarray]]] The environment matrix.

get_avg(default: float = 0) → Dict[str, float]

Get the average of the environment matrix.

Parameters

default
[float, optional] The default value of the average, by default 0.

Returns

Dict[str, float]
The average of the environment matrix.

get_std(default: float = 0.1, protection: float = 0.01) → Dict[str, float]

Get the standard deviation of the environment matrix.

Parameters

default
[float, optional] The default value of the standard deviation, by default 1e-1.

protection
[float, optional] The protection value for the standard deviation, by default 1e-2.

Returns

Dict[str, float]
The standard deviation of the environment matrix.

deepmd.utils.errors

Module Contents

exception deepmd.utils.errors.OutOfMemoryError

Bases: Exception

This error is caused by out-of-memory (OOM).

deepmd.utils.finetune

Module Contents

Functions

change_energy_bias_lower(data, dp, origin_type_map, ...)	Change the energy bias according to the input data and the pretrained model.

Attributes

log

deepmd.utils.finetune.log

deepmd.utils.finetune.**change_energy_bias_lower**(data: deepmd.utils.data_system.DeepmdDataSystem, dp: deepmd.infer.deep_eval.DeepEval, origin_type_map: List[str], full_type_map: List[str], bias_atom_e: numpy.ndarray, bias_adjust_mode='change-by-statistic', ntest=10)

Change the energy bias according to the input data and the pretrained model.

Parameters

data
[DeepmdDataSystem] The training data.

dp
[str] The DeepEval object.

origin_type_map
[list] The original type_map in dataset, they are targets to change the energy bias.

full_type_map
[str] The full type_map in pretrained model

bias_atom_e
[np.ndarray] The old energy bias in the pretrained model.

bias_adjust_mode
[str] The mode for changing energy bias : ['change-by-statistic', 'set-by-statistic']
'change-by-statistic' : perform predictions on energies of target dataset,

and do least sqaure on the errors to obtain the target shift as bias.

'set-by-statistic' : directly use the statistic energy bias in the target dataset.

ntest
[int] The number of test samples in a system to change the energy bias.

deepmd.utils.hostlist

Module Contents

Functions

get_host_names(→ Tuple[str, List[str]])	Get host names of all nodes in the cluster.

deepmd.utils.hostlist.**get_host_names**() → Tuple[str, List[str]]

Get host names of all nodes in the cluster.

If mpi4py is not installed or MPI is not used, then the host name of the current node is returned as those of all nodes.

Returns

str
Host name of the current node

List[**str**]
List of host names of all nodes in the cluster

deepmd.utils.model_stat

Module Contents

Functions

_make_all_stat_ref(data, nbatches)	
make_stat_input(data, nbatches[, merge_sys])	Pack data for statistics.
merge_sys_stat(all_stat)	

deepmd.utils.model_stat.**_make_all_stat_ref**(data, nbatches)

deepmd.utils.model_stat.**make_stat_input**(data, nbatches, merge_sys=True)

Pack data for statistics.

Parameters

data
The data

nbatches
[int] The number of batches

merge_sys
[bool (True)] Merge system data

Returns

all_stat:
A dictionary of list of list storing data for stat. if merge_sys == False data can be accessed by

all_stat[key][sys_idx][batch_idx][frame_idx]

else merge_sys == True can be accessed by
all_stat[key][batch_idx][frame_idx]

deepmd.utils.model_stat.**merge_sys_stat**(all_stat)

deepmd.utils.neighbor_stat

Module Contents

Classes

NeighborStat	Abstract base class for getting training data information.

Attributes

log

deepmd.utils.neighbor_stat.log

class deepmd.utils.neighbor_stat.**NeighborStat**(ntypes: int, rcut: float, mixed_type: bool = False)

Bases: abc.ABC

Abstract base class for getting training data information.

It loads data from DeepmdData object, and measures the data info, including neareest nbor distance between atoms, max nbor size of atoms and the output data range of the environment matrix.

Parameters

ntypes
[int] The num of atom types

rcut
[float] The cut-off radius

mixed_type
[bool, optional, default=False] Treat all types as a single type.

get_stat(data: deepmd.utils.data_system.DeepmdDataSystem) → Tuple[float, numpy.ndarray]

Get the data statistics of the training data, including nearest nbor distance between atoms, max nbor size of atoms.

Parameters

data
Class for manipulating many data systems. It is implemented with the help of DeepmdData.

Returns

min_nbor_dist
The nearest distance between neighbor atoms

max_nbor_size
An array with ntypes integers, denotes the actual achieved max sel

abstract iterator(data: deepmd.utils.data_system.DeepmdDataSystem) → Iterator[Tuple[numpy.ndarray, float, str]]

Abstract method for producing data.

Yields

mn
[np.ndarray] The maximal number of neighbors

dt
[float] The squared minimal distance between two atoms

jj
[str] The directory of the data system

deepmd.utils.out_stat

Output statistics.

Module Contents

Functions

compute_stats_from_redu(→ Tuple[numpy.ndarray, ...)		Compute the output statistics.
compute_stats_from_atomic(→ Tuple[numpy.ndarray, ...)		Compute the output statistics.

deepmd.utils.out_stat.compute_stats_from_redu(output_redu: numpy.ndarray, natoms: numpy.ndarray, assigned_bias: numpy.ndarray | None = None, rcond: float | None = None) → Tuple[numpy.ndarray, numpy.ndarray]

Compute the output statistics.

Given the reduced output value and the number of atoms for each atom, compute the least-squares solution as the atomic output bais and std.

Parameters

output_redu
The reduced output value, shape is [nframes, *(odim0, odim1, ...)].

natoms
The number of atoms for each atom, shape is [nframes, ntypes].

assigned_bias
The assigned output bias, shape is [ntypes, *(odim0, odim1, ...)]. Set to a tensor of shape (odim0, odim1, ...) filled with nan if the bias of the type is not assigned.

rcond
Cut-off ratio for small singular values of a.

Returns

np.ndarray
The computed output bias, shape is [ntypes, *(odim0, odim1, ...)].

np.ndarray
The computed output std, shape is [*(odim0, odim1, …)].

deepmd.utils.out_stat.compute_stats_from_atomic(output: numpy.ndarray, atype: numpy.ndarray) → Tuple[numpy.ndarray, numpy.ndarray]

Compute the output statistics.

Given the output value and the type of atoms, compute the atomic output bais and std.

Parameters

output
The output value, shape is [nframes, nloc, ndim].

atype
The type of atoms, shape is [nframes, nloc].

Returns

np.ndarray
The computed output bias, shape is [ntypes, ndim].

np.ndarray
The computed output std, shape is [ntypes, ndim].

deepmd.utils.pair_tab

Module Contents

Classes

PairTab	Pairwise tabulated potential.

Attributes

log

deepmd.utils.pair_tab.log

class deepmd.utils.pair_tab.PairTab(filename: str, rcut: float | None = None)

Pairwise tabulated potential.

Parameters

filename
File name for the short-range tabulated potential. The table is a text data file with (N_t + 1) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

reinit(filename: str, rcut: float | None = None) → None

Initialize the tabulated interaction.

Parameters

filename
File name for the short-range tabulated potential. The table is a text data file with (N_t + 1) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

serialize() → dict

classmethod deserialize(data) → PairTab

_check_table_upper_boundary() → None

Update User Provided Table Based on rcut.

This function checks the upper boundary provided in the table against rcut. If the table upper boundary values decay to zero before rcut, padding zeros will be added to the table to cover rcut; if the table upper boundary values do not decay to zero before ruct, extrapolation will be performed till rcut.

Examples

table = [[0.005 1. 2. 3.]
[0.01 0.8 1.6 2.4] [0.015 0. 1. 1.5]]

rcut = 0.022

new_table = [[0.005 1. 2. 3.]
[0.01 0.8 1.6 2.4] [0.015 0. 1. 1.5] [0.02 0. 0. 0.]

---

table = [[0.005 1. 2. 3.]
[0.01 0.8 1.6 2.4] [0.015 0.5 1. 1.5] [0.02 0.25 0.4 0.75] [0.025 0. 0.1 0.] [0.03 0. 0. 0.]]

rcut = 0.031

new_table = [[0.005 1. 2. 3.]
[0.01 0.8 1.6 2.4] [0.015 0.5 1. 1.5] [0.02 0.25 0.4 0.75] [0.025 0. 0.1 0.] [0.03 0. 0. 0.] [0.035 0. 0. 0.]]

get() → Tuple[numpy.array, numpy.array]

Get the serialized table.

_extrapolate_table(pad_extrapolation: numpy.array) → numpy.array

Soomth extrapolation between table upper boundary and rcut.

This method should only be used when the table upper boundary rmax is smaller than rcut, and the table upper boundary values are not zeros. To simplify the problem, we use a single cubic spline between rmax and rcut for each pair of atom types. One can substitute this extrapolation to higher order polynomials if needed.

There are two scenarios:

1.ruct - rmax >= hh:
Set values at the grid point right before rcut to 0, and perform exterapolation between the grid point and rmax, this allows smooth decay to 0 at rcut.

2.rcut - rmax < hh:
Set values at rmax + hh to 0, and perform extrapolation between rmax and rmax + hh.

Parameters

pad_extrapolation
[np.array] The emepty grid that holds the extrapolation values.

Returns

np.array
The cubic spline extrapolation.

_make_data()

deepmd.utils.path

Module Contents

Classes

DPPath	The path class to data system (DeepmdData).
DPOSPath	The OS path class to data system (DeepmdData) for real directories.
DPH5Path	The path class to data system (DeepmdData) for HDF5 files.

class deepmd.utils.path.DPPath

Bases: abc.ABC

The path class to data system (DeepmdData).

Parameters

path
[str] path

mode
[str, optional] mode, by default "r"

abstract property name: str

Name of the path.

abstract load_numpy() → numpy.ndarray

Load NumPy array.

Returns

np.ndarray
loaded NumPy array

abstract load_txt(**kwargs) → numpy.ndarray

Load NumPy array from text.

Returns

np.ndarray
loaded NumPy array

abstract save_numpy(arr: numpy.ndarray) → None

Save NumPy array.

Parameters

arr
[np.ndarray] NumPy array

abstract glob(pattern: str) → List[DPPath]

Search path using the glob pattern.

Parameters

pattern
[str] glob pattern

Returns

List[*DPPath*]
list of paths

abstract rglob(pattern: str) → List[DPPath]

This is like calling *DPPath.glob()* with **/ added in front of the given relative pattern.

Parameters

pattern
[str] glob pattern

Returns

List[*DPPath*]
list of paths

abstract is_file() → bool

Check if self is file.

abstract is_dir() → bool

Check if self is directory.

abstract __truediv__(key: str) → DPPath

Used for / operator.

abstract __lt__(other: DPPath) → bool

Whether this DPPath is less than other for sorting.

abstract __str__() → str

Represent string.

__repr__() → str

Return repr(self).

__eq__(other) → bool

Return self==value.

__hash__()

Return hash(self).

abstract mkdir(parents: bool = False, exist_ok: bool = False) → None

Make directory.

Parameters

parents
[bool, optional] If true, any missing parents of this directory are created as well.

exist_ok
[bool, optional] If true, no error will be raised if the target directory already exists.

class deepmd.utils.path.DPOSPath(path: str, mode: str = 'r')

Bases: *DPPath*

The OS path class to data system (DeepmdData) for real directories.

Parameters

path
[str] path

mode
[str, optional] mode, by default "r"

property name: str

Name of the path.

load_numpy() → numpy.ndarray

Load NumPy array.

Returns

np.ndarray
loaded NumPy array

load_txt(**kwargs) → numpy.ndarray

Load NumPy array from text.

Returns

np.ndarray
loaded NumPy array

save_numpy(arr: numpy.ndarray) → None

Save NumPy array.

Parameters

arr
[np.ndarray] NumPy array

glob(pattern: str) → List[DPPath]

Search path using the glob pattern.

Parameters

pattern
[str] glob pattern

Returns

List[*DPPath*]
list of paths

rglob(pattern: str) → List[DPPath]
This is like calling *DPPath.glob()* with **/ added in front of the given relative pattern.

Parameters

pattern
[str] glob pattern

Returns

List[*DPPath*]
list of paths

is_file() → bool
Check if self is file.

is_dir() → bool
Check if self is directory.

__truediv__(key: str) → DPPath
Used for / operator.

__lt__(other: DPOSPath) → bool
Whether this DPPath is less than other for sorting.

__str__() → str
Represent string.

mkdir(parents: bool = False, exist_ok: bool = False) → None
Make directory.

Parameters

parents
[bool, optional] If true, any missing parents of this directory are created as well.

exist_ok
[bool, optional] If true, no error will be raised if the target directory already exists.

class deepmd.utils.path.DPH5Path(path: str, mode: str = 'r')
Bases: *DPPath*

The path class to data system (DeepmdData) for HDF5 files.

Parameters

path
[str] path

mode
[str, optional] mode, by default "r"

Notes

OS - HDF5 relationship:
directory - Group file - Dataset

property _keys: List[str]

Walk all groups and dataset.

property _new_keys

New keys that haven't been cached.

property name: str

Name of the path.

__file_new_keys: ClassVar[Dict[h5py.File, List[str]]]

classmethod _load_h5py(path: str, mode: str = 'r') → h5py.File

Load hdf5 file.

Parameters

path
[str] path to hdf5 file

mode
[str, optional] mode, by default 'r'

load_numpy() → numpy.ndarray

Load NumPy array.

Returns

np.ndarray
loaded NumPy array

load_txt(dtype: numpy.dtype | None = None, **kwargs) → numpy.ndarray

Load NumPy array from text.

Returns

np.ndarray
loaded NumPy array

save_numpy(arr: numpy.ndarray) → None

Save NumPy array.

Parameters

arr
[np.ndarray] NumPy array

glob(pattern: str) → List[DPPath]

Search path using the glob pattern.

Parameters

pattern
[str] glob pattern

Returns

List[*DPPath*]
list of paths

rglob(pattern: str) → List[DPPath]
This is like calling *DPPath.glob()* with **/ added in front of the given relative pattern.

Parameters

pattern
[str] glob pattern

Returns

List[*DPPath*]
list of paths

classmethod _file_keys(file: h5py.File) → List[str]
Walk all groups and dataset.

is_file() → bool
Check if self is file.

is_dir() → bool
Check if self is directory.

__truediv__(key: str) → DPPath
Used for / operator.

_connect_path(path: str) → str
Connect self with path.

__lt__(other: DPH5Path) → bool
Whether this DPPath is less than other for sorting.

__str__() → str
Returns path of self.

mkdir(parents: bool = False, exist_ok: bool = False) → None
Make directory.

Parameters

parents
[bool, **optional**] If true, any missing parents of this directory are created as well.

exist_ok
[bool, **optional**] If true, no error will be raised if the target directory already exists.

deepmd.utils.plugin

Base of plugin systems.

Module Contents

Classes

Plugin	A class to register and restore plugins.
VariantMeta	
VariantABCMeta	Metaclass for defining Abstract Base Classes (ABCs).
PluginVariant	A class to remove type from input arguments.

Functions

make_plugin_registry(→ Type[object])	Make a plugin registry.

class deepmd.utils.plugin.Plugin

A class to register and restore plugins.

Examples

```
>>> plugin = Plugin()
>>> @plugin.register("xx")
    def xxx():
        pass
>>> print(plugin.plugins["xx"])
```

Attributes

plugins
[Dict[str, object]] plugins

__add__(other) → Plugin

register(key: str) → Callable[[object], object]

Register a plugin.

Parameters

key
[str] key of the plugin

Returns

Callable[[object], object]
decorator

get_plugin(key) → object

Visit a plugin by key.

Parameters

key
[str] key of the plugin

Returns

object
the plugin

class deepmd.utils.plugin.VariantMeta

__call__(*args, **kwargs)
Remove type and keys that starts with underline.

class deepmd.utils.plugin.VariantABCMeta
Bases: *VariantMeta*, abc.ABCMeta

Metaclass for defining Abstract Base Classes (ABCs).

Use this metaclass to create an ABC. An ABC can be subclassed directly, and then acts as a mix-in class. You can also register unrelated concrete classes (even built-in classes) and unrelated ABCs as 'virtual subclasses' – these and their descendants will be considered subclasses of the registering ABC by the built-in issubclass() function, but the registering ABC won't show up in their MRO (Method Resolution Order) nor will method implementations defined by the registering ABC be callable (not even via super()).

class deepmd.utils.plugin.PluginVariant
A class to remove type from input arguments.

deepmd.utils.plugin.make_plugin_registry(name: str | None = None) → Type[object]
Make a plugin registry.

Parameters

name
[Optional[str]] the name of the registry for the error message, e.g. descriptor, backend, etc.

Examples

```
>>> class BaseClass(make_plugin_registry()):
        pass
```

deepmd.utils.random

Module Contents

Functions

choice(a[, size, replace, p])	Generates a random sample from a given 1-D array.
random([size])	Return random floats in the half-open interval [0.0, 1.0).
seed([val])	Seed the generator.
shuffle(x)	Modify a sequence in-place by shuffling its contents.

deepmd.utils.random.choice(a: numpy.ndarray | int, size: int | Tuple[int, Ellipsis] | None = None, replace: bool = True, p: numpy.ndarray | None = None)

Generates a random sample from a given 1-D array.

Parameters

a
[1-D array_like or int] If an ndarray, a random sample is generated from its elements. If an int, the random sample is generated as if it were np.arange(a)

size
[int or tuple of ints, optional] Output shape. If the given shape is, e.g., (m, n, k), then m * n * k samples are drawn. Default is None, in which case a single value is returned.

replace
[bool, optional] Whether the sample is with or without replacement. Default is True, meaning that a value of a can be selected multiple times.

p
[1-D array_like, optional] The probabilities associated with each entry in a. If not given, the sample assumes a uniform distribution over all entries in a.

Returns

np.ndarray
arrays with results and their shapes

deepmd.utils.random.random(size=None)

Return random floats in the half-open interval [0.0, 1.0).

Parameters

size
Output shape.

Returns

np.ndarray
Arrays with results and their shapes.

deepmd.utils.random.seed(val: int | None = None)

Seed the generator.

Parameters

val
[int] Seed.

deepmd.utils.random.shuffle(x: numpy.ndarray)

Modify a sequence in-place by shuffling its contents.

Parameters

x
[np.ndarray] The array or list to be shuffled.

deepmd.utils.spin

Module Contents

Classes

Spin	Class for spin, mainly processes the spin type-related information.

class deepmd.utils.spin.Spin(use_spin: List[bool], virtual_scale: List[float] | float)

Class for spin, mainly processes the spin type-related information. Atom types can be split into three kinds: 1. Real types: real atom species, "Fe", "H", "O", etc. 2. Spin types: atom species with spin, as virtual atoms in input, "Fe_spin", etc. 3. Placeholder types: atom species without spin, as placeholders in input without contribution, also name "H_spin", "O_spin", etc. For any types in 2. or 3., the type index is ntypes plus index of its corresponding real type.

Parameters

use_spin: List[bool]
A list of boolean values indicating whether to use atomic spin for each atom type. True for spin and False for not. List of bool values with shape of [ntypes].

virtual_scale: List[float], float
The scaling factor to determine the virtual distance between a virtual atom representing spin and its corresponding real atom for each atom type with spin. This factor is defined as the virtual distance divided by the magnitude of atomic spin for each atom type with spin. The virtual coordinate is defined as the real coordinate plus spin * virtual_scale. List of float values with shape of [ntypes] or [ntypes_spin] or one single float value for all types, only used when use_spin is True for each atom type.

get_ntypes_real() → int

Returns the number of real atom types.

get_ntypes_spin() → int

Returns the number of atom types which contain spin.

get_ntypes_real_and_spin() → int

Returns the number of real atom types and types which contain spin.

get_ntypes_input() → int

Returns the number of double real atom types for input placeholder.

get_use_spin() → List[bool]

Returns the list of whether to use spin for each atom type.

get_virtual_scale() → numpy.ndarray

Returns the list of magnitude of atomic spin for each atom type.

init_pair_exclude_types_placeholder() → None

Initialize the pair-wise exclusion types for descriptor. The placeholder types for those without spin are excluded.

init_atom_exclude_types_placeholder_spin() → None

Initialize the atom-wise exclusion types for fitting. Both the placeholder types and spin types are excluded.

init_atom_exclude_types_placeholder() → None

Initialize the atom-wise exclusion types for fitting. The placeholder types for those without spin are excluded.

get_pair_exclude_types(exclude_types=None) → List[Tuple[int, int]]

Return the pair-wise exclusion types for descriptor. The placeholder types for those without spin are excluded.

get_atom_exclude_types(exclude_types=None) → List[int]

Return the atom-wise exclusion types for fitting before out_def. Both the placeholder types and spin types are excluded.

get_atom_exclude_types_placeholder(exclude_types=None) → List[int]

Return the atom-wise exclusion types for fitting after out_def. The placeholder types for those without spin are excluded.

get_spin_mask()

Return the spin mask of shape [ntypes], with spin types being 1, and non-spin types being 0.

get_virtual_scale_mask()

Return the virtual scale mask of shape [ntypes], with spin types being its virtual scale, and non-spin types being 0.

serialize() → dict

classmethod deserialize(data: dict) → Spin

deepmd.utils.summary

Module Contents

Classes

SummaryPrinter	Base summary printer.

Attributes

log

deepmd.utils.summary.log

class deepmd.utils.summary.**SummaryPrinter**

Bases: abc.ABC

Base summary printer.

Backends should inherit from this class and implement the abstract methods.

WELCOME = (' _____ _____ __ __ _____ _ _ _ ', '| __ \\ | ...

CITATION = ('Please read and cite:', 'Wang, Zhang, Han and E, Comput.Phys.Comm. 228, 178-184 (2018)', 'Zeng...

BUILD: ClassVar

__call__()
Print build and current running cluster configuration summary.

abstract is_built_with_cuda() → bool
Check if the backend is built with CUDA.

abstract is_built_with_rocm() → bool
Check if the backend is built with ROCm.

abstract get_compute_device() → str
Get Compute device.

abstract get_ngpus() → int
Get the number of GPUs.

get_backend_info() → dict
Get backend information.

deepmd.utils.update_sel

Module Contents

Classes

BaseUpdateSel	Update the sel field in the descriptor.

Attributes

log

deepmd.utils.update_sel.log

class deepmd.utils.update_sel.BaseUpdateSel
Update the sel field in the descriptor.

abstract property neighbor_stat: Type[*deepmd.utils.neighbor_stat.NeighborStat*]

update_one_sel(jdata, descriptor, mixed_type: bool = False, rcut_key='rcut', sel_key='sel')

parse_auto_sel(sel)

parse_auto_sel_ratio(sel)

wrap_up_4(xx)

get_sel(jdata, rcut, mixed_type: bool = False)

get_rcut(jdata)

get_type_map(jdata)

get_nbor_stat(jdata, rcut, mixed_type: bool = False)

abstract hook(min_nbor_dist, max_nbor_size)

get_min_nbor_dist(jdata, rcut)

deepmd.utils.version

Module Contents

Functions

check_version_compatibility(current_version, ...[, ...])	Check if the current version is compatible with the supported versions.

deepmd.utils.version.**check_version_compatibility**(current_version: int, maximum_supported_version: int, minimal_supported_version: int = 1)

Check if the current version is compatible with the supported versions.

Parameters

current_version
: [int] The current version.

maximum_supported_version
: [int] The maximum supported version.

minimal_supported_version
: [int, optional] The minimal supported version. Default is 1.

Raises

ValueError
: If the current version is not compatible with the supported versions.

deepmd.utils.weight_avg

Module Contents

Functions

weighted_average(→ Dict)	Compute wighted average of prediction errors (MAE or RMSE) for model.

deepmd.utils.weight_avg.weighted_average(errors: List[Dict[str, Tuple[float, float]]]) → Dict

Compute wighted average of prediction errors (MAE or RMSE) for model.

Parameters

errors
[List[Dict[str, Tuple[float, float]]]] List: the error of systems Dict: the error of quantities, name given by the key str: the name of the quantity, must starts with 'mae' or 'rmse' Tuple: (error, weight)

Returns

Dict
weighted averages

20.2 Submodules

20.2.1 deepmd.__main__

Package dp entry point.

20.2.2 deepmd.calculator

ASE calculator interface module.

Module Contents

Classes

DP	Implementation of ASE deepmd calculator.

class deepmd.calculator.DP(model: str | pathlib.Path, label: str = 'DP', type_dict: Dict[str, int] | None = None, neighbor_list=None, **kwargs)

Bases: ase.calculators.calculator.Calculator

Implementation of ASE deepmd calculator.

Implemented propertie are energy, forces and stress

Parameters

model
[Union[str, Path]] path to the model

label
[str, optional] calculator label, by default "DP"

type_dict
[Dict[str, int], optional] mapping of element types and their numbers, best left None and the calculator will infer this information from model, by default None

neighbor_list
[ase.neighborlist.NeighborList, optional] The neighbor list object. If None, then build the native neighbor list.

Examples

Compute potential energy

```
>>> from ase import Atoms
>>> from deepmd.tf.calculator import DP
>>> water = Atoms('H2O',
>>>               positions=[(0.7601, 1.9270, 1),
>>>                          (1.9575, 1, 1),
>>>                          (1., 1., 1.)],
>>>               cell=[100, 100, 100],
>>>               calculator=DP(model="frozen_model.pb"))
>>> print(water.get_potential_energy())
>>> print(water.get_forces())
```

Run BFGS structure optimization

```
>>> from ase.optimize import BFGS
>>> dyn = BFGS(water)
>>> dyn.run(fmax=1e-6)
>>> print(water.get_positions())
```

name = 'DP'

implemented_properties: ClassVar[List[str]] = ['energy', 'free_energy', 'forces', 'virial', 'stress']

calculate(atoms: ase.Atoms | None = None, properties: List[str] = ['energy', 'forces', 'virial'], system_changes: List[str] = all_changes)

Run calculation with deepmd model.

Parameters

atoms
[Optional[Atoms], optional] atoms object to run the calculation on, by default None

properties
[List[str], optional] unused, only for function signature compatibility, by default ["energy", "forces", "stress"]

system_changes
[List[str], optional] unused, only for function signature compatibility, by default all_changes

20.2.3 deepmd.common

Module Contents

Functions

add_data_requirement(key, ndof[, atomic, must, ...])	Specify data requirements for training.
select_idx_map(→ numpy.ndarray)	Build map of indices for element supplied element types from all atoms list.
make_default_mesh(→ numpy.ndarray)	Make mesh.
j_must_have(→ _DICT_VAL)	Assert that supplied dictionary conaines specified key.
j_loader(→ Dict[str, Any])	Load yaml or json settings file.
expand_sys_str(→ List[str])	Recursively iterate over directories taking those that contain type.raw file.
get_np_precision(→ numpy.dtype)	Get numpy precision constant from string.

Attributes

VALID_PRECISION
VALID_ACTIVATION
data_requirement

deepmd.common.**VALID_PRECISION: Set[_PRECISION]**

deepmd.common.**VALID_ACTIVATION: Set[_ACTIVATION]**

deepmd.common.**data_requirement**

deepmd.common.**add_data_requirement**(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: bool | None = None, repeat: int = 1, default: float = 0.0, dtype: numpy.dtype | None = None, output_natoms_for_type_sel: bool = False)

Specify data requirements for training.

Parameters

key
: [str] type of data stored in corresponding *.npy file e.g. forces or energy

ndof
: [int] number of the degrees of freedom, this is tied to atomic parameter e.g. forces have atomic=True and ndof=3

atomic
: [bool, optional] specifies whwther the ndof keyworrd applies to per atom quantity or not, by default False

must
[bool, optional] specifi if the *.npy data file must exist, by default False

high_prec
[bool, optional] if true load data to np.float64 else np.float32, by default False

type_sel
[bool, optional] select only certain type of atoms, by default None

repeat
[int, optional] if specify repaeat data repeat times, by default 1

default
[float, optional, default=0.] default value of data

dtype
[np.dtype, optional] the dtype of data, overwrites high_prec if provided

output_natoms_for_type_sel
[bool, optional] if True and type_sel is True, the atomic dimension will be natoms instead of nsel

deepmd.common.**select_idx_map**(atom_types: numpy.ndarray, select_types: numpy.ndarray) → numpy.ndarray

Build map of indices for element supplied element types from all atoms list.

Parameters

atom_types
[np.ndarray] array specifing type for each atoms as integer

select_types
[np.ndarray] types of atoms you want to find indices for

Returns

np.ndarray
indices of types of atoms defined by select_types in atom_types array

> Warning: select_types array will be sorted before finding indices in atom_types

deepmd.common.**make_default_mesh**(pbc: bool, mixed_type: bool) → numpy.ndarray

Make mesh.

Only the size of mesh matters, not the values: * 6 for PBC, no mixed types * 0 for no PBC, no mixed types * 7 for PBC, mixed types * 1 for no PBC, mixed types

Parameters

pbc
[bool] if True, the mesh will be made for periodic boundary conditions

mixed_type
[bool] if True, the mesh will be made for mixed types

Returns

np.ndarray
mesh

deepmd.common.**j_must_have**(jdata: Dict[str, _DICT_VAL], key: str, deprecated_key: List[str] = []) → _DICT_VAL

Assert that supplied dictionary conaines specified key.

Returns

_DICT_VAL
value that was store unde supplied key

Raises

RuntimeError
if the key is not present

deepmd.common.**j_loader**(filename: str | pathlib.Path) → Dict[str, Any]

Load yaml or json settings file.

Parameters

filename
[Union[str, Path]] path to file

Returns

Dict[**str**, ***Any***]
loaded dictionary

Raises

TypeError
if the supplied file is of unsupported type

deepmd.common.**expand_sys_str**(root_dir: str | pathlib.Path) → List[str]

Recursively iterate over directories taking those that contain type.raw file.

Parameters

root_dir
[Union[str, Path]] starting directory

Returns

List[**str**]
list of string pointing to system directories

deepmd.common.**get_np_precision**(precision: _PRECISION) → numpy.dtype

Get numpy precision constant from string.

Parameters

precision
[_PRECISION] string name of numpy constant or default

Returns

np.dtype
numpy presicion constant

Raises

RuntimeError
if string is invalid

20.2.4 deepmd.driver

dpdata driver.

Module Contents

Classes

DPDriver	DeePMD-kit driver.

class deepmd.driver.DPDriver(dp: str)

Bases: dpdata.driver.Driver

DeePMD-kit driver.

Parameters

dp
[deepmd.DeepPot or str] The deepmd-kit potential class or the filename of the model.

Examples

```
>>> DPDriver("frozen_model.pb")
```

label(data: dict) → dict

Label a system data by deepmd-kit. Returns new data with energy, forces, and virials.

Parameters

data
[dict] data with coordinates and atom types

Returns

dict
labeled data with energies and forces

20.2.5 deepmd.env

Module Contents

deepmd.env.SHARED_LIB_MODULE = 'lib'

deepmd.env.SHARED_LIB_DIR

deepmd.env.GLOBAL_NP_FLOAT_PRECISION

deepmd.env.GLOBAL_CONFIG

20.2.6 deepmd.main

The entry points for DeePMD-kit.

If only printing the help message, this module does not call the main DeePMD-kit module to avoid the slow import of TensorFlow.

Module Contents

Classes

RawTextArgumentDefaultsHelpFormatter	This formatter is used to print multile-line help message with default value.
BackendOption	Map backend alias to unique name.
DeprecateAction	Information about how to convert command line strings to Python objects.

Functions

get_ll(→ int)	Convert string to python logging level.
main_parser(→ argparse.ArgumentParser)	DeePMD-Kit commandline options argument parser.
parse_args(→ argparse.Namespace)	Parse arguments and convert argument strings to objects.
main()	DeePMD-kit new entry point.

Attributes

__version__
BACKENDS
BACKEND_TABLE

deepmd.main.**__version__** = 'unknown'

deepmd.main.**get_ll**(log_level: str) → int

Convert string to python logging level.

Parameters

log_level
[str] allowed input values are: DEBUG, INFO, WARNING, ERROR, 3, 2, 1, 0

Returns

int
one of python logging module log levels - 10, 20, 30 or 40

class deepmd.main.RawTextArgumentDefaultsHelpFormatter(prog, indent_increment=2, max_help_position=24, width=None)

Bases: argparse.RawTextHelpFormatter, argparse.ArgumentDefaultsHelpFormatter

This formatter is used to print multile-line help message with default value.

deepmd.main.BACKENDS: Dict[str, Type[deepmd.backend.backend.Backend]]

deepmd.main.BACKEND_TABLE: Dict[str, str]

class deepmd.main.BackendOption(option_strings, dest, nargs=None, const=None, default=None, type=None, choices=None, required=False, help=None, metavar=None)

Bases: argparse.Action

Map backend alias to unique name.

__call__(parser, namespace, values, option_string=None)

class deepmd.main.DeprecateAction(*args, **kwargs)

Bases: argparse.Action

Information about how to convert command line strings to Python objects.

Action objects are used by an ArgumentParser to represent the information needed to parse a single argument from one or more strings from the command line. The keyword arguments to the Action constructor are also all attributes of Action instances.

Keyword Arguments

- which (*- option_strings -- A list of command-line option strings*) – should be associated with this action.
- object(*- dest -- The name of the attribute to hold the created*)
- be (*- nargs -- The number of command-line arguments that should*) – consumed. By default, one argument will be consumed and a single value will be produced. Other values include:
  - N (an integer) consumes N arguments (and produces a list)
  - '?' consumes zero or one arguments
  - '*' consumes zero or more arguments (and produces a list)
  - '+' consumes one or more arguments (and produces a list)

  Note that the difference between the default and nargs=1 is that with the default, a single value will be produced, while with nargs=1, a list containing a single value will be produced.
- the (*- metavar -- The name to be used for the option's argument with*) – option uses an action that takes no values.
- specified. (*- default -- The value to be produced if the option is not*)
- and (*- type -- A callable that accepts a single string argument,*) – returns the converted value. The standard Python types str, int, float, and complex are useful examples of such callables. If None, str is used.
- None, (*- choices -- A container of values that should be allowed. If not*) – after a command-line argument has been converted to the appropriate type, an exception will be raised if it is not a member of this collection.

- **the** – command line. This is only meaningful for optional command-line arguments.
- **argument.** (*- help -- The help string describing the*)
- **the** – help string. If None, the 'dest' value will be used as the name.

__call__(parser, namespace, values, option_string=None)

deepmd.main.**main_parser**() → argparse.ArgumentParser

DeePMD-Kit commandline options argument parser.

Returns

argparse.ArgumentParser
main parser of DeePMD-kit

deepmd.main.**parse_args**(args: List[str] | None = None) → argparse.Namespace

Parse arguments and convert argument strings to objects.

Parameters

args
[List[str]] list of command line arguments, main purpose is testing default option None takes arguments from sys.argv

Returns

argparse.Namespace
the populated namespace

deepmd.main.**main**()

DeePMD-kit new entry point.

Raises

RuntimeError
if no command was input

20.3 Package Contents

20.3.1 Functions

DeepPotential(*args, **kwargs)	Factory function that forwards to DeepEval (for compatbility

deepmd.**DeepPotential**(*args, **kwargs)

Factory function that forwards to DeepEval (for compatbility and performance).

Parameters

*args
positional arguments

**kwargs
keyword arguments

Returns

`DeepEval`
: potentials

CHAPTER

TWENTYONE

OP API

21.1 op_module

Python wrappers around TensorFlow ops.

This file is MACHINE GENERATED! Do not edit.

deepmd.tf.env.op_module.**AddFltNvnmd**(x, w, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

class deepmd.tf.env.op_module.**Annotated**(*args, **kwargs)

Bases: object

Add context-specific metadata to a type.

Example: Annotated[int, runtime_check.Unsigned] indicates to the hypothetical runtime_check module that this type is an unsigned int. Every other consumer of this type can ignore this metadata and treat this type as int.

The first argument to Annotated must be a valid type.

Details:

- It's an error to call Annotated with less than two arguments.
- Access the metadata via the __metadata__ attribute:

```
assert Annotated[int, '$'].__metadata__ == ('$',)
```

- Nested Annotated types are flattened:

```
assert Annotated[Annotated[T, Ann1, Ann2], Ann3] == Annotated[T, Ann1, Ann2, Ann3]
```

- Instantiating an annotated type is equivalent to instantiating the

underlying type:

```
assert Annotated[C, Ann1](5) == C(5)
```

- Annotated can be used as a generic type alias:

```
Optimized: TypeAlias = Annotated[T, runtime.Optimize()]
assert Optimized[int] == Annotated[int, runtime.Optimize()]

OptimizedList: TypeAlias = Annotated[list[T], runtime.Optimize()]
assert OptimizedList[int] == Annotated[list[int], runtime.Optimize()]
```

- Annotated cannot be used with an unpacked TypeVarTuple:

```
Variadic: TypeAlias = Annotated[*Ts, Ann1]  # NOT valid
```

This would be equivalent to:

```
Annotated[T1, T2, T3, ..., Ann1]
```

where T1, T2 etc. are TypeVars, which would be invalid, because only one type should be passed to Annotated.

class deepmd.tf.env.op_module.**Any**(*args, **kwargs)

Bases: object

Special type indicating an unconstrained type.

- Any is compatible with every type.
- Any assumed to have all methods.
- All values assumed to be instances of Any.

Note that all the above statements are true from the point of view of static type checkers. At runtime, Any should not be used with instance checks.

deepmd.tf.env.op_module.**ConvertForwardMap**(sub_forward_map, sub_natoms, natoms, name=None)

TODO: add doc.

Parameters

- **sub_forward_map** – A Tensor of type int32.
- **sub_natoms** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (forward_map, backward_map, new_natoms, mesh).

forward_map: A Tensor of type int32. backward_map: A Tensor of type int32. new_natoms: A Tensor of type int32. mesh: A Tensor of type int32.

deepmd.tf.env.op_module.**CopyFltNvnmd**(x, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.

- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (y1, y2).

y1: A Tensor. Has the same type as x. y2: A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**Descrpt**(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, sel_a, sel_r, axis_rule, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **axis_rule** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist, axis, rot_mat).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. axis: A Tensor of type int32. rot_mat: A Tensor. Has the same type as coord.

deepmd.tf.env.op_module.**DescrptNorot**(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.

- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**DescrptSeA**(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**DescrptSeAEf**(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.

- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**DescrptSeAEfPara**(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**DescrptSeAEfVert**(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**DescrptSeAMask**(coord, type, mask, box, natoms, mesh, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **mask** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **natoms** – A Tensor of type int32.
- **mesh** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**DescrptSeR**(coord, type, natoms, box, mesh, davg, dstd, rcut, rcut_smth, sel, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut** – A float.
- **rcut_smth** – A float.
- **sel** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**DotmulFltNvnmd**(x, w, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**DprcPairwiseIdx**(idxs, natoms, name=None)

TODO: add doc.

Parameters

- **idxs** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (forward_qm_map, backward_qm_map, forward_qmmm_map, backward_qmmm_map, natoms_qm, natoms_qmmm, qmmm_frame_idx).

forward_qm_map: A Tensor of type int32. backward_qm_map: A Tensor of type int32. forward_qmmm_map: A Tensor of type int32. backward_qmmm_map: A Tensor of

type int32. natoms_qm: A Tensor of type int32. natoms_qmmm: A Tensor of type int32. qmmm_frame_idx: A Tensor of type int32.

deepmd.tf.env.op_module.**EwaldRecp**(coord, charge, natoms, box, ewald_beta, ewald_h, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **charge** – A Tensor. Must have the same type as coord.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **ewald_beta** – A float.
- **ewald_h** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (energy, force, virial).

energy: A Tensor. Has the same type as coord. force: A Tensor. Has the same type as coord. virial: A Tensor. Has the same type as coord.

deepmd.tf.env.op_module.**FltNvnmd**(x, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**Gelu**(x, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**GeluCustom**(x, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**GeluGrad**(dy, x, name=None)

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as dy.

deepmd.tf.env.op_module.**GeluGradCustom**(dy, x, name=None)

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as dy.

deepmd.tf.env.op_module.**GeluGradGrad**(dy, dy_, x, name=None)

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as dy.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as dy.

deepmd.tf.env.op_module.**GeluGradGradCustom**(dy, dy_, x, name=None)

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as dy.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as dy.

deepmd.tf.env.op_module.**MapAparam**(aparam, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **aparam** – A Tensor. Must be one of the following types: float32, float64.

- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as aparam.

deepmd.tf.env.op_module.**MapFltNvnmd**(x, table, table_grad, table_info, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **table** – A Tensor. Must have the same type as x.
- **table_grad** – A Tensor. Must have the same type as x.
- **table_info** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**MatmulFitnetNvnmd**(x, w, nbitx, nbitw, normw, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **nbitx** – An int.
- **nbitw** – An int.
- **normw** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**MatmulFlt2fixNvnmd**(x, w, nbit, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **nbit** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**MatmulFltNvnmd**(x, w, normx, normw, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **normx** – An int.
- **normw** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**MulFltNvnmd**(x, w, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**NeighborStat**(coord, type, natoms, box, mesh, rcut, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **rcut** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (max_nbor_size, min_nbor_dist).

max_nbor_size: A Tensor of type int32. min_nbor_dist: A Tensor. Has the same type as coord.

deepmd.tf.env.op_module.**PairTab**(table_info, table_data, type, rij, nlist, natoms, scale, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **table_info** – A Tensor of type float64.
- **table_data** – A Tensor of type float64.

- **type** – A Tensor of type int32.
- **rij** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **scale** – A Tensor. Must have the same type as rij.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (atom_energy, force, atom_virial).

atom_energy: A Tensor. Has the same type as rij. force: A Tensor. Has the same type as rij. atom_virial: A Tensor. Has the same type as rij.

deepmd.tf.env.op_module.**ParallelProdForceSeA**(net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel, parallel=False, start_frac=0, end_frac=1, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **parallel** – An optional bool. Defaults to False.
- **start_frac** – An optional float. Defaults to 0.
- **end_frac** – An optional float. Defaults to 1.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**ProdEnvMatA**(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

Compute the environment matrix for descriptor se_e2_a.

Each row of the environment matrix $\mathcal{R}^i$ can be constructed as follows

$$(\mathcal{R}^i)_j = [\begin{array}{c} s(r_{ji}) \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{array}]$$

In the above equation, $\mathrm{R}_{ji} = \mathrm{R}_j - \mathrm{R}_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \|\mathrm{R}_{ji}\|$ is its norm. The switching function $s(r)$ is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r}\{(\frac{r-r_s}{r_c-r_s})^3(-6(\frac{r-r_s}{r_c-r_s})^2 + 15\frac{r-r_s}{r_c-r_s} - 10) + 1\}, & r_s \le r < r_c \\ 0, & r \ge r_c \end{cases}$$

Note that the environment matrix is normalized by davg and dstd.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64. The coordinates of atoms.
- **type** – A Tensor of type int32. The types of atoms.
- **natoms** – A Tensor of type int32. The number of atoms. This tensor has the length of Ntypes + 2. natoms[0]: number of local atoms. natoms[1]: total number of atoms held by this processor. natoms[i]: 2 <= i < Ntypes+2, number of type i atoms.
- **box** – A Tensor. Must have the same type as coord. The box of frames.
- **mesh** – A Tensor of type int32. Gor historical reasons, only the length of the Tensor matters. If size of mesh == 6, pbc is assumed. If size of mesh == 0, no-pbc is assumed.
- **davg** – A Tensor. Must have the same type as coord. Average value of the environment matrix for normalization.
- **dstd** – A Tensor. Must have the same type as coord. Standard deviation of the environment matrix for normalization.
- **rcut_a** – A float. This argument is not used.
- **rcut_r** – A float. The cutoff radius for the environment matrix.
- **rcut_r_smth** – A float. From where the environment matrix should be smoothed.
- **sel_a** – A list of ints. sel_a[i] specifies the maxmum number of type i atoms in the cut-off radius.
- **sel_r** – A list of ints. This argument is not used.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. The environment matrix. descrpt_deriv: A Tensor. Has the same type as coord. The derivative of the environment matrix. rij: A Tensor. Has the same type as coord. The distance between the atoms. nlist: A Tensor of type int32. The neighbor list of each atom.

`deepmd.tf.env.op_module.ProdEnvMatAMix`(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

Compute the environment matrix mixing the atom types.

The sorting of neighbor atoms depends not on atom types, but on the distance and index. The atoms in nlist matrix will gather forward and thus save space for gaps of types in ProdEnvMatA, resulting in optimized and relative small sel_a.

The additional outputs are listed as following:

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist, ntype, nmask).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. ntype: A Tensor of type int32. The corresponding atom types in nlist. nmask: A Tensor of type bool. The atom mask in nlist.

deepmd.tf.env.op_module.**ProdEnvMatAMixNvnmdQuantize**(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.

- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist, ntype, nmask).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. ntype: A Tensor of type int32. nmask: A Tensor of type bool.

deepmd.tf.env.op_module.**ProdEnvMatANvnmdQuantize**(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**ProdEnvMatR**(coord, type, natoms, box, mesh, davg, dstd, rcut, rcut_smth, sel, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.

- **dstd** – A Tensor. Must have the same type as coord.
- **rcut** – A float.
- **rcut_smth** – A float.
- **sel** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**ProdForce**(net_deriv, in_deriv, nlist, axis, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**ProdForceNorot**(net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**ProdForceSeA**(net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**ProdForceSeAMask**(net_deriv, in_deriv, mask, nlist, total_atom_num, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **mask** – A Tensor of type int32.
- **nlist** – A Tensor of type int32.
- **total_atom_num** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**ProdForceSeR**(net_deriv, in_deriv, nlist, natoms, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**ProdVirial**(net_deriv, in_deriv, rij, nlist, axis, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **rij** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**ProdVirialNorot**(net_deriv, in_deriv, rij, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **rij** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**ProdVirialSeA**(net_deriv, in_deriv, rij, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **rij** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.

- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**ProdVirialSeR**(net_deriv, in_deriv, rij, nlist, natoms, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **rij** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**QuantizeNvnmd**(x, isround, nbit1, nbit2, nbit3, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **isround** – An int.
- **nbit1** – An int.
- **nbit2** – An int.
- **nbit3** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**SoftMinForce**(du, sw_deriv, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **du** – A Tensor. Must be one of the following types: float32, float64.
- **sw_deriv** – A Tensor. Must have the same type as du.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.

- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as du.

deepmd.tf.env.op_module.**SoftMinSwitch**(type, rij, nlist, natoms, sel_a, sel_r, alpha, rmin, rmax, name=None)

TODO: add doc.

Parameters

- **type** – A Tensor of type int32.
- **rij** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **alpha** – A float.
- **rmin** – A float.
- **rmax** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (sw_value, sw_deriv).

sw_value: A Tensor. Has the same type as rij. sw_deriv: A Tensor. Has the same type as rij.

deepmd.tf.env.op_module.**SoftMinVirial**(du, sw_deriv, rij, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **du** – A Tensor. Must be one of the following types: float32, float64.
- **sw_deriv** – A Tensor. Must have the same type as du.
- **rij** – A Tensor. Must have the same type as du.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as du. atom_virial: A Tensor. Has the same type as du.

deepmd.tf.env.op_module.**TabulateFusion**(table, table_info, em_x, em, last_layer_size, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**TabulateFusionGrad**(table, table_info, em_x, em, dy, descriptor, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**TabulateFusionGradGrad**(table, table_info, em_x, em, dz_dy_dem_x, dz_dy_dem, descriptor, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**TabulateFusionSeA**(table, table_info, em_x, em, last_layer_size, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**TabulateFusionSeAGrad**(table, table_info, em_x, em, dy, descriptor, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**TabulateFusionSeAGradGrad**(table, table_info, em_x, em, dz_dy_dem_x, dz_dy_dem, descriptor, is_sorted=True, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.

- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **is_sorted** – An optional bool. Defaults to True.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**TabulateFusionSeAtten**(table, table_info, em_x, em, two_embed, last_layer_size, is_sorted=True, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **two_embed** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **is_sorted** – An optional bool. Defaults to True.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**TabulateFusionSeAttenGrad**(table, table_info, em_x, em, two_embed, dy, descriptor, is_sorted=True, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **two_embed** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **is_sorted** – An optional bool. Defaults to True.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem, dy_dtwo).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table. dy_dtwo: A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**TabulateFusionSeAttenGradGrad**(table, table_info, em_x, em, two_embed, dz_dy_dem_x, dz_dy_dem, dz_dy_dtwo, descriptor, is_sorted=True, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **two_embed** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **dz_dy_dtwo** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **is_sorted** – An optional bool. Defaults to True.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**TabulateFusionSeR**(table, table_info, em, last_layer_size, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**TabulateFusionSeRGrad**(table, table_info, em, dy, descriptor, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**TabulateFusionSeRGradGrad**(table, table_info, em, dz_dy_dem, descriptor, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**TabulateFusionSeT**(table, table_info, em_x, em, last_layer_size, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**TabulateFusionSeTGrad**(table, table_info, em_x, em, dy, descriptor, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**TabulateFusionSeTGradGrad**(table, table_info, em_x, em, dz_dy_dem_x, dz_dy_dem, descriptor, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**Tanh4FltNvnmd**(x, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**UnaggregatedDy2Dx**(z, w, dy_dx, dy2_dx, ybar, functype, name=None)

TODO: add doc.

Parameters

- **z** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as z.
- **dy_dx** – A Tensor. Must have the same type as z.
- **dy2_dx** – A Tensor. Must have the same type as z.
- **ybar** – A Tensor. Must have the same type as z.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as z.

deepmd.tf.env.op_module.**UnaggregatedDy2DxS**(y, dy, w, xbar, functype, name=None)

TODO: add doc.

Parameters

- **y** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as y.
- **w** – A Tensor. Must have the same type as y.
- **xbar** – A Tensor. Must have the same type as y.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as y.

deepmd.tf.env.op_module.**UnaggregatedDyDx**(z, w, dy_dx, ybar, functype, name=None)

TODO: add doc.

Parameters

- **z** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as z.
- **dy_dx** – A Tensor. Must have the same type as z.
- **ybar** – A Tensor. Must have the same type as z.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as z.

deepmd.tf.env.op_module.**UnaggregatedDyDxS**(y, w, xbar, functype, name=None)

TODO: add doc.

Parameters

- **y** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as y.
- **xbar** – A Tensor. Must have the same type as y.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as y.

deepmd.tf.env.op_module.**add_flt_nvnmd**(x: Any, w: Any, name=None) → Any

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**convert_forward_map**(sub_forward_map: Any, sub_natoms: Any, natoms: Any, name=None)

TODO: add doc.

Parameters

- **sub_forward_map** – A Tensor of type int32.
- **sub_natoms** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (forward_map, backward_map, new_natoms, mesh).

forward_map: A Tensor of type int32. backward_map: A Tensor of type int32. new_natoms: A Tensor of type int32. mesh: A Tensor of type int32.

deepmd.tf.env.op_module.**copy_flt_nvnmd**(x: Any, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (y1, y2).

y1: A Tensor. Has the same type as x. y2: A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**descrpt**(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, sel_a, sel_r, axis_rule, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.

- **axis_rule** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist, axis, rot_mat).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. axis: A Tensor of type int32. rot_mat: A Tensor. Has the same type as coord.

deepmd.tf.env.op_module.**descrpt_norot**(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth: float, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**descrpt_se_a**(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth: float, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.

- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**descrpt_se_a_ef**(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, ef: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth: float, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**descrpt_se_a_ef_para**(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, ef: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth: float, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**descrpt_se_a_ef_vert**(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, ef: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth: float, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.

- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**descrpt_se_a_mask**(coord: Any, type: Any, mask: Any, box: Any, natoms: Any, mesh: Any, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **mask** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **natoms** – A Tensor of type int32.
- **mesh** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**descrpt_se_r**(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, davg: Any, dstd: Any, rcut: float, rcut_smth: float, sel, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut** – A float.
- **rcut_smth** – A float.

- **sel** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**dotmul_flt_nvnmd**(x: Any, w: Any, name=None) → Any

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**dprc_pairwise_idx**(idxs: Any, natoms: Any, name=None)

TODO: add doc.

Parameters

- **idxs** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (forward_qm_map, backward_qm_map, forward_qmmm_map, backward_qmmm_map, natoms_qm, natoms_qmmm, qmmm_frame_idx).

forward_qm_map: A Tensor of type int32. backward_qm_map: A Tensor of type int32. forward_qmmm_map: A Tensor of type int32. backward_qmmm_map: A Tensor of type int32. natoms_qm: A Tensor of type int32. natoms_qmmm: A Tensor of type int32. qmmm_frame_idx: A Tensor of type int32.

deepmd.tf.env.op_module.**ewald_recp**(coord: Any, charge: Any, natoms: Any, box: Any, ewald_beta: float, ewald_h: float, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **charge** – A Tensor. Must have the same type as coord.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **ewald_beta** – A float.
- **ewald_h** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (energy, force, virial).

energy: A Tensor. Has the same type as coord. force: A Tensor. Has the same type as coord. virial: A Tensor. Has the same type as coord.

deepmd.tf.env.op_module.**flt_nvnmd**(x: Any, name=None) → Any

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**gelu**(x: Any, name=None) → Any

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**gelu_custom**(x: Any, name=None) → Any

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**gelu_grad**(dy: Any, x: Any, name=None) → Any

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as dy.

deepmd.tf.env.op_module.**gelu_grad_custom**(dy: Any, x: Any, name=None) → Any

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as dy.

deepmd.tf.env.op_module.**gelu_grad_grad**(dy: Any, dy_: Any, x: Any, name=None) → Any

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as dy.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as dy.

deepmd.tf.env.op_module.**gelu_grad_grad_custom**(dy: Any, dy_: Any, x: Any, name=None) → Any

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as dy.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as dy.

deepmd.tf.env.op_module.**map_aparam**(aparam: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None) → Any

TODO: add doc.

Parameters

- **aparam** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as aparam.

deepmd.tf.env.op_module.**map_flt_nvnmd**(x: Any, table: Any, table_grad: Any, table_info: Any, name=None) → Any

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **table** – A Tensor. Must have the same type as x.
- **table_grad** – A Tensor. Must have the same type as x.

- **table_info** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**matmul_fitnet_nvnmd**(x: Any, w: Any, nbitx: int, nbitw: int, normw: int, name=None) → Any

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **nbitx** – An int.
- **nbitw** – An int.
- **normw** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**matmul_flt2fix_nvnmd**(x: Any, w: Any, nbit: int, name=None) → Any

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **nbit** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**matmul_flt_nvnmd**(x: Any, w: Any, normx: int, normw: int, name=None) → Any

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **normx** – An int.
- **normw** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**mul_flt_nvnmd**(x: Any, w: Any, name=None) → Any

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**neighbor_stat**(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, rcut: float, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **rcut** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (max_nbor_size, min_nbor_dist).

max_nbor_size: A Tensor of type int32. min_nbor_dist: A Tensor. Has the same type as coord.

deepmd.tf.env.op_module.**pair_tab**(table_info: Any, table_data: Any, type: Any, rij: Any, nlist: Any, natoms: Any, scale: Any, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **table_info** – A Tensor of type float64.
- **table_data** – A Tensor of type float64.
- **type** – A Tensor of type int32.
- **rij** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **scale** – A Tensor. Must have the same type as rij.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (atom_energy, force, atom_virial).

atom_energy: A Tensor. Has the same type as rij. force: A Tensor. Has the same type as rij. atom_virial: A Tensor. Has the same type as rij.

deepmd.tf.env.op_module.**parallel_prod_force_se_a**(net_deriv: Any, in_deriv: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, parallel: bool = False, start_frac: float = 0, end_frac: float = 1, name=None) → Any

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **parallel** – An optional bool. Defaults to False.
- **start_frac** – An optional float. Defaults to 0.
- **end_frac** – An optional float. Defaults to 1.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**prod_env_mat_a**(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth: float, sel_a, sel_r, name=None)

Compute the environment matrix for descriptor se_e2_a.

Each row of the environment matrix $\mathcal{R}^i$ can be constructed as follows

$$(\mathcal{R}^i)_j = [\ \frac{s(r_{ji})}{\frac{s(r_{ji})x_{ji}}{r_{ji}}}\ \frac{\frac{s(r_{ji})y_{ji}}{r_{ji}}}{\frac{s(r_{ji})z_{ji}}{r_{ji}}}\ ]$$

In the above equation, $\mathrm{R}_{ji} = \mathrm{R}_j - \mathrm{R}_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \|\mathrm{R}_{ji}\|$ is its norm. The switching function $s(r)$ is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r}\{(\frac{r-r_s}{r_c-r_s})^3(-6(\frac{r-r_s}{r_c-r_s})^2 + 15\frac{r-r_s}{r_c-r_s} - 10) + 1\}, & r_s \le r < r_c \\ 0, & r \ge r_c \end{cases}$$

Note that the environment matrix is normalized by davg and dstd.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64. The coordinates of atoms.
- **type** – A Tensor of type int32. The types of atoms.

- **natoms** – A Tensor of type int32. The number of atoms. This tensor has the length of Ntypes + 2. natoms[0]: number of local atoms. natoms[1]: total number of atoms held by this processor. natoms[i]: 2 <= i < Ntypes+2, number of type i atoms.
- **box** – A Tensor. Must have the same type as coord. The box of frames.
- **mesh** – A Tensor of type int32. Gor historical reasons, only the length of the Tensor matters. If size of mesh == 6, pbc is assumed. If size of mesh == 0, no-pbc is assumed.
- **davg** – A Tensor. Must have the same type as coord. Average value of the environment matrix for normalization.
- **dstd** – A Tensor. Must have the same type as coord. Standard deviation of the environment matrix for normalization.
- **rcut_a** – A float. This argument is not used.
- **rcut_r** – A float. The cutoff radius for the environment matrix.
- **rcut_r_smth** – A float. From where the environment matrix should be smoothed.
- **sel_a** – A list of ints. sel_a[i] specifies the maxmum number of type i atoms in the cut-off radius.
- **sel_r** – A list of ints. This argument is not used.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. The environment matrix. descrpt_deriv: A Tensor. Has the same type as coord. The derivative of the environment matrix. rij: A Tensor. Has the same type as coord. The distance between the atoms. nlist: A Tensor of type int32. The neighbor list of each atom.

deepmd.tf.env.op_module.**prod_env_mat_a_mix**(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth: float, sel_a, sel_r, name=None)

Compute the environment matrix mixing the atom types.

The sorting of neighbor atoms depends not on atom types, but on the distance and index. The atoms in nlist matrix will gather forward and thus save space for gaps of types in ProdEnvMatA, resulting in optimized and relative small sel_a.

The additional outputs are listed as following:

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.

- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist, ntype, nmask).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. ntype: A Tensor of type int32. The corresponding atom types in nlist. nmask: A Tensor of type bool. The atom mask in nlist.

deepmd.tf.env.op_module.**prod_env_mat_a_mix_nvnmd_quantize**(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth: float, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist, ntype, nmask).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. ntype: A Tensor of type int32. nmask: A Tensor of type bool.

deepmd.tf.env.op_module.**prod_env_mat_a_nvnmd_quantize**(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth: float, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**prod_env_mat_r**(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, davg: Any, dstd: Any, rcut: float, rcut_smth: float, sel, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut** – A float.
- **rcut_smth** – A float.
- **sel** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

deepmd.tf.env.op_module.**prod_force**(net_deriv: Any, in_deriv: Any, nlist: Any, axis: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None) → Any

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**prod_force_norot**(net_deriv: Any, in_deriv: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None) → Any

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**prod_force_se_a**(net_deriv: Any, in_deriv: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None) → Any

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.

- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**prod_force_se_a_mask**(net_deriv: Any, in_deriv: Any, mask: Any, nlist: Any, total_atom_num: int, name=None) → Any

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **mask** – A Tensor of type int32.
- **nlist** – A Tensor of type int32.
- **total_atom_num** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**prod_force_se_r**(net_deriv: Any, in_deriv: Any, nlist: Any, natoms: Any, name=None) → Any

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**prod_virial**(net_deriv: Any, in_deriv: Any, rij: Any, nlist: Any, axis: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **rij** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.

- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**prod_virial_norot**(net_deriv: Any, in_deriv: Any, rij: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **rij** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**prod_virial_se_a**(net_deriv: Any, in_deriv: Any, rij: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **rij** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**prod_virial_se_r**(net_deriv: Any, in_deriv: Any, rij: Any, nlist: Any, natoms: Any, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **rij** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

deepmd.tf.env.op_module.**quantize_nvnmd**(x: Any, isround: int, nbit1: int, nbit2: int, nbit3: int, name=None) → Any

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **isround** – An int.
- **nbit1** – An int.
- **nbit2** – An int.
- **nbit3** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**soft_min_force**(du: Any, sw_deriv: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None) → Any

TODO: add doc.

Parameters

- **du** – A Tensor. Must be one of the following types: float32, float64.
- **sw_deriv** – A Tensor. Must have the same type as du.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.

- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as du.

deepmd.tf.env.op_module.**soft_min_switch**(type: Any, rij: Any, nlist: Any, natoms: Any, sel_a, sel_r, alpha: float, rmin: float, rmax: float, name=None)

TODO: add doc.

Parameters

- **type** – A Tensor of type int32.
- **rij** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **alpha** – A float.
- **rmin** – A float.
- **rmax** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (sw_value, sw_deriv).

sw_value: A Tensor. Has the same type as rij. sw_deriv: A Tensor. Has the same type as rij.

deepmd.tf.env.op_module.**soft_min_virial**(du: Any, sw_deriv: Any, rij: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None)

TODO: add doc.

Parameters

- **du** – A Tensor. Must be one of the following types: float32, float64.
- **sw_deriv** – A Tensor. Must have the same type as du.
- **rij** – A Tensor. Must have the same type as du.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as du. atom_virial: A Tensor. Has the same type as du.

deepmd.tf.env.op_module.**tabulate_fusion**(table: Any, table_info: Any, em_x: Any, em: Any, last_layer_size: int, name=None) → Any

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tabulate_fusion_grad**(table: Any, table_info: Any, em_x: Any, em: Any, dy: Any, descriptor: Any, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tabulate_fusion_grad_grad**(table: Any, table_info: Any, em_x: Any, em: Any, dz_dy_dem_x: Any, dz_dy_dem: Any, descriptor: Any, name=None) → Any

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.

- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tabulate_fusion_se_a**(table: Any, table_info: Any, em_x: Any, em: Any, last_layer_size: int, name=None) → Any

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tabulate_fusion_se_a_grad**(table: Any, table_info: Any, em_x: Any, em: Any, dy: Any, descriptor: Any, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tabulate_fusion_se_a_grad_grad**(table: Any, table_info: Any, em_x: Any, em: Any, dz_dy_dem_x: Any, dz_dy_dem: Any, descriptor: Any, is_sorted: bool = True, name=None) → Any

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **is_sorted** – An optional bool. Defaults to True.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tabulate_fusion_se_atten**(table: Any, table_info: Any, em_x: Any, em: Any, two_embed: Any, last_layer_size: int, is_sorted: bool = True, name=None) → Any

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **two_embed** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **is_sorted** – An optional bool. Defaults to True.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tabulate_fusion_se_atten_grad**(table: Any, table_info: Any, em_x: Any, em: Any, two_embed: Any, dy: Any, descriptor: Any, is_sorted: bool = True, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **two_embed** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.

- **is_sorted** – An optional bool. Defaults to True.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem, dy_dtwo).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table. dy_dtwo: A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tabulate_fusion_se_atten_grad_grad**(table: Any, table_info: Any, em_x: Any, em: Any, two_embed: Any, dz_dy_dem_x: Any, dz_dy_dem: Any, dz_dy_dtwo: Any, descriptor: Any, is_sorted: bool = True, name=None) → Any

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **two_embed** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **dz_dy_dtwo** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **is_sorted** – An optional bool. Defaults to True.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tabulate_fusion_se_r**(table: Any, table_info: Any, em: Any, last_layer_size: int, name=None) → Any

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tabulate_fusion_se_r_grad**(table: Any, table_info: Any, em: Any, dy: Any, descriptor: Any, name=None) → Any

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tabulate_fusion_se_r_grad_grad**(table: Any, table_info: Any, em: Any, dz_dy_dem: Any, descriptor: Any, name=None) → Any

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tabulate_fusion_se_t**(table: Any, table_info: Any, em_x: Any, em: Any, last_layer_size: int, name=None) → Any

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tabulate_fusion_se_t_grad**(table: Any, table_info: Any, em_x: Any, em: Any, dy: Any, descriptor: Any, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tabulate_fusion_se_t_grad_grad**(table: Any, table_info: Any, em_x: Any, em: Any, dz_dy_dem_x: Any, dz_dy_dem: Any, descriptor: Any, name=None) → Any

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

deepmd.tf.env.op_module.**tanh4_flt_nvnmd**(x: Any, name=None) → Any

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

deepmd.tf.env.op_module.**unaggregated_dy2_dx**(z: Any, w: Any, dy_dx: Any, dy2_dx: Any, ybar: Any, functype: Any, name=None) → Any

TODO: add doc.

Parameters

- **z** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as z.
- **dy_dx** – A Tensor. Must have the same type as z.
- **dy2_dx** – A Tensor. Must have the same type as z.
- **ybar** – A Tensor. Must have the same type as z.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as z.

deepmd.tf.env.op_module.**unaggregated_dy2_dx_s**(y: Any, dy: Any, w: Any, xbar: Any, functype: Any, name=None) → Any

TODO: add doc.

Parameters

- **y** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as y.
- **w** – A Tensor. Must have the same type as y.
- **xbar** – A Tensor. Must have the same type as y.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as y.

deepmd.tf.env.op_module.**unaggregated_dy_dx**(z: Any, w: Any, dy_dx: Any, ybar: Any, functype: Any, name=None) → Any

TODO: add doc.

Parameters

- **z** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as z.
- **dy_dx** – A Tensor. Must have the same type as z.
- **ybar** – A Tensor. Must have the same type as z.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as z.

deepmd.tf.env.op_module.**unaggregated_dy_dx_s**(y: Any, w: Any, xbar: Any, functype: Any, name=None) → Any

TODO: add doc.

Parameters

- **y** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as y.
- **xbar** – A Tensor. Must have the same type as y.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as y.

21.2 op_grads_module

Python wrappers around TensorFlow ops.

This file is MACHINE GENERATED! Do not edit.

class deepmd.tf.env.op_grads_module.**Annotated**(*args, **kwargs)

Bases: object

Add context-specific metadata to a type.

Example: Annotated[int, runtime_check.Unsigned] indicates to the hypothetical runtime_check module that this type is an unsigned int. Every other consumer of this type can ignore this metadata and treat this type as int.

The first argument to Annotated must be a valid type.

Details:

- It's an error to call Annotated with less than two arguments.
- Access the metadata via the `__metadata__` attribute:

```
assert Annotated[int, '$'].__metadata__ == ('$',)
```

- Nested Annotated types are flattened:

```
assert Annotated[Annotated[T, Ann1, Ann2], Ann3] == Annotated[T, Ann1, Ann2, Ann3]
```

- Instantiating an annotated type is equivalent to instantiating the

underlying type:

```
assert Annotated[C, Ann1](5) == C(5)
```

- Annotated can be used as a generic type alias:

```
Optimized: TypeAlias = Annotated[T, runtime.Optimize()]
assert Optimized[int] == Annotated[int, runtime.Optimize()]
```

(continues on next page)

(continued from previous page)

```
OptimizedList: TypeAlias = Annotated[list[T], runtime.Optimize()]
assert OptimizedList[int] == Annotated[list[int], runtime.Optimize()]
```

- Annotated cannot be used with an unpacked TypeVarTuple:

```
Variadic: TypeAlias = Annotated[*Ts, Ann1]  # NOT valid
```

This would be equivalent to:

```
Annotated[T1, T2, T3, ..., Ann1]
```

where T1, T2 etc. are TypeVars, which would be invalid, because only one type should be passed to Annotated.

class deepmd.tf.env.op_grads_module.**Any**(*args, **kwargs)

Bases: object

Special type indicating an unconstrained type.

- Any is compatible with every type.
- Any assumed to have all methods.
- All values assumed to be instances of Any.

Note that all the above statements are true from the point of view of static type checkers. At runtime, Any should not be used with instance checks.

deepmd.tf.env.op_grads_module.**ProdForceGrad**(grad, net_deriv, in_deriv, nlist, axis, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**ProdForceSeAGrad**(grad, net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.

- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**ProdForceSeAMaskGrad**(grad, net_deriv, in_deriv, mask, nlist, total_atom_num, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **mask** – A Tensor of type int32.
- **nlist** – A Tensor of type int32.
- **total_atom_num** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**ProdForceSeRGrad**(grad, net_deriv, in_deriv, nlist, natoms, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**ProdVirialGrad**(grad, net_deriv, in_deriv, rij, nlist, axis, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**ProdVirialSeAGrad**(grad, net_deriv, in_deriv, rij, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**ProdVirialSeRGrad**(grad, net_deriv, in_deriv, rij, nlist, natoms, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**SoftMinForceGrad**(grad, du, sw_deriv, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **du** – A Tensor. Must have the same type as grad.
- **sw_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**SoftMinVirialGrad**(grad, du, sw_deriv, rij, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **du** – A Tensor. Must have the same type as grad.
- **sw_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**prod_force_grad**(grad: Any, net_deriv: Any, in_deriv: Any, nlist: Any, axis: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None) → Any

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.

- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**prod_force_se_a_grad**(grad: Any, net_deriv: Any, in_deriv: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None) → Any

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**prod_force_se_a_mask_grad**(grad: Any, net_deriv: Any, in_deriv: Any, mask: Any, nlist: Any, total_atom_num: int, name=None) → Any

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **mask** – A Tensor of type int32.
- **nlist** – A Tensor of type int32.
- **total_atom_num** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**prod_force_se_r_grad**(grad: Any, net_deriv: Any, in_deriv: Any, nlist: Any, natoms: Any, name=None) → Any

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**prod_virial_grad**(grad: Any, net_deriv: Any, in_deriv: Any, rij: Any, nlist: Any, axis: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None) → Any

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**prod_virial_se_a_grad**(grad: Any, net_deriv: Any, in_deriv: Any, rij: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None) → Any

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.

- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**prod_virial_se_r_grad**(grad: Any, net_deriv: Any, in_deriv: Any, rij: Any, nlist: Any, natoms: Any, name=None) → Any

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**soft_min_force_grad**(grad: Any, du: Any, sw_deriv: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None) → Any

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **du** – A Tensor. Must have the same type as grad.
- **sw_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

deepmd.tf.env.op_grads_module.**soft_min_virial_grad**(grad: Any, du: Any, sw_deriv: Any, rij: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None) → Any

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **du** – A Tensor. Must have the same type as grad.
- **sw_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

CHAPTER

TWENTYTWO

C++ API

22.1 Class Hierarchy

- Namespace deepmd
  - Struct deepmd_exception
  - Struct NeighborListData
  - Struct tf_exception
  - Class AtomMap
  - Class DeepPot
  - Class DeepPotBase
  - Class DeepPotModelDevi
  - Class DeepPotPT
  - Class DeepPotTF
  - Class DeepTensor
  - Class DeepTensorBase
  - Class DeepTensorTF
  - Class DipoleChargeModifier
  - Class DipoleChargeModifierBase
  - Class DipoleChargeModifierTF
  - Enum DPBackend

22.2 File Hierarchy

- dir_source
  - dir_source_api_cc
    - dir_source_api_cc_include
      - file_source_api_cc_include_AtomMap.h
      - file_source_api_cc_include_common.h

- file_source_api_cc_include_commonTF.h
- file_source_api_cc_include_DataModifier.h
- file_source_api_cc_include_DataModifierTF.h
- file_source_api_cc_include_DeepPot.h
- file_source_api_cc_include_DeepPotPT.h
- file_source_api_cc_include_DeepPotTF.h
- file_source_api_cc_include_DeepTensor.h
- file_source_api_cc_include_DeepTensorTF.h
- file_source_api_cc_include_tf_private.h
- file_source_api_cc_include_tf_public.h

22.3 Full API

22.3.1 Namespaces

Namespace deepmd

Contents

- Classes
- Enums
- Functions
- Typedefs

Classes

- Struct deepmd_exception
- Struct NeighborListData
- Struct tf_exception
- Class AtomMap
- Class DeepPot
- Class DeepPotBase
- Class DeepPotModelDevi
- Class DeepPotPT
- Class DeepPotTF
- Class DeepTensor
- Class DeepTensorBase

- Class DeepTensorTF
- Class DipoleChargeModifier
- Class DipoleChargeModifierBase
- Class DipoleChargeModifierTF

Enums

- Enum DPBackend

Functions

- Function deepmd::check_status
- Function deepmd::convert_pbtxt_to_pb
- Function deepmd::get_env_nthreads
- Function deepmd::load_op_library
- Function deepmd::model_compatable
- Function deepmd::name_prefix
- Function deepmd::print_summary
- Function deepmd::read_file_to_string
- Template Function deepmd::select_by_type
- Template Function deepmd::select_map(typename std::vector<VT>::iterator, const typename std::vector<VT>::const_iterator, const std::vector<int>&, const int&, const int&, const int&, const int&)
- Template Function deepmd::select_map(std::vector<VT>&, const std::vector<VT>&, const std::vector<int>&, const int&, const int&, const int&, const int&)
- Template Function deepmd::select_map_inv(typename std::vector<VT>::iterator, const typename std::vector<VT>::const_iterator, const std::vector<int>&, const int&)
- Template Function deepmd::select_map_inv(std::vector<VT>&, const std::vector<VT>&, const std::vector<int>&, const int&)
- Template Function deepmd::select_real_atoms
- Template Function deepmd::select_real_atoms_coord
- Function deepmd::session_get_dtype
- Template Function deepmd::session_get_scalar
- Template Function deepmd::session_get_vector
- Template Function deepmd::session_input_tensors(std::vector<std::pair<std::string, tensorflow::Tensor>>&, const std::vector<VALUETYPE>&, const int&, const std::vector<int>&, const std::vector<VALUETYPE>&, InputNlist&, const std::vector<VALUETYPE>&, const std::vector<VALUETYPE>&, const deepmd::AtomMap&, const int, const int, const std::string, const bool)

- Template Function deepmd::session_input_tensors(std::vector<std::pair<std::string, tensorflow::Tensor>>&, const std::vector<VALUETYPE>&, const int&, const std::vector<int>&, const std::vector<VALUETYPE>&, const double&, const std::vector<VALUETYPE>&, const std::vector<VALUETYPE>&, const deepmd::AtomMap&, const std::string, const bool)
- Template Function deepmd::session_input_tensors_mixed_type

Typedefs

- Typedef deepmd::ENERGYTYPE
- Typedef deepmd::STRINGTYPE

Namespace tensorflow

22.3.2 Classes and Structs

Struct deepmd_exception

- Defined in file_source_api_cc_include_common.h

Inheritance Relationships

Derived Type

- `public deepmd::tf_exception` (Struct tf_exception)

Struct Documentation

struct **deepmd_exception**

Subclassed by deepmd::tf_exception

Struct NeighborListData

- Defined in file_source_api_cc_include_common.h

Struct Documentation

struct **NeighborListData**

Public Functions

void **copy_from_nlist**(const InputNlist &inlist)

void **shuffle**(const std::vector<int> &fwd_map)

void **shuffle**(const deepmd::AtomMap &map)

void **shuffle_exclude_empty**(const std::vector<int> &fwd_map)

void **make_inlist**(InputNlist &inlist)

void **padding**()

Public Members

std::vector<int> **ilist**

Array stores the core region atom's index.

std::vector<std::vector<int>> **jlist**

Array stores the core region atom's neighbor index.

std::vector<int> **numneigh**

Array stores the number of neighbors of core region atoms.

std::vector<int*> **firstneigh**

Array stores the the location of the first neighbor of core region atoms.

Struct tf_exception

- Defined in file_source_api_cc_include_common.h

Inheritance Relationships

Base Type

- `public deepmd_exception` (Struct deepmd_exception)

Struct Documentation

struct **tf_exception** : public deepmd_exception

Throw exception if TensorFlow doesn't work.

Public Functions

inline **tf_exception**()

inline **tf_exception**(const std::string &msg)

Class AtomMap

- Defined in file_source_api_cc_include_AtomMap.h

Class Documentation

class **AtomMap**

Public Functions

AtomMap()

AtomMap(const std::vector<int>::const_iterator in_begin, const std::vector<int>::const_iterator in_end)

template<typename **VALUETYPE**>
void **forward**(typename std::vector<VALUETYPE>::iterator out, const typename std::vector<VALUETYPE>::const_iterator in, const int stride = 1, const int nframes = 1, const int nall = 0) const

template<typename **VALUETYPE**>
void **backward**(typename std::vector<VALUETYPE>::iterator out, const typename std::vector<VALUETYPE>::const_iterator in, const int stride = 1, const int nframes = 1, const int nall = 0) const

inline const std::vector<int> &**get_type**() const

inline const std::vector<int> &**get_fwd_map**() const

inline const std::vector<int> &**get_bkw_map**() const

Class DeepPot

- Defined in file_source_api_cc_include_DeepPot.h

Class Documentation

class **DeepPot**

Deep Potential to automatically switch backends.

Unnamed Group

template<typename **VALUETYPE**>
void **compute**(ENERGYTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**>
void **compute**(std::vector<ENERGYTYPE> &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Unnamed Group

template<typename **VALUETYPE**>
void **compute**(ENERGYTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &inlist, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**>
void **compute**(std::vector<ENERGYTYPE> &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &inlist, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Unnamed Group

template<typename **VALUETYPE**>
void **compute**(ENERGYTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.

- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**>
void **compute**(std::vector<ENERGYTYPE> &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Unnamed Group

template<typename **VALUETYPE**>
void **compute**(ENERGYTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **lmp_list** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.

- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**>
void **compute**(std::vector<ENERGYTYPE> &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Unnamed Group

template<typename **VALUETYPE**>
void **compute_mixed_type**(ENERGYTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const int &nframes, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, and virial with the mixed type by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The array should be of size nframes x natoms.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**>
void **compute_mixed_type**(std::vector<ENERGYTYPE> &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const int &nframes, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Unnamed Group

template<typename **VALUETYPE**>
void **compute_mixed_type**(ENERGYTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy, std::vector<VALUETYPE> &atom_virial, const int &nframes, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, and virial with the mixed type by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The array should be of size nframes x natoms.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**>
void **compute_mixed_type**(std::vector<ENERGYTYPE> &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy, std::vector<VALUETYPE> &atom_virial, const int &nframes, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Public Functions

`DeepPot()`

DP constructor without initialization.

`~DeepPot()`

`DeepPot(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")`

DP constructor with initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **file_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

void `init(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")`

Initialize the DP.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **file_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

void `print_summary(const std::string &pre) const`

Print the DP summary to the screen.

Parameters

pre – [in] The prefix to each line.

double `cutoff() const`

Get the cutoff radius.

Returns

The cutoff radius.

int `numb_types() const`

Get the number of types.

Returns

The number of types.

int `numb_types_spin() const`

Get the number of types with spin.

Returns

The number of types with spin.

int `dim_fparam() const`

Get the dimension of the frame parameter.

Returns

The dimension of the frame parameter.

int **dim_aparam**() const

Get the dimension of the atomic parameter.

Returns
The dimension of the atomic parameter.

void **get_type_map**(std::string &type_map)

Get the type map (element name of the atom types) of this model.

Parameters
type_map – [out] The type map of this model.

bool **is_aparam_nall**() const

Get whether the atom dimension of aparam is nall instead of fparam.

Parameters
aparam_nall – [out] whether the atom dimension of aparam is nall instead of fparam.

Class DeepPotBase

- Defined in file_source_api_cc_include_DeepPot.h

Inheritance Relationships

Derived Types

- `public deepmd::DeepPotPT` (Class DeepPotPT)
- `public deepmd::DeepPotTF` (Class DeepPotTF)

Class Documentation

class **DeepPotBase**

Deep Potential.

Subclassed by deepmd::DeepPotPT, deepmd::DeepPotTF

Unnamed Group

virtual void **computew**(std::vector<double> &ener, std::vector<double> &force,
std::vector<double> &virial, std::vector<double> &atom_energy,
std::vector<double> &atom_virial, const std::vector<double> &coord, const
std::vector<int> &atype, const std::vector<double> &box, const
std::vector<double> &fparam = std::vector<double>(), const
std::vector<double> &aparam = std::vector<double>()) = 0

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

Note: The double precision interface is used by i-PI, GROMACS, ABACUS, and CP2k.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

virtual void **computew**(std::vector<double> &ener, std::vector<float> &force, std::vector<float> &virial, std::vector<float> &atom_energy, std::vector<float> &atom_virial, const std::vector<float> &coord, const std::vector<int> &atype, const std::vector<float> &box, const std::vector<float> &fparam = std::vector<float>(), const std::vector<float> &aparam = std::vector<float>()) = 0

Unnamed Group

virtual void **computew**(std::vector<double> &ener, std::vector<double> &force, std::vector<double> &virial, std::vector<double> &atom_energy, std::vector<double> &atom_virial, const std::vector<double> &coord, const std::vector<int> &atype, const std::vector<double> &box, const int nghost, const InputNlist &inlist, const int &ago, const std::vector<double> &fparam = std::vector<double>(), const std::vector<double> &aparam = std::vector<double>()) = 0

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

Note: The double precision interface is used by LAMMPS and AMBER.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.

- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **lmp_list** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

virtual void **computew**(std::vector<double> &ener, std::vector<float> &force, std::vector<float> &virial, std::vector<float> &atom_energy, std::vector<float> &atom_virial, const std::vector<float> &coord, const std::vector<int> &atype, const std::vector<float> &box, const int nghost, const InputNlist &inlist, const int &ago, const std::vector<float> &fparam = std::vector<float>(), const std::vector<float> &aparam = std::vector<float>()) = 0

Unnamed Group

virtual void **computew_mixed_type**(std::vector<double> &ener, std::vector<double> &force, std::vector<double> &virial, std::vector<double> &atom_energy, std::vector<double> &atom_virial, const int &nframes, const std::vector<double> &coord, const std::vector<int> &atype, const std::vector<double> &box, const std::vector<double> &fparam = std::vector<double>(), const std::vector<double> &aparam = std::vector<double>()) = 0

Evaluate the energy, force, and virial with the mixed type by using this DP.

Note: At this time, no external program uses this interface.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The array should be of size nframes x natoms.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.

- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

virtual void **computew_mixed_type**(std::vector<double> &ener, std::vector<float> &force, std::vector<float> &virial, std::vector<float> &atom_energy, std::vector<float> &atom_virial, const int &nframes, const std::vector<float> &coord, const std::vector<int> &atype, const std::vector<float> &box, const std::vector<float> &fparam = std::vector<float>(), const std::vector<float> &aparam = std::vector<float>()) = 0

Public Functions

inline **DeepPotBase**()

DP constructor without initialization.

inline virtual **~DeepPotBase**()

DeepPotBase(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")

DP constructor with initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **file_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

virtual void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "") = 0

Initialize the DP.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **file_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

virtual double **cutoff**() const = 0

Get the cutoff radius.

Returns
The cutoff radius.

virtual int **numb_types**() const = 0

Get the number of types.

Returns
The number of types.

virtual int **numb_types_spin**() const = 0

Get the number of types with spin.

Returns
The number of types with spin.

virtual int **dim_fparam**() const = 0

Get the dimension of the frame parameter.

Returns
The dimension of the frame parameter.

virtual int **dim_aparam**() const = 0

Get the dimension of the atomic parameter.

Returns
The dimension of the atomic parameter.

virtual void **get_type_map**(std::string &type_map) = 0

Get the type map (element name of the atom types) of this model.

Parameters
type_map – [out] The type map of this model.

virtual bool **is_aparam_nall**() const = 0

Get whether the atom dimension of aparam is nall instead of fparam.

Parameters
aparam_nall – [out] whether the atom dimension of aparam is nall instead of fparam.

Class DeepPotModelDevi

- Defined in file_source_api_cc_include_DeepPot.h

Class Documentation

class **DeepPotModelDevi**

Public Functions

DeepPotModelDevi()

DP model deviation constructor without initialization.

~DeepPotModelDevi()

DeepPotModelDevi(const std::vector<std::string> &models, const int &gpu_rank = 0, const std::vector<std::string> &file_contents = std::vector<std::string>())

DP model deviation constructor with initialization.

Parameters

- **models** – [in] The names of the frozen model files.
- **gpu_rank** – [in] The GPU rank. Default is 0.

- **file_contents** – [in] The contents of the model files. If it is not empty, DP will read from the strings instead of the files.

void **init**(const std::vector<std::string> &models, const int &gpu_rank = 0, const std::vector<std::string> &file_contents = std::vector<std::string>())

Initialize the DP model deviation contrcutor.

Parameters

- **models** – [in] The names of the frozen model files.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **file_contents** – [in] The contents of the model files. If it is not empty, DP will read from the strings instead of the files.

template<typename **VALUETYPE**>
void **compute**(std::vector<ENERGYTYPE> &all_ener, std::vector<std::vector<VALUETYPE>> &all_force, std::vector<std::vector<VALUETYPE>> &all_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using these DP models.

Parameters

- **all_ener** – [out] The system energies of all models.
- **all_force** – [out] The forces on each atom of all models.
- **all_virial** – [out] The virials of all models.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. dim_aparam. Then all frames and atoms are provided with the same aparam.

template<typename **VALUETYPE**>
void **compute**(std::vector<ENERGYTYPE> &all_ener, std::vector<std::vector<VALUETYPE>> &all_force, std::vector<std::vector<VALUETYPE>> &all_virial, std::vector<std::vector<VALUETYPE>> &all_atom_energy, std::vector<std::vector<VALUETYPE>> &all_atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using these DP models.

Parameters

- **all_ener** – [out] The system energies of all models.
- **all_force** – [out] The forces on each atom of all models.
- **all_virial** – [out] The virials of all models.
- **all_atom_energy** – [out] The atomic energies of all models.
- **all_atom_virial** – [out] The atomic virials of all models.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. dim_aparam. Then all frames and atoms are provided with the same aparam.

template<typename **VALUETYPE**>
void **compute**(std::vector<ENERGYTYPE> &all_ener, std::vector<std::vector<VALUETYPE>> &all_force, std::vector<std::vector<VALUETYPE>> &all_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using these DP models.

Parameters

- **all_ener** – [out] The system energies of all models.
- **all_force** – [out] The forces on each atom of all models.
- **all_virial** – [out] The virials of all models.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **lmp_list** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. dim_aparam. Then all frames and atoms are provided with the same aparam.

template<typename **VALUETYPE**>
void **compute**(std::vector<ENERGYTYPE> &all_ener, std::vector<std::vector<VALUETYPE>> &all_force, std::vector<std::vector<VALUETYPE>> &all_virial, std::vector<std::vector<VALUETYPE>> &all_atom_energy, std::vector<std::vector<VALUETYPE>> &all_atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using these DP models.

Parameters

- **all_ener** – [out] The system energies of all models.
- **all_force** – [out] The forces on each atom of all models.
- **all_virial** – [out] The virials of all models.
- **all_atom_energy** – [out] The atomic energies of all models.
- **all_atom_virial** – [out] The atomic virials of all models.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **lmp_list** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. dim_aparam. Then all frames and atoms are provided with the same aparam.

inline double **cutoff**() const

Get the cutoff radius.

Returns
The cutoff radius.

inline int **numb_types**() const

Get the number of types.

Returns
The number of types.

inline int **numb_types_spin**() const

Get the number of types with spin.

Returns
The number of types with spin.

inline int **dim_fparam**() const

Get the dimension of the frame parameter.

Returns
The dimension of the frame parameter.

inline int **dim_aparam**() const

Get the dimension of the atomic parameter.

Returns
The dimension of the atomic parameter.

template<typename **VALUETYPE**>
void **compute_avg**(VALUETYPE &dener, const std::vector<VALUETYPE> &all_energy)

Compute the average energy.

Parameters

- **dener** – [out] The average energy.
- **all_energy** – [in] The energies of all models.

template<typename **VALUETYPE**>
void **compute_avg**(std::vector<VALUETYPE> &avg, const
std::vector<std::vector<VALUETYPE>> &xx)

Compute the average of vectors.

Parameters

- **avg** – [out] The average of vectors.
- **xx** – [in] The vectors of all models.

template<typename **VALUETYPE**>
void **compute_std**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const
std::vector<std::vector<VALUETYPE>> &xx, const int &stride)

Compute the standard deviation of vectors.

Parameters

- **std** – [out] The standard deviation of vectors.
- **avg** – [in] The average of vectors.
- **xx** – [in] The vectors of all models.
- **stride** – [in] The stride to compute the deviation.

template<typename **VALUETYPE**>
void **compute_relative_std**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE>
&avg, const VALUETYPE eps, const int &stride)

Compute the relative standard deviation of vectors.

Parameters

- **std** – [out] The standard deviation of vectors.
- **avg** – [in] The average of vectors.
- **eps** – [in] The level parameter for computing the deviation.
- **stride** – [in] The stride to compute the deviation.

template<typename **VALUETYPE**>

void **compute_std_e**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const std::vector<std::vector<VALUETYPE>> &xx)

Compute the standard deviation of atomic energies.

Parameters

- **std** – [out] The standard deviation of atomic energies.
- **avg** – [in] The average of atomic energies.
- **xx** – [in] The vectors of all atomic energies.

template<typename **VALUETYPE**>
void **compute_std_f**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const std::vector<std::vector<VALUETYPE>> &xx)

Compute the standard deviation of forces.

Parameters

- **std** – [out] The standard deviation of forces.
- **avg** – [in] The average of forces.
- **xx** – [in] The vectors of all forces.

template<typename **VALUETYPE**>
void **compute_relative_std_f**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const VALUETYPE eps)

Compute the relative standard deviation of forces.

Parameters

- **std** – [out] The relative standard deviation of forces.
- **avg** – [in] The relative average of forces.
- **eps** – [in] The level parameter for computing the deviation.

inline bool **is_aparam_nall**() const

Get whether the atom dimension of aparam is nall instead of fparam.

Parameters

aparam_nall – [out] whether the atom dimension of aparam is nall instead of fparam.

Class DeepPotPT

- Defined in file_source_api_cc_include_DeepPotPT.h

Inheritance Relationships

Base Type

- `public deepmd::DeepPotBase` (Class DeepPotBase)

Class Documentation

class **DeepPotPT** : public deepmd::DeepPotBase

PyTorch implementation for Deep Potential.

Public Functions

DeepPotPT()

DP constructor without initialization.

~DeepPotPT()

DeepPotPT(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")

DP constructor with initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **file_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

virtual void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")

Initialize the DP.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **file_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

inline virtual double **cutoff**() const

Get the cutoff radius.

Returns
The cutoff radius.

inline virtual int **numb_types**() const

Get the number of types.

Returns
The number of types.

inline virtual int **numb_types_spin**() const

Get the number of types with spin.

Returns
The number of types with spin.

inline virtual int **dim_fparam**() const

Get the dimension of the frame parameter.

Returns
The dimension of the frame parameter.

inline virtual int **dim_aparam()** const

Get the dimension of the atomic parameter.

Returns
The dimension of the atomic parameter.

virtual void **get_type_map**(std::string &type_map)

Get the type map (element name of the atom types) of this model.

Parameters
type_map – [out] The type map of this model.

inline virtual bool **is_aparam_nall()** const

Get whether the atom dimension of aparam is nall instead of fparam.

Parameters
aparam_nall – [out] whether the atom dimension of aparam is nall instead of fparam.

virtual void **computew**(std::vector<double> &ener, std::vector<double> &force,
std::vector<double> &virial, std::vector<double> &atom_energy,
std::vector<double> &atom_virial, const std::vector<double> &coord, const
std::vector<int> &atype, const std::vector<double> &box, const
std::vector<double> &fparam = std::vector<double>(), const
std::vector<double> &aparam = std::vector<double>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

Note: The double precision interface is used by i-PI, GROMACS, ABACUS, and CP2k.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

virtual void **computew**(std::vector<double> &ener, std::vector<float> &force, std::vector<float> &virial, std::vector<float> &atom_energy, std::vector<float> &atom_virial, const std::vector<float> &coord, const std::vector<int> &atype, const std::vector<float> &box, const std::vector<float> &fparam = std::vector<float>(), const std::vector<float> &aparam = std::vector<float>())

virtual void **computew**(std::vector<double> &ener, std::vector<double> &force, std::vector<double> &virial, std::vector<double> &atom_energy, std::vector<double> &atom_virial, const std::vector<double> &coord, const std::vector<int> &atype, const std::vector<double> &box, const int nghost, const InputNlist &inlist, const int &ago, const std::vector<double> &fparam = std::vector<double>(), const std::vector<double> &aparam = std::vector<double>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

Note: The double precision interface is used by LAMMPS and AMBER.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **lmp_list** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

virtual void **computew**(std::vector<double> &ener, std::vector<float> &force, std::vector<float> &virial, std::vector<float> &atom_energy, std::vector<float> &atom_virial, const std::vector<float> &coord, const std::vector<int> &atype, const std::vector<float> &box, const int nghost, const InputNlist &inlist, const int &ago, const std::vector<float> &fparam = std::vector<float>(), const std::vector<float> &aparam = std::vector<float>())

virtual void **computew_mixed_type**(std::vector<double> &ener, std::vector<double> &force, std::vector<double> &virial, std::vector<double> &atom_energy, std::vector<double> &atom_virial, const int &nframes, const std::vector<double> &coord, const std::vector<int> &atype, const std::vector<double> &box, const std::vector<double> &fparam = std::vector<double>(), const std::vector<double> &aparam = std::vector<double>())

Evaluate the energy, force, and virial with the mixed type by using this DP.

Note: At this time, no external program uses this interface.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The array should be of size nframes x natoms.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

virtual void **computew_mixed_type**(std::vector<double> &ener, std::vector<float> &force, std::vector<float> &virial, std::vector<float> &atom_energy, std::vector<float> &atom_virial, const int &nframes, const std::vector<float> &coord, const std::vector<int> &atype, const std::vector<float> &box, const std::vector<float> &fparam = std::vector<float>(), const std::vector<float> &aparam = std::vector<float>())

Class DeepPotTF

- Defined in file_source_api_cc_include_DeepPotTF.h

Inheritance Relationships

Base Type

- `public deepmd::DeepPotBase` (Class DeepPotBase)

Class Documentation

class **DeepPotTF** : public deepmd::DeepPotBase

TensorFlow implementation for Deep Potential.

Public Functions

DeepPotTF()

DP constructor without initialization.

~DeepPotTF()

DeepPotTF(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")

DP constructor with initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **file_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

virtual void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")

Initialize the DP.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **file_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

inline virtual double **cutoff**() const

Get the cutoff radius.

Returns

The cutoff radius.

inline virtual int **numb_types()** const

Get the number of types.

Returns
The number of types.

inline virtual int **numb_types_spin()** const

Get the number of types with spin.

Returns
The number of types with spin.

inline virtual int **dim_fparam()** const

Get the dimension of the frame parameter.

Returns
The dimension of the frame parameter.

inline virtual int **dim_aparam()** const

Get the dimension of the atomic parameter.

Returns
The dimension of the atomic parameter.

virtual void **get_type_map**(std::string &type_map)

Get the type map (element name of the atom types) of this model.

Parameters
type_map – [out] The type map of this model.

inline virtual bool **is_aparam_nall()** const

Get whether the atom dimension of aparam is nall instead of fparam.

Parameters
aparam_nall – [out] whether the atom dimension of aparam is nall instead of fparam.

virtual void **computew**(std::vector<double> &ener, std::vector<double> &force,
std::vector<double> &virial, std::vector<double> &atom_energy,
std::vector<double> &atom_virial, const std::vector<double> &coord, const
std::vector<int> &atype, const std::vector<double> &box, const
std::vector<double> &fparam = std::vector<double>(), const
std::vector<double> &aparam = std::vector<double>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

Note: The double precision interface is used by i-PI, GROMACS, ABACUS, and CP2k.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.

- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

virtual void **computew**(std::vector<double> &ener, std::vector<float> &force, std::vector<float> &virial, std::vector<float> &atom_energy, std::vector<float> &atom_virial, const std::vector<float> &coord, const std::vector<int> &atype, const std::vector<float> &box, const std::vector<float> &fparam = std::vector<float>(), const std::vector<float> &aparam = std::vector<float>())

virtual void **computew**(std::vector<double> &ener, std::vector<double> &force, std::vector<double> &virial, std::vector<double> &atom_energy, std::vector<double> &atom_virial, const std::vector<double> &coord, const std::vector<int> &atype, const std::vector<double> &box, const int nghost, const InputNlist &inlist, const int &ago, const std::vector<double> &fparam = std::vector<double>(), const std::vector<double> &aparam = std::vector<double>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

Note: The double precision interface is used by LAMMPS and AMBER.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **lmp_list** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.

- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

virtual void **computew**(std::vector<double> &ener, std::vector<float> &force, std::vector<float> &virial, std::vector<float> &atom_energy, std::vector<float> &atom_virial, const std::vector<float> &coord, const std::vector<int> &atype, const std::vector<float> &box, const int nghost, const InputNlist &inlist, const int &ago, const std::vector<float> &fparam = std::vector<float>(), const std::vector<float> &aparam = std::vector<float>())

virtual void **computew_mixed_type**(std::vector<double> &ener, std::vector<double> &force, std::vector<double> &virial, std::vector<double> &atom_energy, std::vector<double> &atom_virial, const int &nframes, const std::vector<double> &coord, const std::vector<int> &atype, const std::vector<double> &box, const std::vector<double> &fparam = std::vector<double>(), const std::vector<double> &aparam = std::vector<double>())

Evaluate the energy, force, and virial with the mixed type by using this DP.

Note: At this time, no external program uses this interface.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The array should be of size nframes x natoms.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

virtual void **computew_mixed_type**(std::vector<double> &ener, std::vector<float> &force, std::vector<float> &virial, std::vector<float> &atom_energy, std::vector<float> &atom_virial, const int &nframes, const std::vector<float> &coord, const std::vector<int> &atype, const std::vector<float> &box, const std::vector<float> &fparam = std::vector<float>(), const std::vector<float> &aparam = std::vector<float>())

Class DeepTensor

- Defined in file_source_api_cc_include_DeepTensor.h

Class Documentation

class **DeepTensor**

Deep Tensor.

Public Functions

DeepTensor()

Deep Tensor constructor without initialization.

~DeepTensor()

DeepTensor(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Deep Tensor constructor with initialization..

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] Name scopes of operations.

void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Initialize the Deep Tensor.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] Name scopes of operations.

void **print_summary**(const std::string &pre) const

Print the DP summary to the screen.

Parameters

pre – [in] The prefix to each line.

template<typename **VALUETYPE**>
void **compute**(std::vector<VALUETYPE> &value, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box)

Evaluate the value by using this model.

Parameters

- **value** – [out] The value to evalute, usually would be the atomic tensor.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.

- **box** – [in] The cell of the region. The array should be of size 9.

template<typename **VALUETYPE**>
void **compute**(std::vector<VALUETYPE> &value, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &inlist)

Evaluate the value by using this model.

Parameters

- **value** – [out] The value to evalute, usually would be the atomic tensor.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.

template<typename **VALUETYPE**>
void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box)

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global_tensor** – [out] The global tensor to evalute.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.

template<typename **VALUETYPE**>
void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &inlist)

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global_tensor** – [out] The global tensor to evalute.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.

- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.

template<typename **VALUETYPE**>
void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_tensor, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box)

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global_tensor** – [out] The global tensor to evalute.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **atom_tensor** – [out] The atomic tensor value of the model, size natoms x odim.
- **atom_virial** – [out] The component-wise atomic virial of the global tensor, size odim x natoms x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.

template<typename **VALUETYPE**>
void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_tensor, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &inlist)

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global_tensor** – [out] The global tensor to evalute.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **atom_tensor** – [out] The atomic tensor value of the model, size natoms x odim.
- **atom_virial** – [out] The component-wise atomic virial of the global tensor, size odim x natoms x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.

double **cutoff**() const

Get the cutoff radius.

Returns
The cutoff radius.

int **numb_types**() const

Get the number of types.

Returns
The number of types.

int **output_dim**() const

Get the output dimension.

Returns
The output dimension.

const std::vector<int> &**sel_types**() const

Get the list of sel types.

Returns
The list of sel types.

void **get_type_map**(std::string &type_map)

Get the type map (element name of the atom types) of this model.

Parameters
type_map – [out] The type map of this model.

Class DeepTensorBase

- Defined in file_source_api_cc_include_DeepTensor.h

Inheritance Relationships

Derived Type

- `public deepmd::DeepTensorTF` (Class DeepTensorTF)

Class Documentation

class **DeepTensorBase**

Deep Tensor.

Subclassed by deepmd::DeepTensorTF

Unnamed Group

virtual void **computew**(std::vector<double> &global_tensor, std::vector<double> &force, std::vector<double> &virial, std::vector<double> &atom_tensor, std::vector<double> &atom_virial, const std::vector<double> &coord, const std::vector<int> &atype, const std::vector<double> &box, const bool request_deriv) = 0

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global_tensor** – [out] The global tensor to evalute.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **atom_tensor** – [out] The atomic tensor value of the model, size natoms x odim.
- **atom_virial** – [out] The component-wise atomic virial of the global tensor, size odim x natoms x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **request_deriv** – [in] Whether to request the derivative of the global tensor, including force and virial.

virtual void **computew**(std::vector<float> &global_tensor, std::vector<float> &force, std::vector<float> &virial, std::vector<float> &atom_tensor, std::vector<float> &atom_virial, const std::vector<float> &coord, const std::vector<int> &atype, const std::vector<float> &box, const bool request_deriv) = 0

Unnamed Group

virtual void **computew**(std::vector<double> &global_tensor, std::vector<double> &force, std::vector<double> &virial, std::vector<double> &atom_tensor, std::vector<double> &atom_virial, const std::vector<double> &coord, const std::vector<int> &atype, const std::vector<double> &box, const int nghost, const InputNlist &inlist, const bool request_deriv) = 0

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global_tensor** – [out] The global tensor to evalute.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **atom_tensor** – [out] The atomic tensor value of the model, size natoms x odim.
- **atom_virial** – [out] The component-wise atomic virial of the global tensor, size odim x natoms x 9.

- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.
- **request_deriv** – [in] Whether to request the derivative of the global tensor, including force and virial.

virtual void **computew**(std::vector<float> &global_tensor, std::vector<float> &force, std::vector<float> &virial, std::vector<float> &atom_tensor, std::vector<float> &atom_virial, const std::vector<float> &coord, const std::vector<int> &atype, const std::vector<float> &box, const int nghost, const InputNlist &inlist, const bool request_deriv) = 0

Public Functions

inline **DeepTensorBase**()

Deep Tensor constructor without initialization.

inline virtual **~DeepTensorBase**()

DeepTensorBase(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Deep Tensor constructor with initialization..

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] Name scopes of operations.

virtual void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "") = 0

Initialize the Deep Tensor.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] Name scopes of operations.

virtual double **cutoff**() const = 0

Get the cutoff radius.

Returns
The cutoff radius.

virtual int **numb_types**() const = 0

Get the number of types.

Returns
The number of types.

virtual int **output_dim**() const = 0

Get the output dimension.

Returns
The output dimension.

virtual const std::vector<int> &**sel_types**() const = 0

Get the list of sel types.

Returns
The list of sel types.

virtual void **get_type_map**(std::string &type_map) = 0

Get the type map (element name of the atom types) of this model.

Parameters
type_map – [out] The type map of this model.

Class DeepTensorTF

- Defined in file_source_api_cc_include_DeepTensorTF.h

Inheritance Relationships

Base Type

- `public deepmd::DeepTensorBase` (Class DeepTensorBase)

Class Documentation

class **DeepTensorTF** : public deepmd::DeepTensorBase

Deep Tensor.

Unnamed Group

virtual void **computew**(std::vector<double> &global_tensor, std::vector<double> &force, std::vector<double> &virial, std::vector<double> &atom_tensor, std::vector<double> &atom_virial, const std::vector<double> &coord, const std::vector<int> &atype, const std::vector<double> &box, const bool request_deriv)

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global_tensor** – [out] The global tensor to evalute.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **atom_tensor** – [out] The atomic tensor value of the model, size natoms x odim.

- **atom_virial** – [out] The component-wise atomic virial of the global tensor, size odim x natoms x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **request_deriv** – [in] Whether to request the derivative of the global tensor, including force and virial.

virtual void **computew**(std::vector<float> &global_tensor, std::vector<float> &force, std::vector<float> &virial, std::vector<float> &atom_tensor, std::vector<float> &atom_virial, const std::vector<float> &coord, const std::vector<int> &atype, const std::vector<float> &box, const bool request_deriv)

Unnamed Group

virtual void **computew**(std::vector<double> &global_tensor, std::vector<double> &force, std::vector<double> &virial, std::vector<double> &atom_tensor, std::vector<double> &atom_virial, const std::vector<double> &coord, const std::vector<int> &atype, const std::vector<double> &box, const int nghost, const InputNlist &inlist, const bool request_deriv)

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global_tensor** – [out] The global tensor to evalute.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **atom_tensor** – [out] The atomic tensor value of the model, size natoms x odim.
- **atom_virial** – [out] The component-wise atomic virial of the global tensor, size odim x natoms x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.
- **request_deriv** – [in] Whether to request the derivative of the global tensor, including force and virial.

virtual void **computew**(std::vector<float> &global_tensor, std::vector<float> &force, std::vector<float> &virial, std::vector<float> &atom_tensor, std::vector<float> &atom_virial, const std::vector<float> &coord, const std::vector<int> &atype, const std::vector<float> &box, const int nghost, const InputNlist &inlist, const bool request_deriv)

Public Functions

`DeepTensorTF()`

Deep Tensor constructor without initialization.

`~DeepTensorTF()`

`DeepTensorTF(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")`

Deep Tensor constructor with initialization..

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] Name scopes of operations.

virtual void `init(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")`

Initialize the Deep Tensor.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] Name scopes of operations.

inline virtual double `cutoff()` const

Get the cutoff radius.

Returns
The cutoff radius.

inline virtual int `numb_types()` const

Get the number of types.

Returns
The number of types.

inline virtual int `output_dim()` const

Get the output dimension.

Returns
The output dimension.

inline virtual const std::vector<int> &`sel_types()` const

Get the list of sel types.

Returns
The list of sel types.

virtual void `get_type_map(std::string &type_map)`

Get the type map (element name of the atom types) of this model.

Parameters
type_map – [out] The type map of this model.

Class DipoleChargeModifier

- Defined in file_source_api_cc_include_DataModifier.h

Class Documentation

class **DipoleChargeModifier**

Dipole charge modifier.

Public Functions

DipoleChargeModifier()

Dipole charge modifier without initialization.

DipoleChargeModifier(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Dipole charge modifier without initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] The name scope.

~DipoleChargeModifier()

void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Initialize the dipole charge modifier.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] The name scope.

void **print_summary**(const std::string &pre) const

Print the DP summary to the screen.

Parameters

pre – [in] The prefix to each line.

template<typename **VALUETYPE**>
void **compute**(std::vector<VALUETYPE> &dfcorr_, std::vector<VALUETYPE> &dvcorr_, const std::vector<VALUETYPE> &dcoord_, const std::vector<int> &datype_, const std::vector<VALUETYPE> &dbox, const std::vector<std::pair<int, int>> &pairs, const std::vector<VALUETYPE> &delef_, const int nghost, const InputNlist &lmp_list)

Evaluate the force and virial correction by using this dipole charge modifier.

Parameters

- **dfcorr_** – [out] The force correction on each atom.
- **dvcorr_** – [out] The virial correction.

- **dcoord_** – [in] The coordinates of atoms. The array should be of size nall x 3.
- **datype_** – [in] The atom types. The list should contain nall ints.
- **dbox** – [in] The cell of the region. The array should be of size 9.
- **pairs** – [in] The pairs of atoms. The list should contain npairs pairs of ints.
- **delef_** – [in] The electric field on each atom. The array should be of size nloc x 3.
- **nghost** – [in] The number of ghost atoms.
- **lmp_list** – [in] The neighbor list.

double **cutoff**() const

Get cutoff radius.

Returns
double cutoff radius.

int **numb_types**() const

Get the number of atom types.

Returns
int number of atom types.

const std::vector<int> &**sel_types**() const

Get the list of sel types.

Returns
The list of sel types.

Class DipoleChargeModifierBase

- Defined in file_source_api_cc_include_DataModifier.h

Inheritance Relationships

Derived Type

- `public deepmd::DipoleChargeModifierTF` (Class DipoleChargeModifierTF)

Class Documentation

class **DipoleChargeModifierBase**

Dipole charge modifier. (Base class)

Subclassed by deepmd::DipoleChargeModifierTF

Unnamed Group

virtual void **computew**(std::vector<double> &dfcorr_, std::vector<double> &dvcorr_, const std::vector<double> &dcoord_, const std::vector<int> &datype_, const std::vector<double> &dbox, const std::vector<std::pair<int, int>> &pairs, const std::vector<double> &delef_, const int nghost, const InputNlist &lmp_list) = 0

Evaluate the force and virial correction by using this dipole charge modifier.

Parameters

- **dfcorr_** – [out] The force correction on each atom.
- **dvcorr_** – [out] The virial correction.
- **dcoord_** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **datype_** – [in] The atom types. The list should contain natoms ints.
- **dbox** – [in] The cell of the region. The array should be of size 9.
- **pairs** – [in] The pairs of atoms. The list should contain npairs pairs of ints.
- **delef_** – [in] The electric field on each atom. The array should be of size natoms x 3.
- **nghost** – [in] The number of ghost atoms.
- **lmp_list** – [in] The neighbor list.

virtual void **computew**(std::vector<float> &dfcorr_, std::vector<float> &dvcorr_, const std::vector<float> &dcoord_, const std::vector<int> &datype_, const std::vector<float> &dbox, const std::vector<std::pair<int, int>> &pairs, const std::vector<float> &delef_, const int nghost, const InputNlist &lmp_list) = 0

Public Functions

inline **DipoleChargeModifierBase**()

Dipole charge modifier without initialization.

DipoleChargeModifierBase(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Dipole charge modifier without initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] The name scope.

inline virtual **~DipoleChargeModifierBase**()

virtual void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "") = 0

Initialize the dipole charge modifier.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] The name scope.

virtual double **cutoff**() const = 0

Get cutoff radius.

Returns
double cutoff radius.

virtual int **numb_types**() const = 0

Get the number of atom types.

Returns
int number of atom types.

virtual const std::vector<int> &**sel_types**() const = 0

Get the list of sel types.

Returns
The list of sel types.

Class DipoleChargeModifierTF

- Defined in file_source_api_cc_include_DataModifierTF.h

Inheritance Relationships

Base Type

- `public deepmd::DipoleChargeModifierBase` (Class DipoleChargeModifierBase)

Class Documentation

class **DipoleChargeModifierTF** : public deepmd::DipoleChargeModifierBase

Dipole charge modifier.

Public Functions

DipoleChargeModifierTF()

Dipole charge modifier without initialization.

DipoleChargeModifierTF(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Dipole charge modifier without initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.

- **name_scope** – [in] The name scope.

~DipoleChargeModifierTF()

virtual void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Initialize the dipole charge modifier.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] The name scope.

inline virtual double **cutoff**() const

Get cutoff radius.

Returns
double cutoff radius.

inline virtual int **numb_types**() const

Get the number of atom types.

Returns
int number of atom types.

inline virtual const std::vector<int> &**sel_types**() const

Get the list of sel types.

Returns
The list of sel types.

virtual void **computew**(std::vector<double> &dfcorr_, std::vector<double> &dvcorr_, const std::vector<double> &dcoord_, const std::vector<int> &datype_, const std::vector<double> &dbox, const std::vector<std::pair<int, int>> &pairs, const std::vector<double> &delef_, const int nghost, const InputNlist &lmp_list)

Evaluate the force and virial correction by using this dipole charge modifier.

Parameters

- **dfcorr_** – [out] The force correction on each atom.
- **dvcorr_** – [out] The virial correction.
- **dcoord_** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **datype_** – [in] The atom types. The list should contain natoms ints.
- **dbox** – [in] The cell of the region. The array should be of size 9.
- **pairs** – [in] The pairs of atoms. The list should contain npairs pairs of ints.
- **delef_** – [in] The electric field on each atom. The array should be of size natoms x 3.
- **nghost** – [in] The number of ghost atoms.
- **lmp_list** – [in] The neighbor list.

virtual void **computew**(std::vector<float> &dfcorr_, std::vector<float> &dvcorr_, const std::vector<float> &dcoord_, const std::vector<int> &datype_, const std::vector<float> &dbox, const std::vector<std::pair<int, int>> &pairs, const std::vector<float> &delef_, const int nghost, const InputNlist &lmp_list)

22.3.3 Enums

Enum DPBackend

- Defined in file_source_api_cc_include_common.h

Enum Documentation

enum **deepmd::DPBackend**

Values:

enumerator **TensorFlow**

enumerator **PyTorch**

enumerator **Paddle**

enumerator **Unknown**

22.3.4 Functions

Function deepmd::check_status

- Defined in file_source_api_cc_include_commonTF.h

Function Documentation

void **deepmd::check_status**(const tensorflow::Status &status)

Check TensorFlow status. Exit if not OK.

Parameters

status – [in] TensorFlow status.

Function deepmd::convert_pbtxt_to_pb

- Defined in file_source_api_cc_include_common.h

Function Documentation

void `deepmd::convert_pbtxt_to_pb`(std::string fn_pb_txt, std::string fn_pb)

Convert pbtxt to pb.

Parameters

- **`fn_pb_txt`** – [in] Filename of the pb txt file.
- **`fn_pb`** – [in] Filename of the pb file.

Function deepmd::get_env_nthreads

- Defined in file_source_api_cc_include_common.h

Function Documentation

void `deepmd::get_env_nthreads`(int &num_intra_nthreads, int &num_inter_nthreads)

Get the number of threads from the environment variable.

A warning will be thrown if environmental variables are not set.

Parameters

- **`num_intra_nthreads`** – [out] The number of intra threads. Read from DP_INTRA_OP_PARALLELISM_THREADS.
- **`num_inter_nthreads`** – [out] The number of inter threads. Read from DP_INTER_OP_PARALLELISM_THREADS.

Function deepmd::load_op_library

- Defined in file_source_api_cc_include_common.h

Function Documentation

void `deepmd::load_op_library`()

Dynamically load OP library. This should be called before loading graphs.

Function deepmd::model_compatable

- Defined in file_source_api_cc_include_common.h

Function Documentation

bool `deepmd::model_compatable`(std::string &model_version)

Check if the model version is supported.

Parameters

`model_version` – [in] The model version.

Returns

Whether the model is supported (true or false).

Function deepmd::name_prefix

- Defined in file_source_api_cc_include_common.h

Function Documentation

std::string `deepmd::name_prefix`(const std::string &name_scope)

Function deepmd::print_summary

- Defined in file_source_api_cc_include_common.h

Function Documentation

void `deepmd::print_summary`(const std::string &pre)

Print the summary of DeePMD-kit, including the version and the build information.

Parameters

`pre` – [in] The prefix to each line.

Function deepmd::read_file_to_string

- Defined in file_source_api_cc_include_common.h

Function Documentation

void `deepmd::read_file_to_string`(std::string model, std::string &file_content)

Read model file to a string.

Parameters

- `model` – [in] Path to the model.
- `file_content` – [out] Content of the model file.

Template Function deepmd::select_by_type

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **VALUETYPE**>
void **deepmd::select_by_type**(std::vector<int> &fwd_map, std::vector<int> &bkw_map, int &nghost_real, const std::vector<VALUETYPE> &dcoord_, const std::vector<int> &datype_, const int &nghost, const std::vector<int> &sel_type_)

Get forward and backward map of selected atoms by atom types.

Parameters

- **fwd_map** – [out] The forward map with size natoms.
- **bkw_map** – [out] The backward map with size nreal.
- **nghost_real** – [out] The number of selected ghost atoms.
- **dcoord_** – [in] The coordinates of all atoms. Reserved for compatibility.
- **datype_** – [in] The atom types of all atoms.
- **nghost** – [in] The number of ghost atoms.
- **sel_type_** – [in] The selected atom types.

Template Function deepmd::select_map(std::vector<VT>&, const std::vector<VT>&, const std::vector<int>&, const int&, const int&, const int&, const int&)

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **VT**>
void **deepmd::select_map**(std::vector<VT> &out, const std::vector<VT> &in, const std::vector<int> &fwd_map, const int &stride, const int &nframes = 1, const int &nall1 = 0, const int &nall2 = 0)

Apply the given map to a vector.

Parameters

- **out** – [out] The output vector.
- **in** – [in] The input vector.
- **fwd_map** – [in] The map.
- **stride** – [in] The stride of the input vector.
- **nframes** – [in] The number of frames.
- **nall1** – [in] The number of atoms in the input vector.
- **nall2** – [in] The number of atoms in the output vector.

Template Function deepmd::select_map(typename std::vector<VT>::iterator, const typename std::vector<VT>::const_iterator, const std::vector<int>&, const int&, const int&, const int&, const int&)

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **VT**>
void **deepmd::select_map**(typename std::vector<VT>::iterator out, const typename std::vector<VT>::const_iterator in, const std::vector<int> &fwd_map, const int &stride, const int &nframes = 1, const int &nall1 = 0, const int &nall2 = 0)

Apply the given map to a vector.

Parameters

- **out** – [out] The output vector.
- **in** – [in] The input vector.
- **fwd_map** – [in] The map.
- **stride** – [in] The stride of the input vector.
- **nframes** – [in] The number of frames.
- **nall1** – [in] The number of atoms in the input vector.
- **nall2** – [in] The number of atoms in the output vector.

Template Function deepmd::select_map_inv(std::vector<VT>&, const std::vector<VT>&, const std::vector<int>&, const int&)

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **VT**>
void **deepmd::select_map_inv**(std::vector<VT> &out, const std::vector<VT> &in, const std::vector<int> &fwd_map, const int &stride)

Template Function deepmd::select_map_inv(typename std::vector<VT>::iterator, const typename std::vector<VT>::const_iterator, const std::vector<int>&, const int&)

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **VT**>
void **deepmd::select_map_inv**(typename std::vector<VT>::iterator out, const typename std::vector<VT>::const_iterator in, const std::vector<int> &fwd_map, const int &stride)

Template Function deepmd::select_real_atoms

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **VALUETYPE**>
void **deepmd::select_real_atoms**(std::vector<int> &fwd_map, std::vector<int> &bkw_map, int &nghost_real, const std::vector<VALUETYPE> &dcoord_, const std::vector<int> &datype_, const int &nghost, const int &ntypes)

Template Function deepmd::select_real_atoms_coord

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **VALUETYPE**>
void **deepmd::select_real_atoms_coord**(std::vector<VALUETYPE> &dcoord, std::vector<int> &datype, std::vector<VALUETYPE> &aparam, int &nghost_real, std::vector<int> &fwd_map, std::vector<int> &bkw_map, int &nall_real, int &nloc_real, const std::vector<VALUETYPE> &dcoord_, const std::vector<int> &datype_, const std::vector<VALUETYPE> &aparam_, const int &nghost, const int &ntypes, const int &nframes, const int &daparam, const int &nall, const bool aparam_nall = false)

Function deepmd::session_get_dtype

- Defined in file_source_api_cc_include_commonTF.h

Function Documentation

int **deepmd::session_get_dtype**(tensorflow::Session *session, const std::string name, const std::string scope = "")

Get the type of a tensor.

Parameters

- **session** – [in] TensorFlow session.

- **name** – [in] The name of the tensor.
- **scope** – [in] The scope of the tensor.

Returns
The type of the tensor as int.

Template Function deepmd::session_get_scalar

- Defined in file_source_api_cc_include_commonTF.h

Function Documentation

template<typename **VT**>
VT **deepmd::session_get_scalar**(tensorflow::Session *session, const std::string name, const std::string scope = "")

Get the value of a tensor.

Parameters

- **session** – [in] TensorFlow session.
- **name** – [in] The name of the tensor.
- **scope** – [in] The scope of the tensor.

Returns
The value of the tensor.

Template Function deepmd::session_get_vector

- Defined in file_source_api_cc_include_commonTF.h

Function Documentation

template<typename **VT**>
void **deepmd::session_get_vector**(std::vector<VT> &o_vec, tensorflow::Session *session, const std::string name_, const std::string scope = "")

Get the vector of a tensor.

Parameters

- **o_vec** – [out] The output vector.
- **session** – [in] TensorFlow session.
- **name** – [in] The name of the tensor.
- **scope** – [in] The scope of the tensor.

Template Function deepmd::session_input_tensors(std::vector<std::pair<std::string, tensorflow::Tensor»&, const std::vector<VALUETYPE>&, const int&, const std::vector<int>&, const std::vector<VALUETYPE>&, const double&, const std::vector<VALUETYPE>&, const std::vector<VALUETYPE>&, const deepmd::AtomMap&, const std::string, const bool)

- Defined in file_source_api_cc_include_commonTF.h

Function Documentation

template<typename **MODELTYPE**, typename **VALUETYPE**>
int **deepmd::session_input_tensors**(std::vector<std::pair<std::string, tensorflow::Tensor>> &input_tensors, const std::vector<VALUETYPE> &dcoord_, const int &ntypes, const std::vector<int> &datype_, const std::vector<VALUETYPE> &dbox, const double &cell_size, const std::vector<VALUETYPE> &fparam_, const std::vector<VALUETYPE> &aparam_, const deepmd::AtomMap &atommap, const std::string scope = "", const bool aparam_nall = false)

Get input tensors.

Parameters

- **input_tensors** – [out] Input tensors.
- **dcoord_** – [in] Coordinates of atoms.
- **ntypes** – [in] Number of atom types.
- **datype_** – [in] Atom types.
- **dbox** – [in] Box matrix.
- **cell_size** – [in] Cell size.
- **fparam_** – [in] Frame parameters.
- **aparam_** – [in] Atom parameters.
- **atommap** – [in] Atom map.
- **scope** – [in] The scope of the tensors.
- **aparam_nall** – [in] Whether the atomic dimesion of atomic parameters is nall.

Template Function deepmd::session_input_tensors(std::vector<std::pair<std::string, tensorflow::Tensor»&, const std::vector<VALUETYPE>&, const int&, const std::vector<int>&, const std::vector<VALUETYPE>&, InputNlist&, const std::vector<VALUETYPE>&, const std::vector<VALUETYPE>&, const deepmd::AtomMap&, const int, const int, const std::string, const bool)

- Defined in file_source_api_cc_include_commonTF.h

Function Documentation

template<typename **MODELTYPE**, typename **VALUETYPE**>
int **deepmd::session_input_tensors**(std::vector<std::pair<std::string, tensorflow::Tensor>> &input_tensors, const std::vector<VALUETYPE> &dcoord_, const int &ntypes, const std::vector<int> &datype_, const std::vector<VALUETYPE> &dbox, InputNlist &dlist, const std::vector<VALUETYPE> &fparam_, const std::vector<VALUETYPE> &aparam_, const deepmd::AtomMap &atommap, const int nghost, const int ago, const std::string scope = "", const bool aparam_nall = false)

Get input tensors.

Parameters

- **input_tensors** – [out] Input tensors.
- **dcoord_** – [in] Coordinates of atoms.
- **ntypes** – [in] Number of atom types.
- **datype_** – [in] Atom types.
- **dlist** – [in] Neighbor list.
- **fparam_** – [in] Frame parameters.
- **aparam_** – [in] Atom parameters.
- **atommap** – [in] Atom map.
- **nghost** – [in] Number of ghost atoms.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **scope** – [in] The scope of the tensors.
- **aparam_nall** – [in] Whether the atomic dimesion of atomic parameters is nall.

Template Function deepmd::session_input_tensors_mixed_type

- Defined in file_source_api_cc_include_commonTF.h

Function Documentation

template<typename **MODELTYPE**, typename **VALUETYPE**>
int **deepmd::session_input_tensors_mixed_type**(std::vector<std::pair<std::string, tensorflow::Tensor>> &input_tensors, const int &nframes, const std::vector<VALUETYPE> &dcoord_, const int &ntypes, const std::vector<int> &datype_, const std::vector<VALUETYPE> &dbox, const double &cell_size, const std::vector<VALUETYPE> &fparam_, const std::vector<VALUETYPE> &aparam_, const deepmd::AtomMap &atommap, const std::string scope = "", const bool aparam_nall = false)

Get input tensors for mixed type.

Parameters

- **input_tensors** – [out] Input tensors.
- **nframes** – [in] Number of frames.
- **dcoord_** – [in] Coordinates of atoms.
- **ntypes** – [in] Number of atom types.
- **datype_** – [in] Atom types.
- **dlist** – [in] Neighbor list.
- **fparam_** – [in] Frame parameters.
- **aparam_** – [in] Atom parameters.
- **atommap** – [in] Atom map.
- **nghost** – [in] Number of ghost atoms.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **scope** – [in] The scope of the tensors.
- **aparam_nall** – [in] Whether the atomic dimesion of atomic parameters is nall.

22.3.5 Typedefs

Typedef deepmd::ENERGYTYPE

- Defined in file_source_api_cc_include_common.h

Typedef Documentation

typedef double `deepmd::ENERGYTYPE`

Typedef deepmd::STRINGTYPE

- Defined in file_source_api_cc_include_tf_private.h

Typedef Documentation

typedef std::string `deepmd::STRINGTYPE`

CHAPTER

TWENTYTHREE

C API

23.1 Class Hierarchy

- Namespace deepmd
  - Namespace deepmd::hpp
    - Struct deepmd_exception
    - Struct InputNlist
    - Class DeepPot
    - Class DeepPotModelDevi
    - Class DeepTensor
    - Class DipoleChargeModifier
- Struct DP_DeepPot
- Struct DP_DeepPotModelDevi
- Struct DP_DeepTensor
- Struct DP_DipoleChargeModifier
- Struct DP_Nlist

23.2 File Hierarchy

- dir_source
  - dir_source_api_c
    - dir_source_api_c_include
      - file_source_api_c_include_c_api.h
      - file_source_api_c_include_c_api_internal.h
      - file_source_api_c_include_deepmd.hpp

23.3 Full API

23.3.1 Namespaces

Namespace deepmd

Contents

- Namespaces

Namespaces

- Namespace deepmd::hpp

Namespace deepmd::hpp

Contents

- Classes
- Functions

Classes

- Struct deepmd_exception
- Struct InputNlist
- Class DeepPot
- Class DeepPotModelDevi
- Class DeepTensor
- Class DipoleChargeModifier

Functions

- Function deepmd::hpp::convert_nlist
- Function deepmd::hpp::convert_pbtxt_to_pb
- Function deepmd::hpp::read_file_to_string
- Template Function deepmd::hpp::select_by_type
- Template Function deepmd::hpp::select_map

23.3.2 Classes and Structs

Struct deepmd_exception

- Defined in file_source_api_c_include_deepmd.hpp

Inheritance Relationships

Base Type

- `public runtime_error`

Struct Documentation

struct **deepmd_exception** : public runtime_error

General DeePMD-kit exception. Throw if anything doesn't work.

Public Functions

inline **deepmd_exception**()

inline **deepmd_exception**(const std::string &msg)

Struct InputNlist

- Defined in file_source_api_c_include_deepmd.hpp

Struct Documentation

struct **InputNlist**

Neighbor list.

Public Functions

inline **InputNlist**()

inline **InputNlist**(int inum_, int *ilist_, int *numneigh_, int **firstneigh_)

inline **InputNlist**(int inum_, int *ilist_, int *numneigh_, int **firstneigh_, int nswap, int *sendnum, int *recvnum, int *firstrecv, int **sendlist, int *sendproc, int *recvproc, void *world)

inline ~**InputNlist**()

Public Members

DP_Nlist *nl

C API neighbor list.

int inum

Number of core region atoms.

int *ilist

Array stores the core region atom's index.

int *numneigh

Array stores the core region atom's neighbor atom number.

int **firstneigh

Array stores the core region atom's neighbor index.

Struct DP_DeepPot

- Defined in file_source_api_c_include_c_api_internal.h

Struct Documentation

struct DP_DeepPot

Public Functions

DP_DeepPot()

DP_DeepPot(deepmd::DeepPot &dp)

Public Members

deepmd::DeepPot dp

std::string exception

int dfparam

int daparam

bool aparam_nall

Struct DP_DeepPotModelDevi

- Defined in file_source_api_c_include_c_api_internal.h

Struct Documentation

struct `DP_DeepPotModelDevi`

Public Functions

`DP_DeepPotModelDevi()`

`DP_DeepPotModelDevi`(deepmd::DeepPotModelDevi &dp)

Public Members

deepmd::DeepPotModelDevi `dp`

std::string `exception`

int `dfparam`

int `daparam`

bool `aparam_nall`

Struct DP_DeepTensor

- Defined in file_source_api_c_include_c_api_internal.h

Struct Documentation

struct `DP_DeepTensor`

Public Functions

`DP_DeepTensor()`

`DP_DeepTensor`(deepmd::DeepTensor &dt)

Public Members

deepmd::DeepTensor **dt**

std::string **exception**

Struct DP_DipoleChargeModifier

- Defined in file_source_api_c_include_c_api_internal.h

Struct Documentation

struct **DP_DipoleChargeModifier**

Public Functions

DP_DipoleChargeModifier()

DP_DipoleChargeModifier(deepmd::DipoleChargeModifier &dcm)

Public Members

deepmd::DipoleChargeModifier **dcm**

std::string **exception**

Struct DP_Nlist

- Defined in file_source_api_c_include_c_api_internal.h

Struct Documentation

struct **DP_Nlist**

Public Functions

`DP_Nlist()`

`DP_Nlist`(deepmd::InputNlist &nl)

Public Members

deepmd::InputNlist `nl`

std::string `exception`

Class DeepPot

- Defined in file_source_api_c_include_deepmd.hpp

Class Documentation

class `DeepPot`

Deep Potential.

Public Functions

inline `DeepPot()`

DP constructor without initialization.

inline `~DeepPot()`

inline `DeepPot`(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")

DP constructor with initialization.

Parameters

- `model` – [in] The name of the frozen model file.
- `gpu_rank` – [in] The GPU rank.
- `file_content` – [in] The content of the frozen model file.

inline void `init`(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")

Initialize the DP.

Parameters

- `model` – [in] The name of the frozen model file.
- `gpu_rank` – [in] The GPU rank.
- `file_content` – [in] The content of the frozen model file.

template<typename `VALUETYPE`, typename `ENERGYVTYPE`>

inline void **compute**(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using this DP.

> Warning: Natoms should not be zero when computing multiple frames.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**, typename **ENERGYVTYPE**>
inline void **compute**(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

> Warning: Natoms should not be zero when computing multiple frames.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.

- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**, typename **ENERGYVTYPE**>
inline void **compute**(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using this DP with the neighbor list.

> Warning: Natoms should not be zero when computing multiple frames.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**, typename **ENERGYVTYPE**>

inline void **compute**(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP with the neighbor list.

> Warning: Natoms should not be zero when computing multiple frames.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**, typename **ENERGYVTYPE**>
inline void **compute_mixed_type**(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const int &nframes, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using this DP with the mixed type.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**, typename **ENERGYVTYPE**>
inline void **compute_mixed_type**(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy, std::vector<VALUETYPE> &atom_virial, const int &nframes, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP with the mixed type.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.

- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

inline double **cutoff**() const

Get the cutoff radius.

Returns
The cutoff radius.

inline int **numb_types**() const

Get the number of types.

Returns
The number of types.

inline int **numb_types_spin**() const

Get the number of types with spin.

Returns
The number of types with spin.

inline void **get_type_map**(std::string &type_map)

Get the type map (element name of the atom types) of this model.

Parameters
type_map – [out] The type map of this model.

inline void **print_summary**(const std::string &pre) const

Print the summary of DeePMD-kit, including the version and the build information.

Parameters
pre – [in] The prefix to each line.

inline int **dim_fparam**() const

Get the dimension of the frame parameter.

Returns
The dimension of the frame parameter.

inline int **dim_aparam**() const

Get the dimension of the atomic parameter.

Returns
The dimension of the atomic parameter.

Class DeepPotModelDevi

- Defined in file_source_api_c_include_deepmd.hpp

Class Documentation

class **DeepPotModelDevi**

Deep Potential model deviation.

Public Functions

inline **DeepPotModelDevi**()

DP model deviation constructor without initialization.

inline **~DeepPotModelDevi**()

inline **DeepPotModelDevi**(const std::vector<std::string> &models)

DP model deviation constructor with initialization.

Parameters

models – [in] The names of the frozen model file.

inline void **init**(const std::vector<std::string> &models, const int &gpu_rank = 0, const std::vector<std::string> &file_content = std::vector<std::string>())

Initialize the DP model deviation.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank.
- **file_content** – [in] The content of the frozen model file.

template<typename **VALUETYPE**>
inline void **compute**(std::vector<double> &ener, std::vector<std::vector<VALUETYPE>> &force, std::vector<std::vector<VALUETYPE>> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using this DP model deviation.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.

- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**>
inline void **compute**(std::vector<double> &ener, std::vector<std::vector<VALUETYPE>> &force, std::vector<std::vector<VALUETYPE>> &virial, std::vector<std::vector<VALUETYPE>> &atom_energy, std::vector<std::vector<VALUETYPE>> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP model deviation.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**>
inline void **compute**(std::vector<double> &ener, std::vector<std::vector<VALUETYPE>> &force, std::vector<std::vector<VALUETYPE>> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using this DP model deviation.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.

- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**>
inline void **compute**(std::vector<double> &ener, std::vector<std::vector<VALUETYPE>> &force, std::vector<std::vector<VALUETYPE>> &virial, std::vector<std::vector<VALUETYPE>> &atom_energy, std::vector<std::vector<VALUETYPE>> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP model deviation.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.

- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

inline double **cutoff**() const

Get the cutoff radius.

Returns
The cutoff radius.

inline int **numb_types**() const

Get the number of types.

Returns
The number of types.

inline int **numb_types_spin**() const

Get the number of types with spin.

Returns
The number of types with spin.

inline int **dim_fparam**() const

Get the dimension of the frame parameter.

Returns
The dimension of the frame parameter.

inline int **dim_aparam**() const

Get the dimension of the atomic parameter.

Returns
The dimension of the atomic parameter.

template<typename **VALUETYPE**>
inline void **compute_avg**(std::vector<VALUETYPE> &avg, const std::vector<std::vector<VALUETYPE>> &xx)

Compute the average of vectors.

Parameters

- **avg** – [out] The average of vectors.
- **xx** – [in] The vectors of all models.

template<typename **VALUETYPE**>
inline void **compute_std**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const std::vector<std::vector<VALUETYPE>> &xx, const int &stride)

Compute the standard deviation of vectors.

Parameters

- **std** – [out] The standard deviation of vectors.
- **avg** – [in] The average of vectors.
- **xx** – [in] The vectors of all models.
- **stride** – [in] The stride to compute the deviation.

template<typename **VALUETYPE**>

inline void **compute_relative_std**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const VALUETYPE eps, const int &stride)

Compute the relative standard deviation of vectors.

Parameters

- **std** – [out] The standard deviation of vectors.
- **avg** – [in] The average of vectors.
- **eps** – [in] The level parameter for computing the deviation.
- **stride** – [in] The stride to compute the deviation.

template<typename **VALUETYPE**>
inline void **compute_std_f**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const std::vector<std::vector<VALUETYPE>> &xx)

Compute the standard deviation of forces.

Parameters

- **std** – [out] The standard deviation of forces.
- **avg** – [in] The average of forces.
- **xx** – [in] The vectors of all forces.

template<typename **VALUETYPE**>
inline void **compute_relative_std_f**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const VALUETYPE eps)

Compute the relative standard deviation of forces.

Parameters

- **std** – [out] The relative standard deviation of forces.
- **avg** – [in] The relative average of forces.
- **eps** – [in] The level parameter for computing the deviation.

Class DeepTensor

- Defined in file_source_api_c_include_deepmd.hpp

Class Documentation

class **DeepTensor**

Deep Tensor.

Public Functions

inline **DeepTensor**()

Deep Tensor constructor without initialization.

inline ~**DeepTensor**()

inline **DeepTensor**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

DeepTensor constructor with initialization.

Parameters

model – [in] The name of the frozen model file.

inline void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Initialize the DeepTensor.

Parameters

model – [in] The name of the frozen model file.

template<typename **VALUETYPE**>
inline void **compute**(std::vector<VALUETYPE> &tensor, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box)

Evaluate the tensor, force and virial by using this Deep Tensor.

Parameters

- **tensor** – [out] The atomic tensor.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).

template<typename **VALUETYPE**>
inline void **compute**(std::vector<VALUETYPE> &tensor, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list)

Evaluate the tensor, force and virial by using this Deep Tensor with the neighbor list.

Parameters

- **tensor** – [out] The tensor.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.

template<typename **VALUETYPE**>

inline void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box)

Evaluate the global tensor, force and virial by using this Deep Tensor.

Parameters

- **global_tensor** – [out] The global tensor.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).

template<typename **VALUETYPE**>
inline void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_tensor, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box)

Evaluate the global tensor, force, virial, atomic tensor, and atomic virial by using this Deep Tensor.

Parameters

- **global_tensor** – [out] The global tensor.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_tensor** – [out] The atomic tensor.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).

template<typename **VALUETYPE**>
inline void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list)

Evaluate the global tensor, force and virial by using this Deep Tensor with the neighbor list.

Parameters

- **global_tensor** – [out] The global tensor.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.

- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.

template<typename **VALUETYPE**>
inline void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_tensor, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list)

Evaluate the global tensor, force, virial, atomic tensor, and atomic virial by using this Deep Tensor with the neighbor list.

Parameters

- **global_tensor** – [out] The global tensor.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_tensor** – [out] The atomic tensor.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.

inline double **cutoff**() const

Get the cutoff radius.

Returns
The cutoff radius.

inline int **numb_types**() const

Get the number of types.

Returns
The number of types.

inline int **output_dim**() const

Get the output dimension.

Returns
The output dimension.

inline std::vector<int> **sel_types**() const

inline void **print_summary**(const std::string &pre) const

Print the summary of DeePMD-kit, including the version and the build information.

Parameters

pre – [in] The prefix to each line.

inline void **get_type_map**(std::string &type_map)

Get the type map (element name of the atom types) of this model.

Parameters

type_map – [out] The type map of this model.

Class DipoleChargeModifier

- Defined in file_source_api_c_include_deepmd.hpp

Class Documentation

class **DipoleChargeModifier**

Public Functions

inline **DipoleChargeModifier**()

DipoleChargeModifier constructor without initialization.

inline **~DipoleChargeModifier**()

inline **DipoleChargeModifier**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

DipoleChargeModifier constructor with initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The rank of the GPU to be used.
- **name_scope** – [in] The name scope of the model.

inline void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Initialize the DipoleChargeModifier.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The rank of the GPU to be used.
- **name_scope** – [in] The name scope of the model.

template<typename **VALUETYPE**>

inline void **compute**(std::vector<VALUETYPE> &dfcorr_, std::vector<VALUETYPE> &dvcorr_, const std::vector<VALUETYPE> &dcoord_, const std::vector<int> &datype_, const std::vector<VALUETYPE> &dbox, const std::vector<std::pair<int, int>> &pairs, const std::vector<VALUETYPE> &delef_, const int nghost, const InputNlist &lmp_list)

Evaluate the force and virial correction by using this dipole charge modifier.

Parameters

- **dfcorr_** – [out] The force correction on each atom.
- **dvcorr_** – [out] The virial correction.
- **dcoord_** – [in] The coordinates of atoms. The array should be of size nall x 3.
- **datype_** – [in] The atom types. The list should contain nall ints.
- **dbox** – [in] The cell of the region. The array should be of size 9.
- **pairs** – [in] The pairs of atoms. The list should contain npairs pairs of ints.
- **delef_** – [in] The electric field on each atom. The array should be of size nghost x 3.
- **nghost** – [in] The number of ghost atoms.
- **lmp_list** – [in] The neighbor list.

inline double **cutoff**() const

Get the cutoff radius.

Returns
The cutoff radius.

inline int **numb_types**() const

Get the number of types.

Returns
The number of types.

inline std::vector<int> **sel_types**() const

inline void **print_summary**(const std::string &pre) const

Print the summary of DeePMD-kit, including the version and the build information.

Parameters
pre – [in] The prefix to each line.

23.3.3 Functions

Template Function _DP_DeepPotCompute

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **FPTYPE**>
inline void **_DP_DeepPotCompute**(DP_DeepPot *dp, const int nframes, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, const FPTYPE *fparam, const FPTYPE *aparam, double *energy, FPTYPE *force, FPTYPE *virial, FPTYPE *atomic_energy, FPTYPE *atomic_virial)

Specialized Template Function _DP_DeepPotCompute< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepPotCompute**<double>(DP_DeepPot *dp, const int nframes, const int natom, const double *coord, const int *atype, const double *cell, const double *fparam, const double *aparam, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Specialized Template Function _DP_DeepPotCompute< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepPotCompute**<float>(DP_DeepPot *dp, const int nframes, const int natom, const float *coord, const int *atype, const float *cell, const float *fparam, const float *aparam, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Template Function _DP_DeepPotComputeMixedType

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **FPTYPE**>
inline void **_DP_DeepPotComputeMixedType**(DP_DeepPot *dp, const int nframes, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, const FPTYPE *fparam, const FPTYPE *aparam, double *energy, FPTYPE *force, FPTYPE *virial, FPTYPE *atomic_energy, FPTYPE *atomic_virial)

Specialized Template Function _DP_DeepPotComputeMixedType< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepPotComputeMixedType**<double>(DP_DeepPot *dp, const int nframes, const int natom, const double *coord, const int *atype, const double *cell, const double *fparam, const double *aparam, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Specialized Template Function _DP_DeepPotComputeMixedType< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepPotComputeMixedType**<float>(DP_DeepPot *dp, const int nframes, const int natom, const float *coord, const int *atype, const float *cell, const float *fparam, const float *aparam, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Template Function _DP_DeepPotComputeNList

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **FPTYPE**>
inline void **_DP_DeepPotComputeNList**(DP_DeepPot *dp, const int nframes, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, const int nghost, const DP_Nlist *nlist, const int ago, const FPTYPE *fparam, const FPTYPE *aparam, double *energy, FPTYPE *force, FPTYPE *virial, FPTYPE *atomic_energy, FPTYPE *atomic_virial)

Specialized Template Function _DP_DeepPotComputeNList< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepPotComputeNList**<double>(DP_DeepPot *dp, const int nframes, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, const int ago, const double *fparam, const double *aparam, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Specialized Template Function _DP_DeepPotComputeNList< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepPotComputeNList**<float>(DP_DeepPot *dp, const int nframes, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, const int ago, const float *fparam, const float *aparam, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Template Function _DP_DeepPotModelDeviCompute

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **FPTYPE**>
inline void **_DP_DeepPotModelDeviCompute**(DP_DeepPotModelDevi *dp, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, const FPTYPE *fparam, const FPTYPE *aparam, double *energy, FPTYPE *force, FPTYPE *virial, FPTYPE *atomic_energy, FPTYPE *atomic_virial)

Specialized Template Function _DP_DeepPotModelDeviCompute< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepPotModelDeviCompute**<double>(DP_DeepPotModelDevi *dp, const int natom, const double *coord, const int *atype, const double *cell, const double *fparam, const double *aparam, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Specialized Template Function _DP_DeepPotModelDeviCompute< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepPotModelDeviCompute**<float>(DP_DeepPotModelDevi *dp, const int natom, const float *coord, const int *atype, const float *cell, const float *fparam, const float *aparam, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Template Function _DP_DeepPotModelDeviComputeNList

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **FPTYPE**>
inline void **_DP_DeepPotModelDeviComputeNList**(DP_DeepPotModelDevi *dp, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, const int nghost, const DP_Nlist *nlist, const int ago, const FPTYPE *fparam, const FPTYPE *aparam, double *energy, FPTYPE *force, FPTYPE *virial, FPTYPE *atomic_energy, FPTYPE *atomic_virial)

Specialized Template Function _DP_DeepPotModelDeviComputeNList< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepPotModelDeviComputeNList**<double>(DP_DeepPotModelDevi *dp, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, const int ago, const double *fparam, const double *aparam, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Specialized Template Function _DP_DeepPotModelDeviComputeNList< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepPotModelDeviComputeNList**<float>(DP_DeepPotModelDevi *dp, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, const int ago, const float *fparam, const float *aparam, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Template Function _DP_DeepTensorCompute

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **FPTYPE**>
inline void **_DP_DeepTensorCompute**(DP_DeepTensor *dt, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, FPTYPE *global_tensor, FPTYPE *force, FPTYPE *virial, FPTYPE **atomic_energy, FPTYPE *atomic_virial, int *size_at)

Specialized Template Function _DP_DeepTensorCompute< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorCompute**<double>(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, double *global_tensor, double *force, double *virial, double **atomic_tensor, double *atomic_virial, int *size_at)

Specialized Template Function _DP_DeepTensorCompute< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorCompute**<float>(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, float *global_tensor, float *force, float *virial, float **atomic_tensor, float *atomic_virial, int *size_at)

Template Function _DP_DeepTensorComputeNList

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **FPTYPE**>
inline void **_DP_DeepTensorComputeNList**(DP_DeepTensor *dt, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, const int nghost, const DP_Nlist *nlist, FPTYPE *global_tensor, FPTYPE *force, FPTYPE *virial, FPTYPE **atomic_energy, FPTYPE *atomic_virial, int *size_at)

Specialized Template Function _DP_DeepTensorComputeNList< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorComputeNList**<double>(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, double *global_tensor, double *force, double *virial, double **atomic_tensor, double *atomic_virial, int *size_at)

Specialized Template Function _DP_DeepTensorComputeNList< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorComputeNList**<float>(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, float *global_tensor, float *force, float *virial, float **atomic_tensor, float *atomic_virial, int *size_at)

Template Function _DP_DeepTensorComputeTensor

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **FPTYPE**>
inline void **_DP_DeepTensorComputeTensor**(DP_DeepTensor *dt, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, FPTYPE **tensor, int *size)

Specialized Template Function _DP_DeepTensorComputeTensor< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorComputeTensor**<double>(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, double **tensor, int *size)

Specialized Template Function _DP_DeepTensorComputeTensor< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorComputeTensor**<float>(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, float **tensor, int *size)

Template Function _DP_DeepTensorComputeTensorNList

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **FPTYPE**>
inline void **_DP_DeepTensorComputeTensorNList**(DP_DeepTensor *dt, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, const int nghost, const DP_Nlist *nlist, FPTYPE **tensor, int *size)

Specialized Template Function _DP_DeepTensorComputeTensorNList< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorComputeTensorNList**<double>(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, double **tensor, int *size)

Specialized Template Function _DP_DeepTensorComputeTensorNList< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorComputeTensorNList**<float>(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, float **tensor, int *size)

Template Function _DP_DipoleChargeModifierComputeNList

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **FPTYPE**>
inline void **_DP_DipoleChargeModifierComputeNList**(DP_DipoleChargeModifier *dcm, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, const int *pairs, const int npairs, const FPTYPE *delef_, const int nghost, const DP_Nlist *nlist, FPTYPE *dfcorr_, FPTYPE *dvcorr_)

Specialized Template Function _DP_DipoleChargeModifierComputeNList< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DipoleChargeModifierComputeNList**<double>(DP_DipoleChargeModifier *dcm, const int natom, const double *coord, const int *atype, const double *cell, const int *pairs, const int npairs, const double *delef_, const int nghost, const DP_Nlist *nlist, double *dfcorr_, double *dvcorr_)

Specialized Template Function _DP_DipoleChargeModifierComputeNList< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DipoleChargeModifierComputeNList**<float>(DP_DipoleChargeModifier *dcm, const int natom, const float *coord, const int *atype, const float *cell, const int *pairs, const int npairs, const float *delef_, const int nghost, const DP_Nlist *nlist, float *dfcorr_, float *dvcorr_)

Function _DP_Get_Energy_Pointer(std::vector<double>&, const int)

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

inline double ***_DP_Get_Energy_Pointer**(std::vector<double> &vec, const int nframes)

Function _DP_Get_Energy_Pointer(double&, const int)

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

inline double ***_DP_Get_Energy_Pointer**(double &vec, const int nframes)

Function deepmd::hpp::convert_nlist

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

inline void **deepmd::hpp::convert_nlist**(InputNlist &to_nlist, std::vector<std::vector<int>> &from_nlist)

Convert int vector to InputNlist.

Parameters

- **to_nlist** – [out] InputNlist.
- **from_nlist** – [in] 2D int vector. The first axis represents the centeral atoms and the second axis represents the neighbor atoms.

Function deepmd::hpp::convert_pbtxt_to_pb

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

inline void **deepmd::hpp::convert_pbtxt_to_pb**(std::string fn_pb_txt, std::string fn_pb)

Convert pbtxt to pb.

Parameters

- **fn_pb_txt** – [in] Filename of the pb txt file.
- **fn_pb** – [in] Filename of the pb file.

Function deepmd::hpp::read_file_to_string

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

inline void **deepmd::hpp::read_file_to_string**(std::string model, std::string &file_content)

Read model file to a string.

Parameters

- **model** – [in] Path to the model.
- **file_content** – [out] Content of the model file.

Template Function deepmd::hpp::select_by_type

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **VALUETYPE**>
void **deepmd::hpp::select_by_type**(std::vector<int> &fwd_map, std::vector<int> &bkw_map, int &nghost_real, const std::vector<VALUETYPE> &dcoord_, const std::vector<int> &datype_, const int &nghost, const std::vector<int> &sel_type_)

Get forward and backward map of selected atoms by atom types.

Parameters

- **fwd_map** – [out] The forward map with size natoms.
- **bkw_map** – [out] The backward map with size nreal.
- **nghost_real** – [out] The number of selected ghost atoms.
- **dcoord_** – [in] The coordinates of all atoms. Reserved for compatibility.
- **datype_** – [in] The atom types of all atoms.

- **nghost** – [in] The number of ghost atoms.
- **sel_type_** – [in] The selected atom types.

Template Function deepmd::hpp::select_map

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **VT**>
void **deepmd::hpp::select_map**(std::vector<VT> &out, const std::vector<VT> &in, const std::vector<int> &fwd_map, const int &stride)

Apply the given map to a vector. Assume nframes is 1.

Template Parameters
VT – The value type of the vector. Only support int.

Parameters

- **out** – [out] The output vector.
- **in** – [in] The input vector.
- **fwd_map** – [in] The map.
- **stride** – [in] The stride of the input vector.

Function DP_ConvertPbtxtToPb

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_ConvertPbtxtToPb**(const char *c_pbtxt, const char *c_pb)

Convert PBtxt to PB.

Parameters

- **c_pbtxt** – [in] The name of the PBtxt file.
- **c_pb** – [in] The name of the PB file.

Function DP_DeepPotCheckOK

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char *`DP_DeepPotCheckOK`(DP_DeepPot *dp)

Check if there is any exceptions throw.

Parameters
`dp` – The DP to use.

Returns
const char* error message.

Function DP_DeepPotCompute

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeepPotCompute`(DP_DeepPot *dp, const int natom, const double *coord, const int *atype, const double *cell, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP. (double version)

Attention
The number of frames is assumed to be 1.

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- `dp` – [in] The DP to use.
- `natoms` – [in] The number of atoms.
- `coord` – [in] The coordinates of atoms. The array should be of size natoms x 3.
- `atype` – [in] The atom types. The array should contain natoms ints.
- `box` – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- `energy` – [out] Output energy.
- `force` – [out] Output force. The array should be of size natoms x 3.
- `virial` – [out] Output virial. The array should be of size 9.
- `atomic_energy` – [out] Output atomic energy. The array should be of size natoms.
- `atomic_virial` – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotCompute2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotCompute2**(DP_DeepPot *dp, const int nframes, const int natom, const double *coord, const int *atype, const double *cell, const double *fparam, const double *aparam, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP. (double version)

Version
2

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputef

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeepPotComputef`(DP_DeepPot *dp, const int natom, const float *coord, const int *atype, const float *cell, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP. (float version)

Attention
The number of frames is assumed to be 1.

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputef2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotComputef2**(DP_DeepPot *dp, const int nframes, const int natom, const float *coord, const int *atype, const float *cell, const float *fparam, const float *aparam, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP. (float version)

Version
2

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputeMixedType

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotComputeMixedType**(DP_DeepPot *dp, const int nframes, const int natoms, const double *coord, const int *atype, const double *cell, const double *fparam, const double *aparam, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP with the mixed type. (double version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain nframes x natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputeMixedTypef

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotComputeMixedTypef**(DP_DeepPot *dp, const int nframes, const int natoms, const float *coord, const int *atype, const float *cell, const float *fparam, const float *aparam, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP with the mixed type. (float version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain nframes x natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputeNList

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotComputeNList**(DP_DeepPot *dp, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, const int ago, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP with the neighbor list. (double version)

Attention

The number of frames is assumed to be 1.

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.

- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputeNList2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotComputeNList2**(DP_DeepPot *dp, const int nframes, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, const int ago, const double *fparam, const double *aparam, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP with the neighbor list. (double version)

Version
2

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.

- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputeNListf

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotComputeNListf**(DP_DeepPot *dp, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, const int ago, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP with the neighbor list. (float version)

Attention
The number of frames is assumed to be 1.

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.

- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputeNListf2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotComputeNListf2**(DP_DeepPot *dp, const int nframes, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, const int ago, const float *fparam, const float *aparam, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP with the neighbor list. (float version)

Version
2

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotGetCutoff

- Defined in file_source_api_c_include_c_api.h

Function Documentation

double `DP_DeepPotGetCutoff`(DP_DeepPot *dp)

Get the type map of a DP.

Parameters
: `dp` – [in] The DP to use.

Returns
: The cutoff radius.

Function DP_DeepPotGetDimAParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int `DP_DeepPotGetDimAParam`(DP_DeepPot *dp)

Get the dimension of atomic parameters of a DP.

Parameters
: `dp` – [in] The DP to use.

Returns
: The dimension of atomic parameters of the DP.

Function DP_DeepPotGetDimFParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int `DP_DeepPotGetDimFParam`(DP_DeepPot *dp)

Get the dimension of frame parameters of a DP.

Parameters
: `dp` – [in] The DP to use.

Returns
: The dimension of frame parameters of the DP.

Function DP_DeepPotGetNumbTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int `DP_DeepPotGetNumbTypes`(DP_DeepPot *dp)

Get the number of types of a DP.

Parameters
: `dp` – [in] The DP to use.

Returns
: The number of types of the DP.

Function DP_DeepPotGetNumbTypesSpin

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int `DP_DeepPotGetNumbTypesSpin`(DP_DeepPot *dp)

Get the number of types with spin of a DP.

Parameters
: `dp` – [in] The DP to use.

Returns
: The number of types with spin of the DP.

Function DP_DeepPotGetTypeMap

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char *`DP_DeepPotGetTypeMap`(DP_DeepPot *dp)

Get the type map of a DP.

Parameters
: `dp` – [in] The DP to use.

Returns
: The type map of the DP.

Function DP_DeepPotIsAParamNAll

- Defined in file_source_api_c_include_c_api.h

Function Documentation

bool `DP_DeepPotIsAParamNAll`(DP_DeepPot *dp)

Check whether the atomic dimension of atomic parameters is nall instead of nloc.

Parameters
: `dp` – [in] The DP to use.

Returns
: true the atomic dimension of atomic parameters is nall

Returns
: false the atomic dimension of atomic parameters is nloc

Function DP_DeepPotModelDeviCheckOK

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char *`DP_DeepPotModelDeviCheckOK`(DP_DeepPotModelDevi *dp)

Check if there is any exceptions throw.

Parameters
: `dp` – The DP model deviation to use.

Returns
: const char* error message.

Function DP_DeepPotModelDeviCompute

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeepPotModelDeviCompute`(DP_DeepPotModelDevi *dp, const int natom, const double *coord, const int *atype, const double *cell, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (double version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters
: - `dp` – [in] The DP model deviation to use.

- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotModelDeviCompute2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotModelDeviCompute2**(DP_DeepPotModelDevi *dp, const int nframes, const int natoms, const double *coord, const int *atype, const double *cell, const double *fparam, const double *aparam, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (double version)

Version
2

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP model deviation to use.
- **nframes** – [in] The number of frames. Only support 1 for now.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x natoms x dim_aparam.

- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotModelDeviComputef

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotModelDeviComputef**(DP_DeepPotModelDevi *dp, const int natom, const float *coord, const int *atype, const float *cell, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (float version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP model deviation to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotModelDeviComputef2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotModelDeviComputef2**(DP_DeepPotModelDevi *dp, const int nframes, const int natoms, const float *coord, const int *atype, const float *cell, const float *fparam, const float *aparam, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (float version)

Version
2

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP model deviation to use.
- **nframes** – [in] The number of frames. Only support 1 for now.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x natoms x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotModelDeviComputeNList

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeepPotModelDeviComputeNList`(DP_DeepPotModelDevi *dp, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, const int ago, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (double version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP model deviation to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotModelDeviComputeNList2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotModelDeviComputeNList2**(DP_DeepPotModelDevi *dp, const int nframes, const int natoms, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, const int ago, const double *fparam, const double *aparam, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (double version)

Version
2

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP model deviation to use.
- **nframes** – [in] The number of frames. Only support 1 for now.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x natoms x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotModelDeviComputeNListf

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeepPotModelDeviComputeNListf`(DP_DeepPotModelDevi *dp, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, const int ago, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (float version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- `dp` – [in] The DP model deviation to use.
- `natoms` – [in] The number of atoms.
- `coord` – [in] The coordinates of atoms. The array should be of size natoms x 3.
- `atype` – [in] The atom types. The array should contain natoms ints.
- `box` – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- `nghost` – [in] The number of ghost atoms.
- `nlist` – [in] The neighbor list.
- `ago` – [in] Update the internal neighbour list if ago is 0.
- `energy` – [out] Output energy.
- `force` – [out] Output force. The array should be of size natoms x 3.
- `virial` – [out] Output virial. The array should be of size 9.
- `atomic_energy` – [out] Output atomic energy. The array should be of size natoms.
- `atomic_virial` – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotModelDeviComputeNListf2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotModelDeviComputeNListf2**(DP_DeepPotModelDevi *dp, const int nframes, const int natoms, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, const int ago, const float *fparam, const float *aparam, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (float version)

Version
2

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP model deviation to use.
- **nframes** – [in] The number of frames. Only support 1 for now.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x natoms x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotModelDeviGetCutoff

- Defined in file_source_api_c_include_c_api.h

Function Documentation

double `DP_DeepPotModelDeviGetCutoff`(DP_DeepPotModelDevi *dp)

Get the type map of a DP model deviation.

Parameters
: `dp` – [in] The DP model deviation to use.

Returns
: The cutoff radius.

Function DP_DeepPotModelDeviGetDimAParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int `DP_DeepPotModelDeviGetDimAParam`(DP_DeepPotModelDevi *dp)

Get the dimension of atomic parameters of a DP Model Deviation.

Parameters
: `dp` – [in] The DP Model Deviation to use.

Returns
: The dimension of atomic parameters of the DP Model Deviation.

Function DP_DeepPotModelDeviGetDimFParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int `DP_DeepPotModelDeviGetDimFParam`(DP_DeepPotModelDevi *dp)

Get the dimension of frame parameters of a DP Model Deviation.

Parameters
: `dp` – [in] The DP Model Deviation to use.

Returns
: The dimension of frame parameters of the DP Model Deviation.

Function DP_DeepPotModelDeviGetNumbTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int `DP_DeepPotModelDeviGetNumbTypes`(DP_DeepPotModelDevi *dp)

Get the number of types of a DP model deviation.

Parameters
: `dp` – [in] The DP model deviation to use.

Returns
: The number of types of the DP model deviation.

Function DP_DeepPotModelDeviGetNumbTypesSpin

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int `DP_DeepPotModelDeviGetNumbTypesSpin`(DP_DeepPotModelDevi *dp)

Get the number of types with spin of a DP model deviation.

Parameters
: `dp` – [in] The DP model deviation to use.

Returns
: The number of types with spin of the DP model deviation.

Function DP_DeepPotModelDeviIsAParamNAll

- Defined in file_source_api_c_include_c_api.h

Function Documentation

bool `DP_DeepPotModelDeviIsAParamNAll`(DP_DeepPotModelDevi *dp)

Check whether the atomic dimension of atomic parameters is nall instead of nloc.

Parameters
: `dp` – [in] The DP Model Deviation to use.

Returns
: true the atomic dimension of atomic parameters is nall

Returns
: false the atomic dimension of atomic parameters is nloc

Function DP_DeepTensorCheckOK

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char *`DP_DeepTensorCheckOK`(DP_DeepTensor *dt)

Check if there is any exceptions throw.

Parameters
: `dt` – The Deep Tensor to use.

Returns
: const char* error message.

Function DP_DeepTensorCompute

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeepTensorCompute`(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, double *global_tensor, double *force, double *virial, double **atomic_tensor, double *atomic_virial, int *size_at)

Evaluate the global tensor, force and virial by using a DP. (double version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- `dt` – [in] The Deep Tensor to use.
- `natoms` – [in] The number of atoms.
- `coord` – [in] The coordinates of atoms. The array should be of size natoms x 3.
- `atype` – [in] The atom types. The array should contain natoms ints.
- `box` – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- `global_tensor` – [out] Output global tensor.
- `force` – [out] Output force. The array should be of size natoms x 3.
- `virial` – [out] Output virial. The array should be of size 9.
- `atomic_tensor` – [out] Output atomic tensor. The array should be of size natoms.
- `atomic_virial` – [out] Output atomic virial. The array should be of size natoms x 9.
- `size_at` – [out] Output size of atomic tensor.

Function DP_DeepTensorComputef

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepTensorComputef**(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, float *global_tensor, float *force, float *virial, float **atomic_tensor, float *atomic_virial, int *size_at)

Evaluate the global tensor, force and virial by using a DP. (float version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **global_tensor** – [out] Output global tensor.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_tensor** – [out] Output atomic tensor. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.
- **size_at** – [out] Output size of atomic tensor.

Function DP_DeepTensorComputeNList

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepTensorComputeNList**(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, double *global_tensor, double *force, double *virial, double **atomic_tensor, double *atomic_virial, int *size_at)

Evaluate the global tensor, force and virial by using a DP with the neighbor list. (double version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **global_tensor** – [out] Output global tensor.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_tensor** – [out] Output atomic tensor. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.
- **size_at** – [out] Output size of atomic tensor.

Function DP_DeepTensorComputeNListf

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepTensorComputeNListf**(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, float *global_tensor, float *force, float *virial, float **atomic_tensor, float *atomic_virial, int *size_at)

Evaluate the global tensor, force and virial by using a DP with the neighbor list. (float version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.

- **global_tensor** – [out] Output global tensor.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_tensor** – [out] Output atomic tensor. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.
- **size_at** – [out] Output size of atomic tensor.

Function DP_DeepTensorComputeTensor

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepTensorComputeTensor**(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, double **tensor, int *size)

Evaluate the tensor by using a DP. (double version)

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **tensor** – [out] Output tensor.

Function DP_DeepTensorComputeTensorf

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepTensorComputeTensorf**(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, float **tensor, int *size)

Evaluate the tensor by using a DP. (float version)

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.

- **tensor** – [out] Output tensor.
- **size** – [out] Output size of the tensor.

Function DP_DeepTensorComputeTensorNList

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepTensorComputeTensorNList**(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, double **tensor, int *size)

Evaluate the tensor by using a DP with the neighbor list. (double version)

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **tensor** – [out] Output tensor.
- **size** – [out] Output size of the tensor.

Function DP_DeepTensorComputeTensorNListf

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepTensorComputeTensorNListf**(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, float **tensor, int *size)

Evaluate the tensor by using a DP with the neighbor list. (float version)

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.

- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **tensor** – [out] Output tensor.
- **size** – [out] Output size of the tensor.

Function DP_DeepTensorGetCutoff

- Defined in file_source_api_c_include_c_api.h

Function Documentation

double **DP_DeepTensorGetCutoff**(DP_DeepTensor *dt)

Get the type map of a Deep Tensor.

Parameters
dt – [in] The Deep Tensor to use.

Returns
The cutoff radius.

Function DP_DeepTensorGetNumbSelTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int **DP_DeepTensorGetNumbSelTypes**(DP_DeepTensor *dt)

Get the number of sel types of a Deep Tensor.

Parameters
dt – [in] The Deep Tensor to use.

Returns
The number of sel types

Function DP_DeepTensorGetNumbTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int `DP_DeepTensorGetNumbTypes`(DP_DeepTensor *dt)

Get the type map of a Deep Tensor.

Parameters
: `dt` – [in] The Deep Tensor to use.

Returns
: The number of types of the Deep Tensor.

Function DP_DeepTensorGetOutputDim

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int `DP_DeepTensorGetOutputDim`(DP_DeepTensor *dt)

Get the output dimension of a Deep Tensor.

Parameters
: `dt` – [in] The Deep Tensor to use.

Returns
: The output dimension of the Deep Tensor.

Function DP_DeepTensorGetSelTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int *`DP_DeepTensorGetSelTypes`(DP_DeepTensor *dt)

Get sel types of a Deep Tensor.

Parameters
: `dt` – [in] The Deep Tensor to use.

Returns
: The sel types

Function DP_DeepTensorGetTypeMap

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char *`DP_DeepTensorGetTypeMap`(DP_DeepTensor *dt)

Get the type map of a Deep Tensor.

Parameters
`dt` – [in] The Deep Tensor to use.

Returns
The type map of the Deep Tensor.

Function DP_DeleteChar

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeleteChar`(const char *c_str)

Destroy a char array.

Parameters
`c_str` – The char array.

Function DP_DeleteDeepPot

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeleteDeepPot`(DP_DeepPot *dp)

Delete a Deep Potential.

Parameters
`dp` – Deep Potential to delete.

Function DP_DeleteDeepPotModelDevi

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeleteDeepPotModelDevi`(DP_DeepPotModelDevi *dp)

Delete a Deep Potential Model Deviation.

Parameters
`dp` – Deep Potential to delete.

Function DP_DeleteDeepTensor

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeleteDeepTensor`(DP_DeepTensor *dt)

Delete a Deep Tensor.

Parameters

`dp` – Deep Tensor to delete.

Function DP_DeleteDipoleChargeModifier

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeleteDipoleChargeModifier`(DP_DipoleChargeModifier *dcm)

Delete a Dipole Charge Modifier.

Parameters

`dp` – Dipole Charge Modifier to delete.

Function DP_DeleteNlist

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeleteNlist`(DP_Nlist *nl)

Delete a neighbor list.

Parameters

`nl` – Neighbor list to delete.

Function DP_DipoleChargeModifierCheckOK

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char *`DP_DipoleChargeModifierCheckOK`(DP_DipoleChargeModifier *dcm)

Check if there is any exceptions throw.

Parameters
: `dcm` – The DipoleChargeModifier to use.

Returns
: const char* error message.

Function DP_DipoleChargeModifierComputeNList

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DipoleChargeModifierComputeNList`(DP_DipoleChargeModifier *dcm, const int natom, const double *coord, const int *atype, const double *cell, const int *pairs, const int npairs, const double *delef_, const int nghost, const DP_Nlist *nlist, double *dfcorr_, double *dvcorr_)

Evaluate the force and virial correction by using a dipole charge modifier with the neighbor list. (double version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- `dcm` – [in] The dipole charge modifier to use.
- `natoms` – [in] The number of atoms.
- `coord` – [in] The coordinates of atoms. The array should be of size nall x 3.
- `atype` – [in] The atom types. The array should contain nall ints.
- `cell` – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- `pairs` – [in] The pairs of atoms. The list should contain npairs pairs of ints.
- `npairs` – [in] The number of pairs.
- `delef_` – [in] The electric field on each atom. The array should be of size nframes x nloc x 3.
- `nghost` – [in] The number of ghost atoms.
- `nlist` – [in] The neighbor list.
- `dfcorr_` – [out] Output force correction. The array should be of size nall x 3.
- `dvcorr_` – [out] Output virial correction. The array should be of size 9.

Function DP_DipoleChargeModifierComputeNListf

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DipoleChargeModifierComputeNListf**(DP_DipoleChargeModifier *dcm, const int natom, const float *coord, const int *atype, const float *cell, const int *pairs, const int npairs, const float *delef_, const int nghost, const DP_Nlist *nlist, float *dfcorr_, float *dvcorr_)

Evaluate the force and virial correction by using a dipole charge modifier with the neighbor list. (float version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dcm** – [in] The dipole charge modifier to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **cell** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **pairs** – [in] The pairs of atoms. The list should contain npairs pairs of ints.
- **npairs** – [in] The number of pairs.
- **delef_** – [in] The electric field on each atom. The array should be of size nframes x natoms x 3.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **dfcorr_** – [out] Output force correction. The array should be of size natoms x 3.
- **dvcorr_** – [out] Output virial correction. The array should be of size 9.

Function DP_DipoleChargeModifierGetCutoff

- Defined in file_source_api_c_include_c_api.h

Function Documentation

double `DP_DipoleChargeModifierGetCutoff`(DP_DipoleChargeModifier *dt)

Get the type map of a DipoleChargeModifier.

Parameters
: `dcm` – [in] The DipoleChargeModifier to use.

Returns
: The cutoff radius.

Function DP_DipoleChargeModifierGetNumbSelTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int `DP_DipoleChargeModifierGetNumbSelTypes`(DP_DipoleChargeModifier *dt)

Get the number of sel types of a DipoleChargeModifier.

Parameters
: `dcm` – [in] The DipoleChargeModifier to use.

Returns
: The number of sel types

Function DP_DipoleChargeModifierGetNumbTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int `DP_DipoleChargeModifierGetNumbTypes`(DP_DipoleChargeModifier *dt)

Get the type map of a DipoleChargeModifier.

Parameters
: `dcm` – [in] The DipoleChargeModifier to use.

Returns
: The number of types of the DipoleChargeModifier.

Function DP_DipoleChargeModifierGetSelTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int *`DP_DipoleChargeModifierGetSelTypes`(DP_DipoleChargeModifier *dt)

Get sel types of a DipoleChargeModifier.

Parameters

`dcm` – [in] The DipoleChargeModifier to use.

Returns

The sel types

Function DP_NewDeepPot

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DeepPot *`DP_NewDeepPot`(const char *c_model)

DP constructor with initialization.

Parameters

`c_model` – [in] The name of the frozen model file.

Returns

A pointer to the deep potential.

Function DP_NewDeepPotModelDevi

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DeepPotModelDevi *`DP_NewDeepPotModelDevi`(const char **c_models, int n_models)

DP model deviation constructor with initialization.

Parameters

- `c_models` – [in] The array of the name of the frozen model file.
- `nmodels` – [in] The number of models.

Function DP_NewDeepPotModelDeviWithParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DeepPotModelDevi *DP_NewDeepPotModelDeviWithParam(const char **c_model, const int n_models, const int gpu_rank, const char **c_file_contents, const int n_file_contents, const int *size_file_contents)

DP model deviation constructor with initialization.

Parameters

- **c_models** – [in] The array of the name of the frozen model file.
- **nmodels** – [in] The number of models.
- **gpu_rank** – [in] The rank of the GPU.
- **c_file_contents** – [in] The contents of the model file.
- **n_file_contents** – [in] The number of the contents of the model file.
- **size_file_contents** – [in] The sizes of the contents of the model file.

Returns

DP_DeepPotModelDevi* A pointer to the deep potential model deviation.

Function DP_NewDeepPotWithParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DeepPot *DP_NewDeepPotWithParam(const char *c_model, const int gpu_rank, const char *c_file_content)

DP constructor with initialization.

Parameters

- **c_model** – The name of the frozen model file.
- **gpu_rank** – The rank of the GPU.
- **c_file_content** – Broken implementation. Use DP_NewDeepPotWithParam2 instead.

Returns

DP_DeepPot* A pointer to the deep potential.

Function DP_NewDeepPotWithParam2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DeepPot *DP_NewDeepPotWithParam2(const char *c_model, const int gpu_rank, const char *c_file_content, const int size_file_content)

DP constructor with initialization.

Version 2

Parameters

- **c_model** – The name of the frozen model file.
- **gpu_rank** – The rank of the GPU.
- **c_file_content** – The content of the model file.
- **size_file_content** – The size of the model file.

Returns

DP_DeepPot* A pointer to the deep potential.

Function DP_NewDeepTensor

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DeepTensor *DP_NewDeepTensor(const char *c_model)

Deep Tensor constructor with initialization.

Parameters

c_model – [in] The name of the frozen model file.

Returns

A pointer to the deep tensor.

Function DP_NewDeepTensorWithParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DeepTensor *DP_NewDeepTensorWithParam(const char *c_model, const int gpu_rank, const char *c_name_scope)

Deep Tensor constructor with initialization.

Parameters

- **c_model** – The name of the frozen model file.
- **gpu_rank** – The rank of the GPU.
- **c_name_scope** – The name scope.

Returns
DP_DeepTensor* A pointer to the deep tensor.

Function DP_NewDipoleChargeModifier

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DipoleChargeModifier *`DP_NewDipoleChargeModifier`(const char *c_model)

Dipole charge modifier constructor with initialization.

Parameters
`c_model` – [in] The name of the frozen model file.

Returns
A pointer to the dipole charge modifier.

Function DP_NewDipoleChargeModifierWithParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DipoleChargeModifier *`DP_NewDipoleChargeModifierWithParam`(const char *c_model, const int gpu_rank, const char *c_name_scope)

Dipole charge modifier constructor with initialization.

Parameters

- `c_model` – The name of the frozen model file.
- `gpu_rank` – The rank of the GPU.
- `c_name_scope` – The name scope.

Returns
DP_DipoleChargeModifier* A pointer to the dipole charge modifier.

Function DP_NewNlist

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_Nlist *`DP_NewNlist`(int inum_, int *ilist_, int *numneigh_, int **firstneigh_)

Create a new neighbor list.

Parameters

- `inum_` – [in] Number of core region atoms
- `Array` – [in] stores the core region atom's index
- `Array` – [in] stores the core region atom's neighbor atom number
- `Array` – [in] stores the core region atom's neighbor index

Returns

A pointer to the neighbor list.

Function DP_NewNlist_comm

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_Nlist *`DP_NewNlist_comm`(int inum_, int *ilist_, int *numneigh_, int **firstneigh_, int nswap, int *sendnum, int *recvnum, int *firstrecv, int **sendlist, int *sendproc, int *recvproc, void *world)

Function DP_NlistCheckOK

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char *`DP_NlistCheckOK`(DP_Nlist *dp)

Check if there is any exceptions throw.

Parameters

`dp` – The neighbor list to use.

Returns

const char* error message.

Function DP_PrintSummary

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_PrintSummary`(const char *c_pre)

Print the summary of DeePMD-kit, including the version and the build information.

Parameters

`c_pre` – [in] The prefix to each line.

Function DP_ReadFileToChar

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char *`DP_ReadFileToChar`(const char *c_model)

Read a file to a char array.

Parameters

`c_model` – [in] The name of the file.

Returns

const char* The char array.

Function DP_ReadFileToChar2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char *`DP_ReadFileToChar2`(const char *c_model, int *size)

Read a file to a char array. This version can handle string with '\0'.

Version

2

Parameters

- `c_model` – [in] The name of the file.
- `size` – [out] The size of the char array.

Returns

const char* The char array.

Function DP_SelectByType

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_SelectByType`(const int natoms, const int *atype, const int nghost, const int nsel_type, const int *sel_type, int *fwd_map, int *nreal, int *bkw_map, int *nghost_real)

Get forward and backward map of selected atoms by atom types.

Parameters

- **`natoms`** – [in] The number of atoms.
- **`atype`** – [in] The atom types of all atoms.
- **`nghost`** – [in] The number of ghost atoms.
- **`nsel_type`** – [in] The number of selected atom types.
- **`sel_type`** – [in] The selected atom types.
- **`fwd_map`** – [out] The forward map with size natoms.
- **`nreal`** – [out] The number of selected real atoms.
- **`bkw_map`** – [out] The backward map with size nreal.
- **`nghost_real`** – [out] The number of selected ghost atoms.

Function DP_SelectMapInt

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_SelectMapInt`(const int *in, const int *fwd_map, const int stride, const int nall1, const int nall2, int *out)

Apply the given map to a vector. Assume nframes is 1.

Parameters

- **`in`** – [in] The input vector.
- **`fwd_map`** – [in] The map.
- **`stride`** – [in] The stride of the input vector.
- **`nall1`** – [in] The number of atoms in the input vector.
- **`nall2`** – [out] The number of atoms in the output vector.
- **`out`** – [out] The output vector.

23.3.4 Defines

Define DP_C_API_VERSION

- Defined in file_source_api_c_include_c_api.h

Define Documentation

`DP_C_API_VERSION`

C API version. Bumped whenever the API is changed.

Since

API version 22

Define DP_CHECK_OK

- Defined in file_source_api_c_include_deepmd.hpp

Define Documentation

`DP_CHECK_OK`(check_func, dp)

Check if any exceptions throw in the C++ API. Throw if possible.

Define DP_NEW_OK

- Defined in file_source_api_c_include_c_api_internal.h

Define Documentation

`DP_NEW_OK`(dpcls, xx)

Define DP_REQUIRES_OK

- Defined in file_source_api_c_include_c_api_internal.h

Define Documentation

`DP_REQUIRES_OK`(dp, xx)

23.3.5 Typedefs

Typedef DP_DeepPot

- Defined in file_source_api_c_include_c_api.h

Typedef Documentation

typedef struct DP_DeepPot **DP_DeepPot**

The deep potential.

Typedef DP_DeepPotModelDevi

- Defined in file_source_api_c_include_c_api.h

Typedef Documentation

typedef struct DP_DeepPotModelDevi **DP_DeepPotModelDevi**

The deep potential model deviation.

Typedef DP_DeepTensor

- Defined in file_source_api_c_include_c_api.h

Typedef Documentation

typedef struct DP_DeepTensor **DP_DeepTensor**

The deep tensor.

Typedef DP_DipoleChargeModifier

- Defined in file_source_api_c_include_c_api.h

Typedef Documentation

typedef struct DP_DipoleChargeModifier **DP_DipoleChargeModifier**

The dipole charge modifier.

Typedef DP_Nlist

- Defined in file_source_api_c_include_c_api.h

Typedef Documentation

typedef struct DP_Nlist **DP_Nlist**

Neighbor list.

CHAPTER

TWENTYFOUR

CORE API

24.1 Class Hierarchy

- Namespace deepmd
  - Struct deepmd_exception
  - Struct deepmd_exception_oom
  - Template Struct EwaldParameters
  - Struct InputNlist
  - Template Struct Region
- Template Struct DescrptSeRGPUExecuteFunctor
- Template Struct GeluGPUExecuteFunctor
- Template Struct GeluGradGPUExecuteFunctor
- Template Struct GeluGradGradGPUExecuteFunctor
- Template Struct ProdForceSeAGPUExecuteFunctor
- Template Struct ProdForceSeRGPUExecuteFunctor
- Template Struct ProdVirialSeAGPUExecuteFunctor
- Template Struct ProdVirialSeRGPUExecuteFunctor
- Template Struct TabulateCheckerGPUExecuteFunctor
- Template Struct TabulateFusionGPUExecuteFunctor
- Template Struct TabulateFusionGradGPUExecuteFunctor
- Template Class SimulationRegion
- Union U_Flt64_Int64

24.2 File Hierarchy

- dir_source
  - dir_source_lib
    - dir_source_lib_include
      - file_source_lib_include_ComputeDescriptor.h
      - file_source_lib_include_coord.h
      - file_source_lib_include_device.h
      - file_source_lib_include_DeviceFunctor.h
      - file_source_lib_include_env_mat.h
      - file_source_lib_include_env_mat_nvnmd.h
      - file_source_lib_include_errors.h
      - file_source_lib_include_ewald.h
      - file_source_lib_include_fmt_nlist.h
      - file_source_lib_include_gelu.h
      - file_source_lib_include_gpu_cuda.h
      - file_source_lib_include_gpu_rocm.h
      - file_source_lib_include_map_aparam.h
      - file_source_lib_include_neighbor_list.h
      - file_source_lib_include_neighbor_stat.h
      - file_source_lib_include_pair_tab.h
      - file_source_lib_include_pairwise.h
      - file_source_lib_include_prod_env_mat.h
      - file_source_lib_include_prod_env_mat_nvnmd.h
      - file_source_lib_include_prod_force.h
      - file_source_lib_include_prod_force_grad.h
      - file_source_lib_include_prod_virial.h
      - file_source_lib_include_prod_virial_grad.h
      - file_source_lib_include_region.h
      - file_source_lib_include_SimulationRegion.h
      - file_source_lib_include_SimulationRegion_Impl.h
      - file_source_lib_include_soft_min_switch.h
      - file_source_lib_include_soft_min_switch_force.h
      - file_source_lib_include_soft_min_switch_force_grad.h
      - file_source_lib_include_soft_min_switch_virial.h
      - file_source_lib_include_soft_min_switch_virial_grad.h

- file_source_lib_include_switcher.h
- file_source_lib_include_tabulate.h
- file_source_lib_include_utilities.h

24.3 Full API

24.3.1 Namespaces

Namespace deepmd

Contents

- Classes
- Functions
- Variables

Classes

- Struct deepmd_exception
- Struct deepmd_exception_oom
- Template Struct EwaldParameters
- Struct InputNlist
- Template Struct Region

Functions

- Template Function deepmd::build_nlist_cpu
- Template Function deepmd::build_nlist_gpu
- Template Function deepmd::compute_cell_info
- Function deepmd::convert_nlist
- Function deepmd::convert_nlist_gpu_device
- Template Function deepmd::convert_to_inter_cpu
- Template Function deepmd::convert_to_inter_gpu
- Template Function deepmd::convert_to_phys_cpu
- Template Function deepmd::convert_to_phys_gpu
- Template Function deepmd::copy_coord_cpu
- Template Function deepmd::copy_coord_gpu
- Function deepmd::cos_switch(const double&, const double&, const double&)

- Function deepmd::cos_switch(double&, double&, const double&, const double&, const double&)
- Template Function deepmd::cprod
- Function deepmd::cum_sum
- Template Function deepmd::delete_device_memory
- Template Function deepmd::dot1
- Template Function deepmd::dot2
- Template Function deepmd::dot3
- Template Function deepmd::dot4
- Template Function deepmd::dotmv3
- Function deepmd::DPGetDeviceCount
- Function deepmd::dprc_pairwise_map_cpu
- Function deepmd::DPSetDevice
- Template Function deepmd::env_mat_a_cpu
- Template Function deepmd::env_mat_a_nvnmd_quantize_cpu
- Function deepmd::env_mat_nbor_update
- Template Function deepmd::env_mat_r_cpu
- Template Function deepmd::ewald_recp
- Function deepmd::filter_ftype_gpu
- Template Function deepmd::format_nbor_list_gpu
- Template Function deepmd::format_nlist_cpu
- Function deepmd::free_nlist_gpu_device
- Template Function deepmd::gelu_cpu
- Template Function deepmd::gelu_gpu
- Template Function deepmd::gelu_grad_cpu
- Template Function deepmd::gelu_grad_gpu
- Template Function deepmd::gelu_grad_grad_cpu
- Template Function deepmd::gelu_grad_grad_gpu
- Function deepmd::group_atoms_cpu
- Template Function deepmd::init_region_cpu
- Template Function deepmd::invsqrt
- Specialized Template Function deepmd::invsqrt< double >
- Specialized Template Function deepmd::invsqrt< float >
- Template Function deepmd::malloc_device_memory(FPTYPE *&, std::vector<FPTYPE>&)
- Template Function deepmd::malloc_device_memory(FPTYPE *&, const int)
- Template Function deepmd::malloc_device_memory(FPTYPE *&, const std::vector<FPTYPE>&)
- Template Function deepmd::malloc_device_memory_sync(FPTYPE *&, const FPTYPE *, const int)

- Template Function deepmd::malloc_device_memory_sync(FPTYPE *&, const std::vector<FPTYPE>&)
- Template Function deepmd::malloc_device_memory_sync(FPTYPE *&, std::vector<FPTYPE>&)
- Template Function deepmd::map_aparam_cpu
- Function deepmd::max_numneigh
- Template Function deepmd::memcpy_device_to_host(const FPTYPE *, FPTYPE *, const int)
- Template Function deepmd::memcpy_device_to_host(const FPTYPE *, std::vector<FPTYPE>&)
- Template Function deepmd::memcpy_host_to_device(FPTYPE *, std::vector<FPTYPE>&)
- Template Function deepmd::memcpy_host_to_device(FPTYPE *, const FPTYPE *, const int)
- Template Function deepmd::memcpy_host_to_device(FPTYPE *, const std::vector<FPTYPE>&)
- Template Function deepmd::memset_device_memory
- Template Function deepmd::neighbor_stat_gpu
- Template Function deepmd::normalize_coord_cpu
- Template Function deepmd::normalize_coord_gpu
- Template Function deepmd::pair_tab_cpu
- Template Function deepmd::prod_env_mat_a_cpu
- Template Function deepmd::prod_env_mat_a_gpu
- Template Function deepmd::prod_env_mat_a_nvnmd_quantize_cpu
- Template Function deepmd::prod_env_mat_r_cpu
- Template Function deepmd::prod_env_mat_r_gpu
- Template Function deepmd::prod_force_a_cpu(FPTYPE *, const FPTYPE *, const FPTYPE *, const int *, const int, const int, const int, const int)
- Template Function deepmd::prod_force_a_cpu(FPTYPE *, const FPTYPE *, const FPTYPE *, const int *, const int, const int, const int, const int, const int, const int)
- Template Function deepmd::prod_force_a_gpu
- Template Function deepmd::prod_force_grad_a_cpu
- Template Function deepmd::prod_force_grad_a_gpu
- Template Function deepmd::prod_force_grad_r_cpu
- Template Function deepmd::prod_force_grad_r_gpu
- Template Function deepmd::prod_force_r_cpu
- Template Function deepmd::prod_force_r_gpu
- Template Function deepmd::prod_virial_a_cpu
- Template Function deepmd::prod_virial_a_gpu
- Template Function deepmd::prod_virial_grad_a_cpu
- Template Function deepmd::prod_virial_grad_a_gpu
- Template Function deepmd::prod_virial_grad_r_cpu
- Template Function deepmd::prod_virial_grad_r_gpu

- Template Function deepmd::prod_virial_r_cpu
- Template Function deepmd::prod_virial_r_gpu
- Template Function deepmd::soft_min_switch_cpu
- Template Function deepmd::soft_min_switch_force_cpu
- Template Function deepmd::soft_min_switch_force_grad_cpu
- Template Function deepmd::soft_min_switch_virial_cpu
- Template Function deepmd::soft_min_switch_virial_grad_cpu
- Function deepmd::spline3_switch
- Template Function deepmd::spline5_switch
- Template Function deepmd::tabulate_fusion_se_a_cpu
- Template Function deepmd::tabulate_fusion_se_a_gpu
- Template Function deepmd::tabulate_fusion_se_a_grad_cpu
- Template Function deepmd::tabulate_fusion_se_a_grad_gpu
- Template Function deepmd::tabulate_fusion_se_a_grad_grad_cpu
- Template Function deepmd::tabulate_fusion_se_a_grad_grad_gpu
- Template Function deepmd::tabulate_fusion_se_r_cpu
- Template Function deepmd::tabulate_fusion_se_r_gpu
- Template Function deepmd::tabulate_fusion_se_r_grad_cpu
- Template Function deepmd::tabulate_fusion_se_r_grad_gpu
- Template Function deepmd::tabulate_fusion_se_r_grad_grad_cpu
- Template Function deepmd::tabulate_fusion_se_r_grad_grad_gpu
- Template Function deepmd::tabulate_fusion_se_t_cpu
- Template Function deepmd::tabulate_fusion_se_t_gpu
- Template Function deepmd::tabulate_fusion_se_t_grad_cpu
- Template Function deepmd::tabulate_fusion_se_t_grad_gpu
- Template Function deepmd::tabulate_fusion_se_t_grad_grad_cpu
- Template Function deepmd::tabulate_fusion_se_t_grad_grad_gpu
- Template Function deepmd::test_encoding_decoding_nbor_info_gpu
- Function deepmd::use_nei_info_cpu
- Function deepmd::use_nei_info_gpu
- Function deepmd::use_nlist_map
- Template Function deepmd::volume_cpu
- Template Function deepmd::volume_gpu

Variables

- Variable deepmd::ElectrostaticConvertion

24.3.2 Classes and Structs

Struct deepmd_exception

- Defined in file_source_lib_include_errors.h

Inheritance Relationships

Base Type

- `public runtime_error`

Derived Type

- `public deepmd::deepmd_exception_oom` (Struct deepmd_exception_oom)

Struct Documentation

struct **`deepmd_exception`** : public runtime_error

General DeePMD-kit exception. Throw if anything doesn't work.

Subclassed by deepmd::deepmd_exception_oom

Public Functions

inline **`deepmd_exception`**()

inline **`deepmd_exception`**(const std::string &msg)

Struct deepmd_exception_oom

- Defined in file_source_lib_include_errors.h

Inheritance Relationships

Base Type

- `public deepmd::deepmd_exception` (Struct deepmd_exception)

Struct Documentation

struct **deepmd_exception_oom** : public deepmd::deepmd_exception

Public Functions

inline **deepmd_exception_oom**()

inline **deepmd_exception_oom**(const std::string &msg)

Template Struct EwaldParameters

- Defined in file_source_lib_include_ewald.h

Struct Documentation

template<typename **VALUETYPE**>

struct **EwaldParameters**

Public Members

VALUETYPE **rcut** = 6.0

VALUETYPE **beta** = 2

VALUETYPE **spacing** = 4

Struct InputNlist

- Defined in file_source_lib_include_neighbor_list.h

Struct Documentation

struct **InputNlist**

Construct InputNlist with the input LAMMPS nbor list info.

Public Functions

inline **InputNlist**()

inline **InputNlist**(int inum_, int *ilist_, int *numneigh_, int **firstneigh_)

inline **InputNlist**(int inum_, int *ilist_, int *numneigh_, int **firstneigh_, int nswap, int *sendnum, int *recvnum, int *firstrecv, int **sendlist, int *sendproc, int *recvproc, void *world)

inline ~**InputNlist**()

Public Members

int **inum**

Number of core region atoms.

int ***ilist**

Array stores the core region atom's index.

int ***numneigh**

Array stores the core region atom's neighbor atom number.

int ****firstneigh**

Array stores the core region atom's neighbor index.

int **nswap**

of swaps to perform = sum of maxneed

int ***sendnum**

of atoms to send in each swap

int *recvnum

of atoms to recv in each swap

int *firstrecv
where to put 1st recv atom in each swap

int **sendlist
list of atoms to send in each swap

int *sendproc
proc to send to at each swap

int *recvproc
proc to recv from at each swap

void *world
MPI_comm data in lmp.

Template Struct Region

- Defined in file_source_lib_include_region.h

Struct Documentation

template<typename FPTYPE>

struct Region

Public Functions

Region()

Region(FPTYPE *extern_boxt, FPTYPE *extern_rec_boxt)

~Region()

Public Members

FPTYPE *boxt

FPTYPE *rec_boxt

Template Struct DescrptSeRGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename FPTYPE>

struct DescrptSeRGPUExecuteFunctor

Public Functions

void operator()(const FPTYPE *coord, const int *type, const int *ilist, const int *jrange, const int *jlist, int *array_int, unsigned long long *array_longlong, const FPTYPE *avg, const FPTYPE *std, FPTYPE *descript, FPTYPE *descript_deriv, FPTYPE *rij, int *nlist, const int nloc, const int nall, const int nnei, const int ndescrpt, const float rcut_r, const float rcut_r_smth, const std::vector<int> sec_a, const bool fill_nei_a, const int MAGIC_NUMBER)

Template Struct GeluGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename FPTYPE>

struct GeluGPUExecuteFunctor

Public Functions

void operator()(const FPTYPE *in, FPTYPE *out, const int size)

Template Struct GeluGradGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **GeluGradGPUExecuteFunctor**

Public Functions

void **operator()**(const FPTYPE *dy, const FPTYPE *in, FPTYPE *out, const int size)

Template Struct GeluGradGradGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **GeluGradGradGPUExecuteFunctor**

Public Functions

void **operator()**(const FPTYPE *dy, const FPTYPE *dy_, const FPTYPE *in, FPTYPE *out, const int size)

Template Struct ProdForceSeAGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **ProdForceSeAGPUExecuteFunctor**

Public Functions

void **operator()**(FPTYPE *force, const FPTYPE *net_derive, const FPTYPE *in_deriv, const int *nlist, const int nloc, const int nall, const int nnei, const int ndescrpt, const int n_a_sel, const int n_a_shift)

Template Struct ProdForceSeRGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **ProdForceSeRGPUExecuteFunctor**

Public Functions

void **operator()**(FPTYPE *force, const FPTYPE *net_derive, const FPTYPE *in_deriv, const int *nlist, const int nloc, const int nall, const int nnei, const int ndescrpt)

Template Struct ProdVirialSeAGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **ProdVirialSeAGPUExecuteFunctor**

Public Functions

void **operator()**(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE *in_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int nnei, const int ndescrpt, const int n_a_sel, const int n_a_shift)

Template Struct ProdVirialSeRGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **ProdVirialSeRGPUExecuteFunctor**

Public Functions

void **operator()**(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE *in_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int nnei, const int ndescrpt)

Template Struct TabulateCheckerGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **TabulateCheckerGPUExecuteFunctor**

Public Functions

void **operator()**(const FPTYPE *table_info, const FPTYPE *in, int *out, const int nloc, const int nnei)

Template Struct TabulateFusionGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **TabulateFusionGPUExecuteFunctor**

Public Functions

void **operator()**(const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *in, const FPTYPE *ff, const int nloc, const int nnei, const int last_layer_size, FPTYPE *out)

Template Struct TabulateFusionGradGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **TabulateFusionGradGPUExecuteFunctor**

Public Functions

void **operator()**(const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *in, const FPTYPE *ff, const FPTYPE *dy, const int nloc, const int nnei, const int last_layer_size, FPTYPE *dy_dx, FPTYPE *dy_df)

Template Class SimulationRegion

- Defined in file_source_lib_include_SimulationRegion.h

Class Documentation

template<typename **VALUETYPE**>

class **SimulationRegion**

Public Functions

inline void **reinitBox**(const double *boxv)

inline void **affineTransform**(const double *affine_map)

inline void **reinitOrigin**(const double *orig)

inline void **reinitOrigin**(const std::vector<double> &orig)

void **backup**()

void **recover**()

SimulationRegion()

~SimulationRegion()

inline double ***getBoxTensor**()

inline const double ***getBoxTensor**() const

inline double ***getRecBoxTensor**()

inline const double ***getRecBoxTensor**() const

inline double *getBoxOrigin()

inline const double *getBoxOrigin() const

inline double getVolume() const

inline void toFaceDistance(double *dd) const

inline void phys2Inter(double *i_v, const VALUETYPE *p_v) const

inline void inter2Phys(VALUETYPE *p_v, const double *i_v) const

inline bool isPeriodic(const int dim) const

inline double *getShiftVec(const int index = 0)

inline const double *getShiftVec(const int index = 0) const

inline int getShiftIndex(const int *idx) const

inline int getNullShiftIndex() const

inline void shiftCoord(const int *idx, VALUETYPE &x, VALUETYPE &y, VALUETYPE &z) const

inline void diffNearestNeighbor(const VALUETYPE *r0, const VALUETYPE *r1, VALUETYPE *phys) const

inline virtual void diffNearestNeighbor(const VALUETYPE x0, const VALUETYPE y0, const VALUETYPE z0, const VALUETYPE x1, const VALUETYPE y1, const VALUETYPE z1, VALUETYPE &dx, VALUETYPE &dy, VALUETYPE &dz) const

inline virtual void diffNearestNeighbor(const VALUETYPE x0, const VALUETYPE y0, const VALUETYPE z0, const VALUETYPE x1, const VALUETYPE y1, const VALUETYPE z1, VALUETYPE &dx, VALUETYPE &dy, VALUETYPE &dz, int &shift_x, int &shift_y, int &shift_z) const

inline virtual void diffNearestNeighbor(const VALUETYPE x0, const VALUETYPE y0, const VALUETYPE z0, const VALUETYPE x1, const VALUETYPE y1, const VALUETYPE z1, VALUETYPE &dx, VALUETYPE &dy, VALUETYPE &dz, VALUETYPE &shift_x, VALUETYPE &shift_y, VALUETYPE &shift_z) const

Public Static Functions

static inline int compactIndex(const int *idx)

static inline int getNumbShiftVec()

static inline int getShiftVecTotalSize()

Protected Functions

void **computeShiftVec**()

inline double ***getInterShiftVec**(const int index = 0)

inline const double ***getInterShiftVec**(const int index = 0) const

Protected Attributes

double **shift_vec**[shift_vec_size]

double **inter_shift_vec**[shift_vec_size]

Protected Static Functions

static inline int **index3to1**(const int tx, const int ty, const int tz)

Protected Static Attributes

static const int **SPACENDIM** = 3

static const int **DBOX_XX** = 1

static const int **DBOX_YY** = 1

static const int **DBOX_ZZ** = 1

static const int **NBOX_XX** = DBOX_XX * 2 + 1

static const int **NBOX_YY** = DBOX_YY * 2 + 1

static const int **NBOX_ZZ** = DBOX_ZZ * 2 + 1

static const int **shift_info_size** = NBOX_XX * NBOX_YY * NBOX_ZZ

static const int **shift_vec_size** = SPACENDIM * shift_info_size

24.3.3 Unions

Union U_Flt64_Int64

- Defined in file_source_lib_include_env_mat_nvnmd.h

Union Documentation

union **U_Flt64_Int64**

Public Members

double **nflt**

int64_t **nint**

24.3.4 Functions

Template Function add_flt_nvnmd

- Defined in file_source_lib_include_env_mat_nvnmd.h

Function Documentation

template<class **T**>
void **add_flt_nvnmd**(T &y, T x1, T x2)

Function build_nlist(std::vector<std::vector<int»&, std::vector<std::vector<int»&, const std::vector<double>&, const int&, const double&, const double&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const SimulationRegion<double>&, const std::vector<int>&)

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **build_nlist**(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const std::vector<double> &coord, const int &nloc, const double &rc0, const double &rc1, const std::vector<int> &nat_stt_, const std::vector<int> &nat_end_, const std::vector<int> &ext_stt_, const std::vector<int> &ext_end_, const SimulationRegion<double> &region, const std::vector<int> &global_grid)

Function build_nlist(std::vector<std::vector<int»&, std::vector<std::vector<int»&, const std::vector<double>&, const double&, const double&, const std::vector<int>&, const SimulationRegion<double>&)

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **build_nlist**(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const std::vector<double> &coord, const double &rc0, const double &rc1, const std::vector<int> &grid, const SimulationRegion<double> &region)

Function build_nlist(std::vector<std::vector<int»&, std::vector<std::vector<int»&, const std::vector<double>&, const std::vector<int>&, const std::vector<int>&, const double&, const double&, const std::vector<int>&, const SimulationRegion<double>&)

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **build_nlist**(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const std::vector<double> &coord, const std::vector<int> &sel0, const std::vector<int> &sel1, const double &rc0, const double &rc1, const std::vector<int> &grid, const SimulationRegion<double> &region)

Function build_nlist(std::vector<std::vector<int»&, std::vector<std::vector<int»&, const std::vector<double>&, const double&, const double&, const SimulationRegion<double> *)

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **build_nlist**(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const std::vector<double> &coord, const double &rc0_, const double &rc1_, const SimulationRegion<double> *region = NULL)

Function compute_descriptor(std::vector<double>&, std::vector<double>&, std::vector<double>&, const std::vector<double>&, const int&, const std::vector<int>&, const SimulationRegion<double>&, const bool&, const int&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const int, const int, const int, const int)

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

inline void **compute_descriptor**(std::vector<double> &descrpt_a, std::vector<double> &descrpt_r, std::vector<double> &rot_mat, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const int &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int> &fmt_nlist_r, const std::vector<int> &sec_a, const std::vector<int> &sec_r, const int axis0_type, const int axis0_idx, const int axis1_type, const int axis1_idx)

Function compute_descriptor(std::vector<double>&, std::vector<double>&, std::vector<double>&, std::vector<double>&, std::vector<double>&, std::vector<double>&, std::vector<double>&, const std::vector<double>&, const int&, const std::vector<int>&, const SimulationRegion<double>&, const bool&, const int&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const int, const int, const int, const int)

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

inline void **compute_descriptor**(std::vector<double> &descrpt_a, std::vector<double> &descrpt_a_deriv, std::vector<double> &descrpt_r, std::vector<double> &descrpt_r_deriv, std::vector<double> &rij_a, std::vector<double> &rij_r, std::vector<double> &rot_mat, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const int &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int> &fmt_nlist_r, const std::vector<int> &sec_a, const std::vector<int> &sec_r, const int axis0_type, const int axis0_idx, const int axis1_type, const int axis1_idx)

Function compute_descriptor_se_a_ef_para

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

inline void **compute_descriptor_se_a_ef_para**(std::vector<double> &descrpt_a, std::vector<double> &descrpt_a_deriv, std::vector<double> &rij_a, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const std::vector<double> &efield, const int &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int> &sec_a, const double &rmin, const double &rmax)

Function compute_descriptor_se_a_ef_vert

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

inline void **compute_descriptor_se_a_ef_vert**(std::vector<double> &descrpt_a, std::vector<double> &descrpt_a_deriv, std::vector<double> &rij_a, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const std::vector<double> &efield, const int &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int> &sec_a, const double &rmin, const double &rmax)

Function compute_descriptor_se_a_extf

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

inline void **compute_descriptor_se_a_extf**(std::vector<double> &descrpt_a, std::vector<double> &descrpt_a_deriv, std::vector<double> &rij_a, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const std::vector<double> &efield, const int &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int> &sec_a, const double &rmin, const double &rmax)

Function compute_dRdT

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "compute_dRdT" with arguments (double (*), const double*, const double*, const double*) in doxygen xml output for project "core" from directory: _build/core/xml/. Potential matches:

```
- void compute_dRdT(double (*dRdT)[9], const double *r1, const double *r2, const double *rot)
```

Function compute_dRdT_1

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "compute_dRdT_1" with arguments (double (*), const double*, const double*, const double*) in doxygen xml output for project "core" from directory: _build/core/xml/. Potential matches:

```
- void compute_dRdT_1(double (*dRdT)[9], const double *r1, const double *r2, const double *rot)
```

Function compute_dRdT_2

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "compute_dRdT_2" with arguments (double (*), const double*, const double*, const double*) in doxygen xml output for project "core" from directory: _build/core/xml/. Potential matches:

```
- void compute_dRdT_2(double (*dRdT)[9], const double *r1, const double *r2, const double *rot)
```

Function copy_coord

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **copy_coord**(std::vector<double> &out_c, std::vector<int> &out_t, std::vector<int> &mapping, std::vector<int> &ncell, std::vector<int> &ngcell, const std::vector<double> &in_c, const std::vector<int> &in_t, const double &rc, const SimulationRegion<double> &region)

Template Function deepmd::build_nlist_cpu

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

template<typename **FPTYPE**>
int **deepmd::build_nlist_cpu**(InputNlist &nlist, int *max_list_size, const FPTYPE *c_cpy, const int &nloc, const int &nall, const int &mem_size, const float &rcut)

Template Function deepmd::build_nlist_gpu

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

template<typename **FPTYPE**>
int **deepmd::build_nlist_gpu**(InputNlist &nlist, int *max_list_size, int *nlist_data, const FPTYPE *c_cpy, const int &nloc, const int &nall, const int &mem_size, const float &rcut)

Template Function deepmd::compute_cell_info

- Defined in file_source_lib_include_coord.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::compute_cell_info**(int *cell_info, const float &rcut, const deepmd::Region<FPTYPE> &region)

Function deepmd::convert_nlist

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **deepmd::convert_nlist**(InputNlist &to_nlist, std::vector<std::vector<int>> &from_nlist)

Construct the InputNlist with a two-dimensional vector.

Parameters

- **to_nlist** – InputNlist struct which stores the neighbor information of the core region atoms.
- **from_nlist** – Vector which stores the neighbor information of the core region atoms.

Function deepmd::convert_nlist_gpu_device

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **deepmd::convert_nlist_gpu_device**(InputNlist &gpu_nlist, InputNlist &cpu_nlist, int *&gpu_memory, const int &max_nbor_size)

Convert the a host memory InputNlist to a device memory InputNlist.

Parameters

- **cpu_nlist** – Host memory InputNlist struct which stores the neighbor information of the core region atoms
- **gpu_nlist** – Device memory InputNlist struct which stores the neighbor information of the core region atoms
- **gpu_memory** – Device array which stores the elements of gpu_nlist
- **max_nbor_size** –

Template Function deepmd::convert_to_inter_cpu

- Defined in file_source_lib_include_region.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::convert_to_inter_cpu**(FPTYPE *ri, const Region<FPTYPE> &region, const FPTYPE *rp)

Template Function deepmd::convert_to_inter_gpu

- Defined in file_source_lib_include_region.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::convert_to_inter_gpu**(FPTYPE *ri, const Region<FPTYPE> &region, const FPTYPE *rp)

Template Function deepmd::convert_to_phys_cpu

- Defined in file_source_lib_include_region.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::convert_to_phys_cpu**(FPTYPE *rp, const Region<FPTYPE> &region, const FPTYPE *ri)

Template Function deepmd::convert_to_phys_gpu

- Defined in file_source_lib_include_region.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::convert_to_phys_gpu**(FPTYPE *rp, const Region<FPTYPE> &region, const FPTYPE *ri)

Template Function deepmd::copy_coord_cpu

- Defined in file_source_lib_include_coord.h

Function Documentation

template<typename **FPTYPE**>
int **deepmd::copy_coord_cpu**(FPTYPE *out_c, int *out_t, int *mapping, int *nall, const FPTYPE *in_c, const int *in_t, const int &nloc, const int &mem_nall, const float &rcut, const deepmd::Region<FPTYPE> &region)

Template Function deepmd::copy_coord_gpu

- Defined in file_source_lib_include_coord.h

Function Documentation

template<typename **FPTYPE**>
int **deepmd::copy_coord_gpu**(FPTYPE *out_c, int *out_t, int *mapping, int *nall, int *int_data, const FPTYPE *in_c, const int *in_t, const int &nloc, const int &mem_nall, const int &loc_cellnum, const int &total_cellnum, const int *cell_info, const deepmd::Region<FPTYPE> &region)

Function deepmd::cos_switch(const double&, const double&, const double&)

- Defined in file_source_lib_include_switcher.h

Function Documentation

inline double **deepmd::cos_switch**(const double &xx, const double &rmin, const double &rmax)

Function deepmd::cos_switch(double&, double&, const double&, const double&, const double&)

- Defined in file_source_lib_include_switcher.h

Function Documentation

inline void **deepmd::cos_switch**(double &vv, double &dd, const double &xx, const double &rmin, const double &rmax)

Template Function deepmd::cprod

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<typename **TYPE**>
inline void **deepmd::cprod**(const TYPE *r0, const TYPE *r1, TYPE *r2)

Function deepmd::cum_sum

- Defined in file_source_lib_include_utilities.h

Function Documentation

void **deepmd::cum_sum**(std::vector<int> &sec, const std::vector<int> &n_sel)

Template Function deepmd::delete_device_memory

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::delete_device_memory**(FPTYPE *&device)

Template Function deepmd::dot1

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<typename **TYPE**>
inline TYPE **deepmd::dot1**(const TYPE *r0, const TYPE *r1)

Template Function deepmd::dot2

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<typename **TYPE**>
inline TYPE **deepmd::dot2**(const TYPE *r0, const TYPE *r1)

Template Function deepmd::dot3

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<typename **TYPE**>
inline TYPE **deepmd::dot3**(const TYPE *r0, const TYPE *r1)

Template Function deepmd::dot4

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<typename **TYPE**>
inline TYPE **deepmd::dot4**(const TYPE *r0, const TYPE *r1)

Template Function deepmd::dotmv3

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<typename **TYPE**>
inline void **deepmd::dotmv3**(TYPE *vec_o, const TYPE *tensor, const TYPE *vec_i)

Function deepmd::DPGetDeviceCount

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

inline void **deepmd::DPGetDeviceCount**(int &gpu_num)

Function deepmd::dprc_pairwise_map_cpu

- Defined in file_source_lib_include_pairwise.h

Function Documentation

void **deepmd::dprc_pairwise_map_cpu**(std::vector<int> &forward_qm_map, std::vector<int> &backward_qm_map, std::vector<int> &forward_qmmm_map, std::vector<int> &backward_qmmm_map, int &nloc_qm, int &nloc_qmmm, int &nall_qm, int &nall_qmmm, const std::vector<std::vector<int>> &fragments, const int nloc, const int nall)

DPRc pairwise map.

Parameters

- **forward_qm_map** – [out] Forward map for QM atoms.
- **backward_qm_map** – [out] Backward map for QM atoms.
- **forward_qmmm_map** – [out] Forward map for QM/MM atoms.
- **backward_qmmm_map** – [out] Backward map for QM/MM atoms.
- **nloc_qm** – [out] The number of local QM atoms.
- **nloc_qmmm** – [out] The number of local QM/MM atoms.
- **nall_qm** – [out] The number of all QM atoms, including local and ghost atoms.
- **nall_qmmm** – [out] The number of all QM/MM atoms, including local and ghost atoms.
- **fragments** – [in] The indexes of atoms that each fragment contains. Assume that only the first fragment consists of QM atoms.
- **nloc** – [in] The number of local atoms.
- **nall** – [in] The number of all atoms, including local and ghost atoms.

Function deepmd::DPSetDevice

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

inline cudaError_t **deepmd::DPSetDevice**(int rank)

Template Function deepmd::env_mat_a_cpu

- Defined in file_source_lib_include_env_mat.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::env_mat_a_cpu**(std::vector<FPTYPE> &descrpt_a, std::vector<FPTYPE> &descrpt_a_deriv, std::vector<FPTYPE> &rij_a, const std::vector<FPTYPE> &posi, const std::vector<int> &type, const int &i_idx, const std::vector<int> &fmt_nlist, const std::vector<int> &sec, const float &rmin, const float &rmax)

Template Function deepmd::env_mat_a_nvnmd_quantize_cpu

- Defined in file_source_lib_include_env_mat_nvnmd.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::env_mat_a_nvnmd_quantize_cpu**(std::vector<FPTYPE> &descrpt_a, std::vector<FPTYPE> &descrpt_a_deriv, std::vector<FPTYPE> &rij_a, const std::vector<FPTYPE> &posi, const std::vector<int> &type, const int &i_idx, const std::vector<int> &fmt_nlist, const std::vector<int> &sec, const float &rmin, const float &rmax)

Function deepmd::env_mat_nbor_update

- Defined in file_source_lib_include_prod_env_mat.h

Function Documentation

void deepmd::env_mat_nbor_update(InputNlist &inlist, InputNlist &gpu_inlist, int &max_nbor_size, int *&nbor_list_dev, const int *mesh, const int size)

Template Function deepmd::env_mat_r_cpu

- Defined in file_source_lib_include_env_mat.h

Function Documentation

template<typename FPTYPE>
void deepmd::env_mat_r_cpu(std::vector<FPTYPE> &descrpt_a, std::vector<FPTYPE> &descrpt_a_deriv, std::vector<FPTYPE> &rij_a, const std::vector<FPTYPE> &posi, const std::vector<int> &type, const int &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int> &sec_a, const float &rmin, const float &rmax)

Template Function deepmd::ewald_recp

- Defined in file_source_lib_include_ewald.h

Function Documentation

template<typename VALUETYPE>
void deepmd::ewald_recp(VALUETYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<VALUETYPE> &charge, const deepmd::Region<VALUETYPE> &region, const EwaldParameters<VALUETYPE> &param)

Function deepmd::filter_ftype_gpu

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void deepmd::filter_ftype_gpu(int *ftype_out, const int *ftype_in, const int nloc)

Filter the fake atom type.

If >=0, set to 0; if <0, set to -1.

Parameters

- **ftype_out** – The output filtered atom type.
- **ftype_in** – The input atom type.
- **nloc** – The number of atoms.

Template Function deepmd::format_nbor_list_gpu

- Defined in file_source_lib_include_fmt_nlist.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::format_nbor_list_gpu**(int *nlist, const FPTYPE *coord, const int *type, const deepmd::InputNlist &gpu_inlist, int *array_int, uint_64 *array_longlong, const int max_nbor_size, const int nloc, const int nall, const float rcut, const std::vector<int> sec)

Template Function deepmd::format_nlist_cpu

- Defined in file_source_lib_include_fmt_nlist.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::format_nlist_cpu**(int *nlist, const InputNlist &in_nlist, const FPTYPE *coord, const int *type, const int nloc, const int nall, const float rcut, const std::vector<int> sec)

Function deepmd::free_nlist_gpu_device

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **deepmd::free_nlist_gpu_device**(InputNlist &gpu_nlist)

Reclaim the allocated device memory of struct InputNlist.

Parameters

gpu_nlist – Device memory InputNlist struct which stores the neighbor information of the core region atoms

Template Function deepmd::gelu_cpu

- Defined in file_source_lib_include_gelu.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::gelu_cpu**(FPTYPE *out, const FPTYPE *xx, const int_64 size)

Template Function deepmd::gelu_gpu

- Defined in file_source_lib_include_gelu.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::gelu_gpu**(FPTYPE *out, const FPTYPE *xx, const int_64 size)

Template Function deepmd::gelu_grad_cpu

- Defined in file_source_lib_include_gelu.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::gelu_grad_cpu**(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const int_64 size)

Template Function deepmd::gelu_grad_gpu

- Defined in file_source_lib_include_gelu.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::gelu_grad_gpu**(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const int_64 size)

Template Function deepmd::gelu_grad_grad_cpu

- Defined in file_source_lib_include_gelu.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::gelu_grad_grad_cpu**(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const FPTYPE *dy_2, const int_64 size)

Template Function deepmd::gelu_grad_grad_gpu

- Defined in file_source_lib_include_gelu.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::gelu_grad_grad_gpu**(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const FPTYPE *dy_2, const int_64 size)

Function deepmd::group_atoms_cpu

- Defined in file_source_lib_include_pairwise.h

Function Documentation

void **deepmd::group_atoms_cpu**(std::vector<std::vector<int>> &fragments, const std::vector<int> &idxs)

Group atoms into different fragments according to indexes.

Parameters

- **fragments** – [out] The indexes of atoms that each fragment contains. Fragment has been sorted.
- **idxs** – [in] The indexes of the fragment that each atom belongs to. -1 will be ignored.

Template Function deepmd::init_region_cpu

- Defined in file_source_lib_include_region.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::init_region_cpu**(Region<FPTYPE> &region, const FPTYPE *boxt)

Template Function deepmd::invsqrt

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<typename **TYPE**>
inline TYPE **deepmd::invsqrt**(const TYPE x)

Specialized Template Function deepmd::invsqrt< double >

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<>
inline double **deepmd::invsqrt**<double>(const double x)

Specialized Template Function deepmd::invsqrt< float >

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<>
inline float **deepmd::invsqrt**<float>(const float x)

Template Function deepmd::malloc_device_memory(FPTYPE *&, const std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::malloc_device_memory**(FPTYPE *&device, const std::vector<FPTYPE> &host)

Template Function deepmd::malloc_device_memory(FPTYPE *&, const int)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::malloc_device_memory**(FPTYPE *&device, const int size)

Template Function deepmd::malloc_device_memory(FPTYPE *&, std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_rocm.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::malloc_device_memory**(FPTYPE *&device, std::vector<FPTYPE> &host)

Template Function deepmd::malloc_device_memory_sync(FPTYPE *&, const std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::malloc_device_memory_sync**(FPTYPE *&device, const std::vector<FPTYPE> &host)

Template Function deepmd::malloc_device_memory_sync(FPTYPE *&, const FPTYPE *, const int)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::malloc_device_memory_sync**(FPTYPE *&device, const FPTYPE *host, const int size)

Template Function deepmd::malloc_device_memory_sync(FPTYPE *&, std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_rocm.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::malloc_device_memory_sync**(FPTYPE *&device, std::vector<FPTYPE> &host)

Template Function deepmd::map_aparam_cpu

- Defined in file_source_lib_include_map_aparam.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::map_aparam_cpu**(FPTYPE *output, const FPTYPE *aparam, const int *nlist, const int &nloc, const int &nnei, const int &numb_aparam)

Function deepmd::max_numneigh

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

int **deepmd::max_numneigh**(const InputNlist &to_nlist)

Compute the max number of neighbors within the core region atoms.

Parameters
: **to_nlist** – InputNlist struct which stores the neighbor information of the core region atoms.

Return values
: **max** – number of neighbors

Returns
: integer

Template Function deepmd::memcpy_device_to_host(const FPTYPE *, std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::memcpy_device_to_host**(const FPTYPE *device, std::vector<FPTYPE> &host)

Template Function deepmd::memcpy_device_to_host(const FPTYPE *, FPTYPE *, const int)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::memcpy_device_to_host**(const FPTYPE *device, FPTYPE *host, const int size)

Template Function deepmd::memcpy_host_to_device(FPTYPE *, const std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::memcpy_host_to_device**(FPTYPE *device, const std::vector<FPTYPE> &host)

Template Function deepmd::memcpy_host_to_device(FPTYPE *, const FPTYPE *, const int)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::memcpy_host_to_device**(FPTYPE *device, const FPTYPE *host, const int size)

Template Function deepmd::memcpy_host_to_device(FPTYPE *, std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_rocm.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::memcpy_host_to_device**(FPTYPE *device, std::vector<FPTYPE> &host)

Template Function deepmd::memset_device_memory

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::memset_device_memory**(FPTYPE *device, const int var, const int size)

Template Function deepmd::neighbor_stat_gpu

- Defined in file_source_lib_include_neighbor_stat.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::neighbor_stat_gpu**(const FPTYPE *coord, const int *type, const int nloc, const deepmd::InputNlist &gpu_nlist, int *max_nbor_size, FPTYPE *min_nbor_dist, const int ntypes, const int MAX_NNEI)

Template Function deepmd::normalize_coord_cpu

- Defined in file_source_lib_include_coord.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::normalize_coord_cpu**(FPTYPE *coord, const int natom, const deepmd::Region<FPTYPE> &region)

Template Function deepmd::normalize_coord_gpu

- Defined in file_source_lib_include_coord.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::normalize_coord_gpu**(FPTYPE *coord, const int natom, const deepmd::Region<FPTYPE> &region)

Template Function deepmd::pair_tab_cpu

- Defined in file_source_lib_include_pair_tab.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::pair_tab_cpu**(FPTYPE *energy, FPTYPE *force, FPTYPE *virial, const double *table_info, const double *table_data, const FPTYPE *rij, const FPTYPE *scale, const int *type, const int *nlist, const int *natoms, const std::vector<int> &sel_a, const std::vector<int> &sel_r)

Template Function deepmd::prod_env_mat_a_cpu

- Defined in file_source_lib_include_prod_env_mat.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_env_mat_a_cpu**(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const FPTYPE *coord, const int *type, const InputNlist &inlist, const int max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int nloc, const int nall, const float rcut, const float rcut_smth, const std::vector<int> sec, const int *f_type = NULL)

Template Function deepmd::prod_env_mat_a_gpu

- Defined in file_source_lib_include_prod_env_mat.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_env_mat_a_gpu**(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const FPTYPE *coord, const int *type, const InputNlist &gpu_inlist, int *array_int, unsigned long long *array_longlong, const int max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int nloc, const int nall, const float rcut, const float rcut_smth, const std::vector<int> sec, const int *f_type = NULL)

Template Function deepmd::prod_env_mat_a_nvnmd_quantize_cpu

- Defined in file_source_lib_include_prod_env_mat_nvnmd.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_env_mat_a_nvnmd_quantize_cpu**(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const FPTYPE *coord, const int *type, const InputNlist &inlist, const int max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int nloc, const int nall, const float rcut, const float rcut_smth, const std::vector<int> sec, const int *f_type = NULL)

Template Function deepmd::prod_env_mat_r_cpu

- Defined in file_source_lib_include_prod_env_mat.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_env_mat_r_cpu**(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const FPTYPE *coord, const int *type, const InputNlist &inlist, const int max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int nloc, const int nall, const float rcut, const float rcut_smth, const std::vector<int> sec)

Template Function deepmd::prod_env_mat_r_gpu

- Defined in file_source_lib_include_prod_env_mat.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_env_mat_r_gpu**(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const FPTYPE *coord, const int *type, const InputNlist &gpu_inlist, int *array_int, unsigned long long *array_longlong, const int max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int nloc, const int nall, const float rcut, const float rcut_smth, const std::vector<int> sec)

Template Function deepmd::prod_force_a_cpu(FPTYPE *, const FPTYPE *, const FPTYPE *, const int *, const int, const int, const int, const int)

- Defined in file_source_lib_include_prod_force.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_force_a_cpu**(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE *in_deriv, const int *nlist, const int nloc, const int nall, const int nnei, const int nframes)

Produce force from net_deriv and in_deriv.

Template Parameters
FPTYPE – float or double

Parameters

- **force** – [out] Atomic forces.
- **net_deriv** – [in] Net derivative.
- **in_deriv** – [in] Environmental derivative.

- **nlist** – [in] Neighbor list.
- **nloc** – [in] The number of local atoms.
- **nall** – [in] The number of all atoms, including ghost atoms.
- **nnei** – [in] The number of neighbors.
- **nframes** – [in] The number of frames.

Template Function deepmd::prod_force_a_cpu(FPTYPE *, const FPTYPE *, const FPTYPE *, const int *, const int, const int, const int, const int, const int, const int)

- Defined in file_source_lib_include_prod_force.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_force_a_cpu**(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE *in_deriv, const int *nlist, const int nloc, const int nall, const int nnei, const int nframes, const int thread_nloc, const int thread_start_index)

Produce force from net_deriv and in_deriv.

This function is used for multi-threading. Only part of atoms are computed in this thread. They will be comptued in parallel.

Template Parameters
FPTYPE – float or double

Parameters

- **force** – [out] Atomic forces.
- **net_deriv** – [in] Net derivative.
- **in_deriv** – [in] Environmental derivative.
- **nlist** – [in] Neighbor list.
- **nloc** – [in] The number of local atoms.
- **nall** – [in] The number of all atoms, including ghost atoms.
- **nnei** – [in] The number of neighbors.
- **nframes** – [in] The number of frames.
- **thread_nloc** – [in] The number of local atoms to be computed in this thread.
- **thread_start_index** – [in] The start index of local atoms to be computed in this thread. The index should be in [0, nloc).

Template Function deepmd::prod_force_a_gpu

- Defined in file_source_lib_include_prod_force.h

Function Documentation

template<typename FPTYPE>
void deepmd::prod_force_a_gpu(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE *in_deriv, const int *nlist, const int nloc, const int nall, const int nnei, const int nframes)

Template Function deepmd::prod_force_grad_a_cpu

- Defined in file_source_lib_include_prod_force_grad.h

Function Documentation

template<typename FPTYPE>
void deepmd::prod_force_grad_a_cpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const int *nlist, const int nloc, const int nnei, const int nframes)

Template Function deepmd::prod_force_grad_a_gpu

- Defined in file_source_lib_include_prod_force_grad.h

Function Documentation

template<typename FPTYPE>
void deepmd::prod_force_grad_a_gpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const int *nlist, const int nloc, const int nnei, const int nframes)

Template Function deepmd::prod_force_grad_r_cpu

- Defined in file_source_lib_include_prod_force_grad.h

Function Documentation

template<typename FPTYPE>
void deepmd::prod_force_grad_r_cpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const int *nlist, const int nloc, const int nnei, const int nframes)

Template Function deepmd::prod_force_grad_r_gpu

- Defined in file_source_lib_include_prod_force_grad.h

Function Documentation

template<typename FPTYPE>
void deepmd::prod_force_grad_r_gpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const int *nlist, const int nloc, const int nnei, const int nframes)

Template Function deepmd::prod_force_r_cpu

- Defined in file_source_lib_include_prod_force.h

Function Documentation

template<typename FPTYPE>
void deepmd::prod_force_r_cpu(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE *in_deriv, const int *nlist, const int nloc, const int nall, const int nnei, const int nframes)

Template Function deepmd::prod_force_r_gpu

- Defined in file_source_lib_include_prod_force.h

Function Documentation

template<typename FPTYPE>
void deepmd::prod_force_r_gpu(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE *in_deriv, const int *nlist, const int nloc, const int nall, const int nnei, const int nframes)

Template Function deepmd::prod_virial_a_cpu

- Defined in file_source_lib_include_prod_virial.h

Function Documentation

template<typename FPTYPE>
void deepmd::prod_virial_a_cpu(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int nnei)

Template Function deepmd::prod_virial_a_gpu

- Defined in file_source_lib_include_prod_virial.h

Function Documentation

template<typename FPTYPE>
void deepmd::prod_virial_a_gpu(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int nnei)

Template Function deepmd::prod_virial_grad_a_cpu

- Defined in file_source_lib_include_prod_virial_grad.h

Function Documentation

template<typename FPTYPE>
void deepmd::prod_virial_grad_a_cpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nnei)

Template Function deepmd::prod_virial_grad_a_gpu

- Defined in file_source_lib_include_prod_virial_grad.h

Function Documentation

template<typename FPTYPE>
void deepmd::prod_virial_grad_a_gpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nnei)

Template Function deepmd::prod_virial_grad_r_cpu

- Defined in file_source_lib_include_prod_virial_grad.h

Function Documentation

template<typename FPTYPE>
void deepmd::prod_virial_grad_r_cpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nnei)

Template Function deepmd::prod_virial_grad_r_gpu

- Defined in file_source_lib_include_prod_virial_grad.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_virial_grad_r_gpu**(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nnei)

Template Function deepmd::prod_virial_r_cpu

- Defined in file_source_lib_include_prod_virial.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_virial_r_cpu**(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int nnei)

Template Function deepmd::prod_virial_r_gpu

- Defined in file_source_lib_include_prod_virial.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_virial_r_gpu**(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int nnei)

Template Function deepmd::soft_min_switch_cpu

- Defined in file_source_lib_include_soft_min_switch.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::soft_min_switch_cpu**(FPTYPE *sw_value, FPTYPE *sw_deriv, const FPTYPE *rij, const int *nlist, const int &nloc, const int &nnei, const FPTYPE &alpha, const FPTYPE &rmin, const FPTYPE &rmax)

Template Function deepmd::soft_min_switch_force_cpu

- Defined in file_source_lib_include_soft_min_switch_force.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::soft_min_switch_force_cpu**(FPTYPE *force, const FPTYPE *du, const FPTYPE *sw_deriv, const int *nlist, const int nloc, const int nall, const int nnei)

Template Function deepmd::soft_min_switch_force_grad_cpu

- Defined in file_source_lib_include_soft_min_switch_force_grad.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::soft_min_switch_force_grad_cpu**(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *sw_deriv, const int *nlist, const int nloc, const int nnei)

Template Function deepmd::soft_min_switch_virial_cpu

- Defined in file_source_lib_include_soft_min_switch_virial.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::soft_min_switch_virial_cpu**(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *du, const FPTYPE *sw_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int nnei)

Template Function deepmd::soft_min_switch_virial_grad_cpu

- Defined in file_source_lib_include_soft_min_switch_virial_grad.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::soft_min_switch_virial_grad_cpu**(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *sw_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nnei)

Function deepmd::spline3_switch

- Defined in file_source_lib_include_switcher.h

Function Documentation

inline void **deepmd::spline3_switch**(double &vv, double &dd, const double &xx, const double &rmin, const double &rmax)

Template Function deepmd::spline5_switch

- Defined in file_source_lib_include_switcher.h

Function Documentation

template<typename **FPTYPE**>
inline void **deepmd::spline5_switch**(FPTYPE &vv, FPTYPE &dd, const FPTYPE &xx, const float &rmin, const float &rmax)

Template Function deepmd::tabulate_fusion_se_a_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_a_cpu**(FPTYPE *out, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *two_embed, const int nloc, const int nnei, const int last_layer_size, const bool is_sorted = true)

Template Function deepmd::tabulate_fusion_se_a_gpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_a_gpu**(FPTYPE *out, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *two_embed, const int nloc, const int nnei, const int last_layer_size, const bool is_sorted = true)

Template Function deepmd::tabulate_fusion_se_a_grad_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_a_grad_cpu**(FPTYPE *dy_dem_x, FPTYPE *dy_dem, FPTYPE *dy_dtwo, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *two_embed, const FPTYPE *dy, const int nloc, const int nnei, const int last_layer_size, const bool is_sorted = true)

Template Function deepmd::tabulate_fusion_se_a_grad_gpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_a_grad_gpu**(FPTYPE *dy_dem_x, FPTYPE *dy_dem, FPTYPE *dy_dtwo, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *two_embed, const FPTYPE *dy, const int nloc, const int nnei, const int last_layer_size, const bool is_sorted = true)

Template Function deepmd::tabulate_fusion_se_a_grad_grad_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_a_grad_grad_cpu**(FPTYPE *dz_dy, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *two_embed, const FPTYPE *dz_dy_dem_x, const FPTYPE *dz_dy_dem, const FPTYPE *dz_dy_dtwo, const int nloc, const int nnei, const int last_layer_size, const bool is_sorted = true)

Template Function deepmd::tabulate_fusion_se_a_grad_grad_gpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_a_grad_grad_gpu**(FPTYPE *dz_dy, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *two_embed, const FPTYPE *dz_dy_dem_x, const FPTYPE *dz_dy_dem, const FPTYPE *dz_dy_dtwo, const int nloc, const int nnei, const int last_layer_size, const bool is_sorted = true)

Template Function deepmd::tabulate_fusion_se_r_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_r_cpu**(FPTYPE *out, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em, const int nloc, const int nnei, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_r_gpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_r_gpu**(FPTYPE *out, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em, const int nloc, const int nnei, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_r_grad_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_grad_cpu(FPTYPE *dy_dem, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em, const FPTYPE *dy, const int nloc, const int nnei, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_r_grad_gpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_grad_gpu(FPTYPE *dy_dem, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em, const FPTYPE *dy, const int nloc, const int nnei, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_r_grad_grad_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_grad_grad_cpu(FPTYPE *dz_dy, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em, const FPTYPE *dz_dy_dem, const int nloc, const int nnei, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_r_grad_grad_gpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_grad_grad_gpu(FPTYPE *dz_dy, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em, const FPTYPE *dz_dy_dem, const int nloc, const int nnei, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_t_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_t_cpu**(FPTYPE *out, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const int nloc, const int nnei_i, const int nnei_j, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_t_gpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_t_gpu**(FPTYPE *out, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const int nloc, const int nnei_i, const int nnei_j, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_t_grad_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_t_grad_cpu**(FPTYPE *dy_dem_x, FPTYPE *dy_dem, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *dy, const int nloc, const int nnei_i, const int nnei_j, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_t_grad_gpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_grad_gpu(FPTYPE *dy_dem_x, FPTYPE *dy_dem, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *dy, const int nloc, const int nnei_i, const int nnei_j, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_t_grad_grad_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_grad_grad_cpu(FPTYPE *dz_dy, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *dz_dy_dem_x, const FPTYPE *dz_dy_dem, const int nloc, const int nnei_i, const int nnei_j, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_t_grad_grad_gpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_grad_grad_gpu(FPTYPE *dz_dy, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *dz_dy_dem_x, const FPTYPE *dz_dy_dem, const int nloc, const int nnei_i, const int nnei_j, const int last_layer_size)

Template Function deepmd::test_encoding_decoding_nbor_info_gpu

- Defined in file_source_lib_include_fmt_nlist.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::test_encoding_decoding_nbor_info_gpu**(uint_64 *key, int *out_type, int *out_index,
const int *in_type, const FPTYPE *in_dist,
const int *in_index, const int size_of_array)

Function deepmd::use_nei_info_cpu

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **deepmd::use_nei_info_cpu**(int *nlist, int *ntype, bool *nmask, const int *type, const int *nlist_map,
const int nloc, const int nnei, const int ntypes, const bool b_nlist_map)

Function deepmd::use_nei_info_gpu

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **deepmd::use_nei_info_gpu**(int *nlist, int *ntype, bool *nmask, const int *type, const int *nlist_map,
const int nloc, const int nnei, const int ntypes, const bool b_nlist_map)

Function deepmd::use_nlist_map

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **deepmd::use_nlist_map**(int *nlist, const int *nlist_map, const int nloc, const int nnei)

Template Function deepmd::volume_cpu

- Defined in file_source_lib_include_region.h

Function Documentation

template<typename FPTYPE>
FPTYPE deepmd::volume_cpu(const Region<FPTYPE> &region)

Template Function deepmd::volume_gpu

- Defined in file_source_lib_include_region.h

Function Documentation

template<typename FPTYPE>
void deepmd::volume_gpu(FPTYPE *volume, const Region<FPTYPE> &region)

Template Function dotmul_flt_nvnmd

- Defined in file_source_lib_include_env_mat_nvnmd.h

Function Documentation

template<class T>
void dotmul_flt_nvnmd(T &y, T *x1, T *x2, int64_t M)

Function DPAssert(cudaError_t, const char *, int, bool)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

inline void DPAssert(cudaError_t code, const char *file, int line, bool abort = true)

Function DPAssert(hipError_t, const char *, int, bool)

- Defined in file_source_lib_include_gpu_rocm.h

Function Documentation

inline void DPAssert(hipError_t code, const char *file, int line, bool abort = true)

Function env_mat_a

- Defined in file_source_lib_include_env_mat.h

Function Documentation

void **env_mat_a**(std::vector<double> &descrpt_a, std::vector<double> &descrpt_a_deriv, std::vector<double> &rij_a, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const int &i_idx, const std::vector<int> &fmt_nlist, const std::vector<int> &sec, const double &rmin, const double &rmax)

Function env_mat_r

- Defined in file_source_lib_include_env_mat.h

Function Documentation

void **env_mat_r**(std::vector<double> &descrpt_r, std::vector<double> &descrpt_r_deriv, std::vector<double> &rij_r, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const int &i_idx, const std::vector<int> &fmt_nlist, const std::vector<int> &sec, const double &rmin, const double &rmax)

Template Function find_max_expo(int64_t&, T *, int64_t)

- Defined in file_source_lib_include_env_mat_nvnmd.h

Function Documentation

template<class **T**>
void **find_max_expo**(int64_t &max_expo, T *x, int64_t M)

Template Function find_max_expo(int64_t&, T *, int64_t, int64_t)

- Defined in file_source_lib_include_env_mat_nvnmd.h

Function Documentation

template<class **T**>
void **find_max_expo**(int64_t &max_expo, T *x, int64_t N, int64_t M)

Template Function format_nlist_i_cpu

- Defined in file_source_lib_include_fmt_nlist.h

Function Documentation

template<typename **FPTYPE**>
int **format_nlist_i_cpu**(std::vector<int> &fmt_nei_idx_a, const std::vector<FPTYPE> &posi, const std::vector<int> &type, const int &i_idx, const std::vector<int> &nei_idx_a, const float &rcut, const std::vector<int> &sec_a)

Function format_nlist_i_fill_a

- Defined in file_source_lib_include_fmt_nlist.h

Function Documentation

int **format_nlist_i_fill_a**(std::vector<int> &fmt_nei_idx_a, std::vector<int> &fmt_nei_idx_r, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const int &i_idx, const std::vector<int> &nei_idx_a, const std::vector<int> &nei_idx_r, const double &rcut, const std::vector<int> &sec_a, const std::vector<int> &sec_r)

Template Function mul_flt_nvnmd

- Defined in file_source_lib_include_env_mat_nvnmd.h

Function Documentation

template<class **T**>
void **mul_flt_nvnmd**(T &y, T x1, T x2)

Function nborAssert(cudaError_t, const char *, int, bool)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

inline void **nborAssert**(cudaError_t code, const char *file, int line, bool abort = true)

Function nborAssert(hipError_t, const char *, int, bool)

- Defined in file_source_lib_include_gpu_rocm.h

Function Documentation

inline void **nborAssert**(hipError_t code, const char *file, int line, bool abort = true)

Function omp_get_num_threads

- Defined in file_source_lib_include_ewald.h

Function Documentation

int **omp_get_num_threads**()

Function omp_get_thread_num

- Defined in file_source_lib_include_ewald.h

Function Documentation

int **omp_get_thread_num**()

Template Function split_flt

- Defined in file_source_lib_include_env_mat_nvnmd.h

Function Documentation

template<class **T**>
void **split_flt**(T x, int64_t &sign, int64_t &expo, int64_t &mant)

24.3.5 Variables

Variable deepmd::ElectrostaticConvertion

- Defined in file_source_lib_include_ewald.h

Variable Documentation

const double `deepmd::ElectrostaticConvertion` = 14.399645354756969950 31

24.3.6 Defines

Define DPErrcheck

- Defined in file_source_lib_include_gpu_cuda.h

Define Documentation

`DPErrcheck`(res)

Define DPErrcheck

- Defined in file_source_lib_include_gpu_rocm.h

Define Documentation

`DPErrcheck`(res)

Define FLT_MASK

- Defined in file_source_lib_include_env_mat_nvnmd.h

Define Documentation

`FLT_MASK`

Define GPU_MAX_NBOR_SIZE

- Defined in file_source_lib_include_gpu_cuda.h

Define Documentation

`GPU_MAX_NBOR_SIZE`

Define GPU_MAX_NBOR_SIZE

- Defined in file_source_lib_include_gpu_rocm.h

Define Documentation

```
GPU_MAX_NBOR_SIZE
```

Define gpuDeviceSynchronize

- Defined in file_source_lib_include_gpu_cuda.h

Define Documentation

```
gpuDeviceSynchronize
```

Define gpuDeviceSynchronize

- Defined in file_source_lib_include_gpu_rocm.h

Define Documentation

```
gpuDeviceSynchronize
```

Define gpuGetLastError

- Defined in file_source_lib_include_gpu_cuda.h

Define Documentation

```
gpuGetLastError
```

Define gpuGetLastError

- Defined in file_source_lib_include_gpu_rocm.h

Define Documentation

```
gpuGetLastError
```

Define gpuMemcpy

- Defined in file_source_lib_include_gpu_cuda.h

Define Documentation

```
gpuMemcpy
```

Define gpuMemcpy

- Defined in file_source_lib_include_gpu_rocm.h

Define Documentation

```
gpuMemcpy
```

Define gpuMemcpyDeviceToDevice

- Defined in file_source_lib_include_gpu_cuda.h

Define Documentation

```
gpuMemcpyDeviceToDevice
```

Define gpuMemcpyDeviceToDevice

- Defined in file_source_lib_include_gpu_rocm.h

Define Documentation

```
gpuMemcpyDeviceToDevice
```

Define gpuMemcpyDeviceToHost

- Defined in file_source_lib_include_gpu_cuda.h

Define Documentation

```
gpuMemcpyDeviceToHost
```

Define gpuMemcpyDeviceToHost

- Defined in file_source_lib_include_gpu_rocm.h

Define Documentation

```
gpuMemcpyDeviceToHost
```

Define gpuMemcpyHostToDevice

- Defined in file_source_lib_include_gpu_cuda.h

Define Documentation

```
gpuMemcpyHostToDevice
```

Define gpuMemcpyHostToDevice

- Defined in file_source_lib_include_gpu_rocm.h

Define Documentation

```
gpuMemcpyHostToDevice
```

Define gpuMemset

- Defined in file_source_lib_include_gpu_cuda.h

Define Documentation

```
gpuMemset
```

Define gpuMemset

- Defined in file_source_lib_include_gpu_rocm.h

Define Documentation

```
gpuMemset
```

Define MOASPNDIM

- Defined in file_source_lib_include_SimulationRegion.h

Define Documentation

```
MOASPNDIM
```

Define NBIT_CUTF

- Defined in file_source_lib_include_env_mat_nvnmd.h

Define Documentation

```
NBIT_CUTF
```

Define NBIT_FLTF

- Defined in file_source_lib_include_env_mat_nvnmd.h

Define Documentation

```
NBIT_FLTF
```

Define nborErrcheck

- Defined in file_source_lib_include_gpu_cuda.h

Define Documentation

`nborErrcheck`(res)

Define nborErrcheck

- Defined in file_source_lib_include_gpu_rocm.h

Define Documentation

`nborErrcheck`(res)

Define SQRT_2_PI

- Defined in file_source_lib_include_device.h

Define Documentation

`SQRT_2_PI`

Define TPB

- Defined in file_source_lib_include_device.h

Define Documentation

`TPB`

24.3.7 Typedefs

Typedef int_64

- Defined in file_source_lib_include_device.h

Typedef Documentation

typedef long long **int_64**

Typedef uint_64

- Defined in file_source_lib_include_device.h

Typedef Documentation

typedef unsigned long long **uint_64**

CHAPTER

TWENTYFIVE

LICENSE

The project DeePMD-kit is licensed under GNU LGPLv3.0.

CHAPTER

TWENTYSIX

AUTHORS AND CREDITS

26.1 Cite DeePMD-kit and methods

- For general purpose,
- If GPU version is used,
- If local frame (loc_frame) is used,
- If DeepPot-SE (se_e2_a, se_e2_r, se_e3, se_atten) is used,
- If three-body embedding DeepPot-SE (se_e3) is used,
- If attention-based descriptor (se_atten, se_atten_v2) is used,
- If DPA-2 descriptor (dpa2) is used,
- If frame-specific parameters (fparam, e.g. electronic temperature) is used,
- If atom-specific parameters (aparam, e.g. electronic temperature) is used,
- If fitting dipole,
- If fitting polarizability,
- If fitting density of states,
- If fitting relative energies,
- If DPLR is used, or se_e2_r and hybrid are used,
- If DPRc is used,
- If interpolation with a pair-wise potential is used,
- If the model is compressed (dp compress),
- If model deviation is computed,
- If relative or atomic model deviation is computed,
- If NVNMD is used,

26.2 Package Contributors

- A bot of @njzjz
- AngelJia
- AnguseZhang
- Anurag Kumar Singh
- Anyang Peng
- Chenqqian Zhang
- Chenxing Luo
- Chun Cai
- Davide Tisi
- Denghui Lu
- Duo
- Eisuke Kawashima
- Futaki Haduki
- GeiduanLiu
- Han Wang
- Hananeh Oliaei
- Harvey Que
- HuangJiameng
- HydrogenSulfate
- Jia-Xin Zhu
- Jiequn Han
- Jingchao Zhang
- Jinzhe Zeng
- Koki MURAOKA
- LiangWenshuo1118
- Linfeng Zhang
- LiuGroupHNU
- Lu
- Lysithea
- Marián Rynik
- Nick Lin
- Rhys Goodall
- Shaochen Shi
- TrellixVulnTeam

- Wanrun Jiang
- Xia, Yu
- Xu zhangmancang
- YWolfeee
- Ye Ding
- Yifan Li□□□
- Yingze Wang
- Yixiao Chen
- Yong-Bin Zhuang
- Zeyu Li
- ZhengdQin
- ZiyaoLi
- baohan
- bwang-ecnu
- deepmodeling
- denghuilu
- dependabot[bot]
- haidi
- hlyang
- hsulab
- hztttt
- iProzd
- imgbot[bot]
- jxxiaoshaoye
- liangadam
- likefallwind
- link89
- marian-code
- mingzhong15
- nahso
- njzjz
- pkulzy
- pre-commit-ci[bot]
- readthedocs-assistant
- shiruosong
- sigbjobo

- tuoping
- wsyxbcl
- zhenyu
- ziyao

26.3 Other Credits

- Zhang ZiXuan for designing the Deepmodeling logo.
- Everyone on the Deepmodeling mailing list for contributing to many discussions and decisions!

CHAPTER

TWENTYSEVEN

LOGO

The logo of DeePMD-kit is a beaver. Beavers were widely distributed in Europe and Asia but became nearly extinct due to hunting. Listed as a first-class state-protected animal in China, the population of beavers in China is less than the giant pandas. We hope that users of DeePMD-kit can enhance the awareness to protect beavers.

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] Duo Zhang, Hangrui Bi, Fu-Zhi Dai, Wanrun Jiang, Linfeng Zhang, and Han Wang. 2022. DPA-1: Pretraining of Attention-based Deep Potential Model for Molecular Simulation. arXiv preprint arXiv:2208.08236.

[1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 4441-4451.

[1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 4441-4451.

[1] Duo Zhang, Hangrui Bi, Fu-Zhi Dai, Wanrun Jiang, Linfeng Zhang, and Han Wang. 2022. DPA-1: Pretraining of Attention-based Deep Potential Model for Molecular Simulation. arXiv preprint arXiv:2208.08236.

[1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 4441-4451.

[1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 4441-4451.

[1] Duo Zhang, Hangrui Bi, Fu-Zhi Dai, Wanrun Jiang, Linfeng Zhang, and Han Wang. 2022. DPA-1: Pretraining of Attention-based Deep Potential Model for Molecular Simulation. arXiv preprint arXiv:2208.08236.

[1] Duo Zhang, Hangrui Bi, Fu-Zhi Dai, Wanrun Jiang, Linfeng Zhang, and Han Wang. 2022. DPA-1: Pretraining of Attention-based Deep Potential Model for Molecular Simulation. arXiv preprint arXiv:2208.08236.

[1] Jinzhe Zeng, Timothy J. Giese, Sölen Ekesan, Darrin M. York, Development of Range-Corrected Deep Learning Potentials for Fast, Accurate Quantum Mechanical/molecular Mechanical Simulations of Chemical Reactions in Solution, J. Chem. Theory Comput., 2021, 17 (11), 6993-7009.

[1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 4441-4451.

[1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 4441-4451.

[1] Jinzhe Zeng, Timothy J. Giese, Sölen Ekesan, Darrin M. York, Development of Range-Corrected Deep Learning Potentials for Fast, Accurate Quantum Mechanical/molecular Mechanical Simulations of Chemical Reactions in Solution, J. Chem. Theory Comput., 2021, 17 (11), 6993-7009.

[1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 4441-4451.

[1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 4441-4451.

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identitymappings in deep residual networks. InComputer Vision - ECCV 2016,pages 630-645. Springer International Publishing, 2016.

[WZHE18] Han Wang, Linfeng Zhang, Jiequn Han, and Weinan E. DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics. Comput. Phys. Comm., 228:178–184, jul 2018. doi:10.1016/j.cpc.2018.03.016.

[ZZL+23] Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E A Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M York, Weinan E, Roberto Car, Linfeng Zhang, and Han Wang. DeePMD-kit v2: A software package for deep potential models. J. Chem. Phys., 159:054801, 2023. doi:10.1063/5.0155600.

[LWC+21] Denghui Lu, Han Wang, Mohan Chen, Lin Lin, Roberto Car, Weinan E, Weile Jia, and Linfeng Zhang. 86 PFLOPS Deep Potential Molecular Dynamics simulation of 100 million atoms with ab initio accuracy. Comput. Phys. Comm., 259:107624, 2021. doi:10.1016/j.cpc.2020.107624.

[ZHW+18] Linfeng Zhang, Jiequn Han, Han Wang, Roberto Car, and Weinan E. Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics. Phys. Rev. Lett., 120(14):143001, 2018. doi:10.1103/PhysRevLett.120.143001.

[ZHW+18] Linfeng Zhang, Jiequn Han, Han Wang, Wissam Saidi, Roberto Car, and Weinan E. End-to-end Symmetry Preserving Inter-atomic Potential Energy Model for Finite and Extended Systems. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems 31, pages 4436–4446. Curran Associates, Inc., 2018. URL: https://dl.acm.org/doi/10.5555/3327345.3327356.

[WWZ+22] Xiaoyang Wang, Yinan Wang, Linfeng Zhang, Fuzhi Dai, and Han Wang. A tungsten deep neural-network potential for simulating mechanical property degradation under fusion service environment. Nucl. Fusion, 62:126013, 2022. doi:10.1088/1741-4326/ac888b.

[ZBD+22] Duo Zhang, Hangrui Bi, Fu-Zhi Dai, Wanrun Jiang, Linfeng Zhang, and Han Wang. DPA-1: Pretraining of Attention-based Deep Potential Model for Molecular Simulation. 2022. doi:10.48550/arXiv.2208.08236.

[ZLZ+23] Duo Zhang, Xinzijian Liu, Xiangyu Zhang, Chengqian Zhang, Chun Cai, Hangrui Bi, Yiming Du, Xuejian Qin, Jiameng Huang, Bowen Li, Yifan Shan, Jinzhe Zeng, Yuzhi Zhang, Siyuan Liu, Yifan Li, Junhan Chang, Xinyan Wang, Shuo Zhou, Jianchuan Liu, Xiaoshan Luo, Zhenyu Wang,

Wanrun Jiang, Jing Wu, Yudi Yang, Jiyuan Yang, Manyi Yang, Fu-Qiang Gong, Linshuang Zhang, Mengchao Shi, Fu-Zhi Dai, Darrin M. York, Shi Liu, Tong Zhu, Zhicheng Zhong, Jian Lv, Jun Cheng, Weile Jia, Mohan Chen, Guolin Ke, Weinan E, Linfeng Zhang, and Han Wang. DPA-2: Towards a universal large atomic model for molecular and material simulation. 2023. doi:10.48550/arXiv.2312.15492.

[ZGL+20] Yuzhi Zhang, Chang Gao, Qianrui Liu, Linfeng Zhang, Han Wang, and Mohan Chen. Warm dense matter simulation via electron temperature dependent deep potential molecular dynamics. Phys. Plasmas, 27(12):122704, 12 2020. doi:10.1063/5.0023265.

[ZCZ+23] Qiyu Zeng, Bo Chen, Shen Zhang, Dongdong Kang, Han Wang, Xiaoxiang Yu, and Jiayu Dai. Full-scale ab initio simulations of laser-driven atomistic dynamics. 2023. doi:10.48550/arXiv.2308.13863.

[ZCW+20] Linfeng Zhang, Mohan Chen, Xifan Wu, Han Wang, Weinan E, and Roberto Car. Deep neural network for the dielectric response of insulators. Phys. Rev. B, 102(4):041121, 2020. doi:10.1103/PhysRevB.102.041121.

[SAZ+20] Grace M Sommers, Marcos F Calegari Andrade, Linfeng Zhang, Han Wang, and Roberto Car. Raman spectrum and polarizability of liquid water from deep neural networks. Phys. Chem. Chem. Phys., 22(19):10592–10602, 2020. doi:10.1039/D0CP01893G.

[ZCY+22] Qiyu Zeng, Bo Chen, Xiaoxiang Yu, Shen Zhang, Dongdong Kang, Han Wang, and Jiayu Dai. Towards large-scale and spatiotemporally resolved diagnosis of electronic density of states by deep learning. Phys. Rev. B, 105:174109, 2022. doi:10.1103/PhysRevB.105.174109.

[ZTGY23] Jinzhe Zeng, Yujun Tao, Timothy J Giese, and Darrin M York. QD\pi : A Quantum Deep Potential Interaction Model for Drug Discovery. J. Chem. Theory Comput., 19:1261–1275, 2023. doi:10.1021/acs.jctc.2c01172.

[ZWM+22] Linfeng Zhang, Han Wang, Maria Carolina Muniz, Athanassios Z Panagiotopoulos, Roberto Car, and Weinan E. A deep potential model with long-range electrostatic interactions. J. Chem. Phys., 156:124107, 2022. doi:10.1063/5.0083669.

[ZGEY21] Jinzhe Zeng, Timothy J Giese, Şölen Ekesan, and Darrin M York. Development of Range-Corrected Deep Learning Potentials for Fast, Accurate Quantum Mechanical/molecular Mechanical Simulations of Chemical Reactions in Solution. J. Chem. Theory Comput., 17:6993–7009, 2021. doi:10.1021/acs.jctc.1c00201.

[WGZ+19] Hao Wang, Xun Guo, Linfeng Zhang, Han Wang, and Jianming Xue. Deep learning inter-atomic potential model for accurate irradiation damage simulations. Appl. Phys. Lett., 114(24):244101, 2019. doi:10.1063/1.5098061.

[LJC+22] Denghui Lu, Wanrun Jiang, Yixiao Chen, Linfeng Zhang, Weile Jia, Han Wang, and Mohan Chen. DP Compress: A Model Compression Scheme for Generating Efficient Deep Potential Models. J. Chem. Theory Comput., 18:5555–5567, 2022. doi:10.1021/acs.jctc.2c00102.

[ZLW+19] Linfeng Zhang, De-Ye Lin, Han Wang, Roberto Car, and Weinan E. Active learning of uniformly accurate interatomic potentials for materials simulation. Phys. Rev. Mater., 3:23804, 2019. doi:10.1103/PhysRevMaterials.3.023804.

[ZZWZ21] Jinzhe Zeng, Linfeng Zhang, Han Wang, and Tong Zhu. Exploring the Chemical Space of Linear Alkane Pyrolysis via Deep Potential GENerator. Energy & Fuels, 35(1):762–769, 2021. doi:10.1021/acs.energyfuels.0c03211.

[MLZ+22] Pinghui Mo, Chang Li, Dan Zhao, Yujia Zhang, Mengchao Shi, Junhua Li, and Jie Liu. Accurate and efficient molecular dynamics based on machine learning and non von Neumann architecture. npj Comput. Mater., 8:107, 2022. doi:10.1038/s41524-022-00773-z.

PYTHON MODULE INDEX

d

deepmd, 253
deepmd.__main__, 1058
deepmd.backend, 253
deepmd.backend.backend, 253
deepmd.backend.dpmodel, 256
deepmd.backend.pytorch, 257
deepmd.backend.suffix, 258
deepmd.backend.tensorflow, 259
deepmd.calculator, 1058
deepmd.common, 1060
deepmd.dpmodel, 260
deepmd.dpmodel.atomic_model, 260
deepmd.dpmodel.atomic_model.base_atomic_model, 261
deepmd.dpmodel.atomic_model.dp_atomic_model, 263
deepmd.dpmodel.atomic_model.linear_atomic_model, 264
deepmd.dpmodel.atomic_model.make_base_atomic_model, 268
deepmd.dpmodel.atomic_model.pairtab_atomic_model, 268
deepmd.dpmodel.common, 374
deepmd.dpmodel.descriptor, 281
deepmd.dpmodel.descriptor.base_descriptor, 281
deepmd.dpmodel.descriptor.dpa1, 281
deepmd.dpmodel.descriptor.hybrid, 288
deepmd.dpmodel.descriptor.make_base_descriptor, 290
deepmd.dpmodel.descriptor.se_e2_a, 290
deepmd.dpmodel.descriptor.se_r, 294
deepmd.dpmodel.fitting, 310
deepmd.dpmodel.fitting.base_fitting, 310
deepmd.dpmodel.fitting.dipole_fitting, 310
deepmd.dpmodel.fitting.dos_fitting, 313
deepmd.dpmodel.fitting.ener_fitting, 315
deepmd.dpmodel.fitting.general_fitting, 317
deepmd.dpmodel.fitting.invar_fitting, 320
deepmd.dpmodel.fitting.make_base_fitting, 323
deepmd.dpmodel.fitting.polarizability_fitting, 324
deepmd.dpmodel.infer, 338
deepmd.dpmodel.infer.deep_eval, 338
deepmd.dpmodel.model, 341
deepmd.dpmodel.model.base_model, 341
deepmd.dpmodel.model.dp_model, 342
deepmd.dpmodel.model.ener_model, 343
deepmd.dpmodel.model.make_model, 343
deepmd.dpmodel.model.model, 344
deepmd.dpmodel.model.spin_model, 344
deepmd.dpmodel.model.transform_output, 347
deepmd.dpmodel.output_def, 374
deepmd.dpmodel.utils, 351
deepmd.dpmodel.utils.batch_size, 351
deepmd.dpmodel.utils.env_mat, 352
deepmd.dpmodel.utils.exclude_mask, 353
deepmd.dpmodel.utils.neighbor_stat, 354
deepmd.dpmodel.utils.network, 355
deepmd.dpmodel.utils.nlist, 359
deepmd.dpmodel.utils.region, 361
deepmd.dpmodel.utils.serialization, 363
deepmd.dpmodel.utils.type_embed, 364
deepmd.dpmodel.utils.update_sel, 365
deepmd.driver, 1063
deepmd.entrypoints, 382
deepmd.entrypoints.convert_backend, 382
deepmd.entrypoints.doc, 383
deepmd.entrypoints.gui, 383
deepmd.entrypoints.main, 384
deepmd.entrypoints.neighbor_stat, 384
deepmd.entrypoints.test, 386
deepmd.env, 1063
deepmd.infer, 387
deepmd.infer.deep_dipole, 387
deepmd.infer.deep_dos, 388
deepmd.infer.deep_eval, 389
deepmd.infer.deep_polar, 395
deepmd.infer.deep_pot, 397
deepmd.infer.deep_tensor, 399
deepmd.infer.deep_wfc, 401
deepmd.infer.model_devi, 402
deepmd.loggers, 411

deepmd.loggers.loggers, 412
deepmd.loggers.training, 413
deepmd.main, 1064
deepmd.pt, 414
deepmd.pt.cxx_op, 639
deepmd.pt.entrypoints, 414
deepmd.pt.entrypoints.main, 414
deepmd.pt.infer, 416
deepmd.pt.infer.deep_eval, 416
deepmd.pt.infer.inference, 419
deepmd.pt.loss, 419
deepmd.pt.loss.denoise, 419
deepmd.pt.loss.dos, 420
deepmd.pt.loss.ener, 422
deepmd.pt.loss.ener_spin, 423
deepmd.pt.loss.loss, 425
deepmd.pt.loss.tensor, 426
deepmd.pt.model, 434
deepmd.pt.model.atomic_model, 434
deepmd.pt.model.atomic_model.base_atomic_model, 434
deepmd.pt.model.atomic_model.dipole_atomic_model, 438
deepmd.pt.model.atomic_model.dos_atomic_model, 439
deepmd.pt.model.atomic_model.dp_atomic_model, 439
deepmd.pt.model.atomic_model.energy_atomic_model, 442
deepmd.pt.model.atomic_model.linear_atomic_model, 442
deepmd.pt.model.atomic_model.pairtab_atomic_model, 446
deepmd.pt.model.atomic_model.polar_atomic_model, 450
deepmd.pt.model.backbone, 464
deepmd.pt.model.backbone.backbone, 464
deepmd.pt.model.backbone.evoformer2b, 465
deepmd.pt.model.descriptor, 468
deepmd.pt.model.descriptor.base_descriptor, 468
deepmd.pt.model.descriptor.descriptor, 468
deepmd.pt.model.descriptor.dpa1, 470
deepmd.pt.model.descriptor.dpa2, 475
deepmd.pt.model.descriptor.env_mat, 479
deepmd.pt.model.descriptor.gaussian_lcc, 479
deepmd.pt.model.descriptor.hybrid, 481
deepmd.pt.model.descriptor.repformer_layer, 483
deepmd.pt.model.descriptor.repformers, 489
deepmd.pt.model.descriptor.se_a, 492
deepmd.pt.model.descriptor.se_atten, 496
deepmd.pt.model.descriptor.se_r, 502
deepmd.pt.model.model, 528
deepmd.pt.model.model.dipole_model, 529
deepmd.pt.model.model.dos_model, 529
deepmd.pt.model.model.dp_model, 530
deepmd.pt.model.model.dp_zbl_model, 531
deepmd.pt.model.model.ener_model, 532
deepmd.pt.model.model.frozen, 533
deepmd.pt.model.model.make_hessian_model, 534
deepmd.pt.model.model.make_model, 535
deepmd.pt.model.model.model, 535
deepmd.pt.model.model.polar_model, 537
deepmd.pt.model.model.spin_model, 537
deepmd.pt.model.model.transform_output, 540
deepmd.pt.model.network, 548
deepmd.pt.model.network.layernorm, 548
deepmd.pt.model.network.mlp, 550
deepmd.pt.model.network.network, 552
deepmd.pt.model.task, 578
deepmd.pt.model.task.atten_lcc, 578
deepmd.pt.model.task.base_fitting, 579
deepmd.pt.model.task.denoise, 579
deepmd.pt.model.task.dipole, 580
deepmd.pt.model.task.dos, 582
deepmd.pt.model.task.ener, 584
deepmd.pt.model.task.fitting, 588
deepmd.pt.model.task.invar_fitting, 591
deepmd.pt.model.task.polarizability, 593
deepmd.pt.model.task.task, 595
deepmd.pt.model.task.type_predict, 595
deepmd.pt.optimizer, 606
deepmd.pt.optimizer.KFWrapper, 606
deepmd.pt.optimizer.LKF, 606
deepmd.pt.train, 609
deepmd.pt.train.training, 609
deepmd.pt.train.wrapper, 610
deepmd.pt.utils, 611
deepmd.pt.utils.ase_calc, 611
deepmd.pt.utils.auto_batch_size, 613
deepmd.pt.utils.cache, 614
deepmd.pt.utils.dataloader, 614
deepmd.pt.utils.dataset, 616
deepmd.pt.utils.dp_random, 617
deepmd.pt.utils.env, 618
deepmd.pt.utils.env_mat_stat, 618
deepmd.pt.utils.exclude_mask, 620
deepmd.pt.utils.finetune, 621
deepmd.pt.utils.learning_rate, 622
deepmd.pt.utils.multi_task, 623
deepmd.pt.utils.neighbor_stat, 624
deepmd.pt.utils.nlist, 626
deepmd.pt.utils.plugin, 628
deepmd.pt.utils.preprocess, 630
deepmd.pt.utils.region, 631
deepmd.pt.utils.serialization, 633
deepmd.pt.utils.stat, 634

deepmd.pt.utils.update_sel, 636
deepmd.pt.utils.utils, 636
deepmd.tf, 639
deepmd.tf.__about__, 994
deepmd.tf.__main__, 994
deepmd.tf.calculator, 994
deepmd.tf.cluster, 639
deepmd.tf.cluster.local, 639
deepmd.tf.common, 996
deepmd.tf.descriptor, 640
deepmd.tf.descriptor.descriptor, 640
deepmd.tf.descriptor.hybrid, 647
deepmd.tf.descriptor.loc_frame, 653
deepmd.tf.descriptor.se, 656
deepmd.tf.descriptor.se_a, 659
deepmd.tf.descriptor.se_a_ebd, 665
deepmd.tf.descriptor.se_a_ebd_v2, 667
deepmd.tf.descriptor.se_a_ef, 668
deepmd.tf.descriptor.se_a_mask, 673
deepmd.tf.descriptor.se_atten, 676
deepmd.tf.descriptor.se_atten_v2, 688
deepmd.tf.descriptor.se_r, 689
deepmd.tf.descriptor.se_t, 694
deepmd.tf.entrypoints, 746
deepmd.tf.entrypoints.compress, 746
deepmd.tf.entrypoints.convert, 747
deepmd.tf.entrypoints.doc, 747
deepmd.tf.entrypoints.freeze, 747
deepmd.tf.entrypoints.gui, 748
deepmd.tf.entrypoints.ipi, 749
deepmd.tf.entrypoints.main, 750
deepmd.tf.entrypoints.neighbor_stat, 751
deepmd.tf.entrypoints.test, 752
deepmd.tf.entrypoints.train, 753
deepmd.tf.entrypoints.transfer, 754
deepmd.tf.env, 1001
deepmd.tf.env.op_grads_module, 1122
deepmd.tf.env.op_module, 1069
deepmd.tf.fit, 760
deepmd.tf.fit.dipole, 760
deepmd.tf.fit.dos, 763
deepmd.tf.fit.ener, 766
deepmd.tf.fit.fitting, 771
deepmd.tf.fit.polar, 773
deepmd.tf.infer, 794
deepmd.tf.infer.data_modifier, 794
deepmd.tf.infer.deep_dipole, 795
deepmd.tf.infer.deep_dos, 796
deepmd.tf.infer.deep_eval, 798
deepmd.tf.infer.deep_polar, 808
deepmd.tf.infer.deep_pot, 810
deepmd.tf.infer.deep_tensor, 812
deepmd.tf.infer.deep_wfc, 814
deepmd.tf.infer.ewald_recp, 815
deepmd.tf.infer.model_devi, 816
deepmd.tf.lmp, 1004
deepmd.tf.loggers, 835
deepmd.tf.loggers.loggers, 835
deepmd.tf.loss, 837
deepmd.tf.loss.dos, 837
deepmd.tf.loss.ener, 838
deepmd.tf.loss.loss, 842
deepmd.tf.loss.tensor, 844
deepmd.tf.model, 850
deepmd.tf.model.dos, 850
deepmd.tf.model.ener, 853
deepmd.tf.model.frozen, 856
deepmd.tf.model.linear, 858
deepmd.tf.model.model, 860
deepmd.tf.model.model_stat, 868
deepmd.tf.model.multi, 869
deepmd.tf.model.pairtab, 872
deepmd.tf.model.pairwise_dprc, 874
deepmd.tf.model.tensor, 877
deepmd.tf.nvnmd, 898
deepmd.tf.nvnmd.data, 898
deepmd.tf.nvnmd.data.data, 898
deepmd.tf.nvnmd.descriptor, 899
deepmd.tf.nvnmd.descriptor.se_a, 899
deepmd.tf.nvnmd.descriptor.se_atten, 900
deepmd.tf.nvnmd.entrypoints, 901
deepmd.tf.nvnmd.entrypoints.freeze, 901
deepmd.tf.nvnmd.entrypoints.mapt, 902
deepmd.tf.nvnmd.entrypoints.train, 904
deepmd.tf.nvnmd.entrypoints.wrap, 905
deepmd.tf.nvnmd.fit, 910
deepmd.tf.nvnmd.fit.ener, 910
deepmd.tf.nvnmd.utils, 910
deepmd.tf.nvnmd.utils.argcheck, 910
deepmd.tf.nvnmd.utils.config, 911
deepmd.tf.nvnmd.utils.encode, 913
deepmd.tf.nvnmd.utils.fio, 915
deepmd.tf.nvnmd.utils.network, 916
deepmd.tf.nvnmd.utils.op, 918
deepmd.tf.nvnmd.utils.weight, 918
deepmd.tf.op, 923
deepmd.tf.op._add_flt_nvnmd_grad, 924
deepmd.tf.op._copy_flt_nvnmd_grad, 924
deepmd.tf.op._dotmul_flt_nvnmd_grad, 924
deepmd.tf.op._flt_nvnmd_grad, 925
deepmd.tf.op._gelu, 925
deepmd.tf.op._map_flt_nvnmd_grad, 926
deepmd.tf.op._matmul_fitnet_nvnmd_grad, 926
deepmd.tf.op._matmul_flt2fix_nvnmd, 926
deepmd.tf.op._matmul_flt_nvnmd_grad, 926
deepmd.tf.op._mul_flt_nvnmd_grad, 927
deepmd.tf.op._prod_force_grad, 927
deepmd.tf.op._prod_force_se_a_grad, 927

deepmd.tf.op._prod_force_se_a_mask_grad, 928
deepmd.tf.op._prod_force_se_r_grad, 928
deepmd.tf.op._prod_virial_grad, 928
deepmd.tf.op._prod_virial_se_a_grad, 929
deepmd.tf.op._prod_virial_se_r_grad, 929
deepmd.tf.op._quantize_nvnmd_grad, 929
deepmd.tf.op._soft_min_force_grad, 930
deepmd.tf.op._soft_min_virial_grad, 930
deepmd.tf.op._tabulate_grad, 930
deepmd.tf.op._tanh4_flt_nvnmd_grad, 931
deepmd.tf.train, 932
deepmd.tf.train.run_options, 932
deepmd.tf.train.trainer, 934
deepmd.tf.utils, 936
deepmd.tf.utils.argcheck, 936
deepmd.tf.utils.batch_size, 937
deepmd.tf.utils.compat, 938
deepmd.tf.utils.compress, 939
deepmd.tf.utils.convert, 940
deepmd.tf.utils.data, 943
deepmd.tf.utils.data_system, 947
deepmd.tf.utils.errors, 950
deepmd.tf.utils.finetune, 951
deepmd.tf.utils.graph, 952
deepmd.tf.utils.learning_rate, 958
deepmd.tf.utils.multi_init, 959
deepmd.tf.utils.neighbor_stat, 960
deepmd.tf.utils.network, 962
deepmd.tf.utils.nlist, 965
deepmd.tf.utils.pair_tab, 965
deepmd.tf.utils.parallel_op, 967
deepmd.tf.utils.path, 968
deepmd.tf.utils.plugin, 974
deepmd.tf.utils.random, 975
deepmd.tf.utils.region, 977
deepmd.tf.utils.serialization, 977
deepmd.tf.utils.sess, 978
deepmd.tf.utils.spin, 978
deepmd.tf.utils.tabulate, 979
deepmd.tf.utils.type_embed, 981
deepmd.tf.utils.update_sel, 984
deepmd.tf.utils.weight_avg, 984
deepmd.utils, 1013
deepmd.utils.argcheck, 1013
deepmd.utils.argcheck_nvnmd, 1019
deepmd.utils.batch_size, 1019
deepmd.utils.compat, 1022
deepmd.utils.data, 1025
deepmd.utils.data_system, 1029
deepmd.utils.econf_embd, 1034
deepmd.utils.env_mat_stat, 1036
deepmd.utils.errors, 1038
deepmd.utils.finetune, 1038
deepmd.utils.hostlist, 1039
deepmd.utils.model_stat, 1040
deepmd.utils.neighbor_stat, 1041
deepmd.utils.out_stat, 1042
deepmd.utils.pair_tab, 1043
deepmd.utils.path, 1045
deepmd.utils.plugin, 1050
deepmd.utils.random, 1052
deepmd.utils.spin, 1054
deepmd.utils.summary, 1055
deepmd.utils.update_sel, 1056
deepmd.utils.version, 1057
deepmd.utils.weight_avg, 1057

INDEX

Symbols

_AddFltNvnmdGrad() (in module deepmd.tf.op._add_flt_nvnmd_grad), 924
_CpoyFltNvnmdGrad() (in module deepmd.tf.op._copy_flt_nvnmd_grad), 924
_DP_DeepPotCompute (C++ function), 1207
_DP_DeepPotCompute<double> (C++ function), 1207
_DP_DeepPotCompute<float> (C++ function), 1207
_DP_DeepPotComputeMixedType (C++ function), 1207
_DP_DeepPotComputeMixedType<double> (C++ function), 1208
_DP_DeepPotComputeMixedType<float> (C++ function), 1208
_DP_DeepPotComputeNList (C++ function), 1208
_DP_DeepPotComputeNList<double> (C++ function), 1209
_DP_DeepPotComputeNList<float> (C++ function), 1209
_DP_DeepPotModelDeviCompute (C++ function), 1209
_DP_DeepPotModelDeviCompute<double> (C++ function), 1210
_DP_DeepPotModelDeviCompute<float> (C++ function), 1210
_DP_DeepPotModelDeviComputeNList (C++ function), 1210
_DP_DeepPotModelDeviComputeNList<double> (C++ function), 1211
_DP_DeepPotModelDeviComputeNList<float> (C++ function), 1211
_DP_DeepTensorCompute (C++ function), 1211
_DP_DeepTensorCompute<double> (C++ function), 1212
_DP_DeepTensorCompute<float> (C++ function), 1212
_DP_DeepTensorComputeNList (C++ function), 1212
_DP_DeepTensorComputeNList<double> (C++ function), 1213
_DP_DeepTensorComputeNList<float> (C++ function), 1213
_DP_DeepTensorComputeTensor (C++ function), 1213
_DP_DeepTensorComputeTensor<double> (C++ function), 1213
_DP_DeepTensorComputeTensor<float> (C++ function), 1214
_DP_DeepTensorComputeTensorNList (C++ function), 1214
_DP_DeepTensorComputeTensorNList<double> (C++ function), 1214
_DP_DeepTensorComputeTensorNList<float> (C++ function), 1215
_DP_DipoleChargeModifierComputeNList (C++ function), 1215
_DP_DipoleChargeModifierComputeNList<double> (C++ function), 1215
_DP_DipoleChargeModifierComputeNList<float> (C++ function), 1216
_DP_Get_Energy_Pointer (C++ function), 1216
_DotmulFltNvnmdGrad() (in module deepmd.tf.op._dotmul_flt_nvnmd_grad), 924
_FltNvnmdGrad() (in module deepmd.tf.op._flt_nvnmd_grad), 925
_HVD (deepmd.tf.train.run_options.RunOptions attribute), 933
_MapFltNvnmdGrad() (in module deepmd.tf.op._map_flt_nvnmd_grad), 926
_MatmulFitnetNvnmdGrad() (in module deepmd.tf.op._matmul_fitnet_nvnmd_grad), 926
_MatmulFlt2fixNvnmdGrad() (in module deepmd.tf.op._matmul_flt2fix_nvnmd), 926
_MatmulFltNvnmdGrad() (in module deepmd.tf.op._matmul_flt_nvnmd_grad), 926
_MulFltNvnmdGrad() (in module

deepmd.tf.op._mul_flt_nvnmd_grad), 927
_NONE (deepmd.dpmodel.output_def.OutputVariableOperation attribute), 376
_OUTDEF_DP2BACKEND (deepmd.infer.deep_eval.DeepEvalBackend attribute), 390
_QuantizeNvnmdGrad() (in module deepmd.tf.op._quantize_nvnmd_grad), 929
_SEC_DERV_R (deepmd.dpmodel.output_def.OutputVariableOperation attribute), 376
_Tanh4FltNvnmdGrad() (in module deepmd.tf.op._tanh4_flt_nvnmd_grad), 931
__add__() (deepmd.pt.utils.plugin.Plugin method), 629
__add__() (deepmd.tf.utils.Plugin method), 993
__add__() (deepmd.tf.utils.plugin.Plugin method), 974
__add__() (deepmd.utils.env_mat_stat.StatItem method), 1036
__add__() (deepmd.utils.plugin.Plugin method), 1051
__call__() (deepmd.dpmodel.NativeOP method), 380
__call__() (deepmd.dpmodel.common.NativeOP method), 374
__call__() (deepmd.dpmodel.utils.serialization.Counter method), 364
__call__() (deepmd.main.BackendOption method), 1065
__call__() (deepmd.main.DeprecateAction method), 1066
__call__() (deepmd.pt.utils.env_mat_stat.EnvMatStatSe method), 619
__call__() (deepmd.pt.utils.plugin.VariantMeta method), 629
__call__() (deepmd.tf.utils.plugin.VariantMeta method), 975
__call__() (deepmd.utils.plugin.VariantMeta method), 1052
__call__() (deepmd.utils.summary.SummaryPrinter method), 1056
__constants__ (deepmd.pt.model.descriptor.DescrptBlockSeA attribute), 522
__constants__ (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA attribute), 494
__contains__() (deepmd.dpmodel.utils.PairExcludeMask method), 368
__contains__() (deepmd.dpmodel.utils.exclude_mask.PairExcludeMask method), 354
__eq__() (deepmd.tf.utils.path.DPPath method), 973
__eq__() (deepmd.utils.path.DPPath method), 1046
__file_new_keys (deepmd.tf.utils.path.DPH5Path attribute), 969
__file_new_keys (deepmd.utils.path.DPH5Path attribute), 1049
__get_blocksize() (deepmd.pt.optimizer.LKF.LKFOptimizer method), 607
__get_blocksize() (deepmd.pt.optimizer.LKFOptimizer method), 608
__get_nue() (deepmd.pt.optimizer.LKF.LKFOptimizer method), 607
__get_nue() (deepmd.pt.optimizer.LKFOptimizer method), 608
__getattr__() (deepmd.dpmodel.model.SpinModel method), 349
__getattr__() (deepmd.dpmodel.model.spin_model.SpinModel method), 345
__getattr__() (deepmd.pt.model.model.SpinModel method), 548
__getattr__() (deepmd.pt.model.model.spin_model.SpinModel method), 539
__getitem__() (deepmd.dpmodel.FittingOutputDef method), 381
__getitem__() (deepmd.dpmodel.ModelOutputDef method), 381
__getitem__() (deepmd.dpmodel.atomic_model.BaseAtomicModel method), 272
__getitem__() (deepmd.dpmodel.atomic_model.base_atomic_model method), 261
__getitem__() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 301
__getitem__() (deepmd.dpmodel.descriptor.DescrptSeA method), 306
__getitem__() (deepmd.dpmodel.descriptor.DescrptSeR method), 308
__getitem__() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 method), 285
__getitem__() (deepmd.dpmodel.descriptor.dpa1.NeighborGatedAtt method), 286
__getitem__() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA method), 292
__getitem__() (deepmd.dpmodel.descriptor.se_r.DescrptSeR method), 295
__getitem__() (deepmd.dpmodel.fitting.PolarFitting method), 337
__getitem__() (deepmd.dpmodel.fitting.general_fitting.GeneralFitti method), 319
__getitem__() (deepmd.dpmodel.fitting.polarizability_fitting.PolarF method), 326
__getitem__() (deepmd.dpmodel.output_def.FittingOutputDef method), 377
__getitem__() (deepmd.dpmodel.output_def.ModelOutputDef method), 378
__getitem__() (deepmd.dpmodel.utils.NativeLayer method), 369

__getitem__() (deepmd.dpmodel.utils.NetworkCollection method), 370
__getitem__() (deepmd.dpmodel.utils.network.LayerNorm method), 358
__getitem__() (deepmd.dpmodel.utils.network.NativeLayer method), 357
__getitem__() (deepmd.dpmodel.utils.network.NetworkCollection method), 359
__getitem__() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 452
__getitem__() (deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel method), 435
__getitem__() (deepmd.pt.model.descriptor.DescrptBlockSeA method), 523
__getitem__() (deepmd.pt.model.descriptor.DescrptBlockSeAtten method), 508
__getitem__() (deepmd.pt.model.descriptor.DescrptSeR method), 527
__getitem__() (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA method), 495
__getitem__() (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtten method), 497
__getitem__() (deepmd.pt.model.descriptor.se_atten.NeighborGatedAttention method), 499
__getitem__() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 503
__getitem__() (deepmd.pt.model.task.PolarFittingNet method), 604
__getitem__() (deepmd.pt.model.task.fitting.GeneralFitting method), 591
__getitem__() (deepmd.pt.model.task.polarizability.PolarFittingNet method), 595
__getitem__() (deepmd.pt.utils.dataloader.DpLoaderSet method), 615
__getitem__() (deepmd.pt.utils.dataset.DeepmdDataSetForLoader method), 616
__getitem__() (deepmd.utils.data.DataRequirementItem method), 1029
__hash__() (deepmd.tf.utils.path.DPPath method), 973
__hash__() (deepmd.utils.path.DPPath method), 1046
__init_P() (deepmd.pt.optimizer.LKF.LKFOptimizer method), 607
__init_P() (deepmd.pt.optimizer.LKFOptimizer method), 608
__iter__() (deepmd.pt.utils.dataloader.BufferedIterator method), 616
__len__() (deepmd.pt.utils.dataloader.BufferedIterator method), 616
__len__() (deepmd.pt.utils.dataloader.DpLoaderSet method), 615
__len__() (deepmd.pt.utils.dataset.DeepmdDataSetForLoader method), 616
__lt__() (deepmd.tf.utils.path.DPH5Path method), 970
__lt__() (deepmd.tf.utils.path.DPOSPath method), 972
__lt__() (deepmd.tf.utils.path.DPPath method), 973
__lt__() (deepmd.utils.path.DPH5Path method), 1050
__lt__() (deepmd.utils.path.DPOSPath method), 1048
__lt__() (deepmd.utils.path.DPPath method), 1046
__next__() (deepmd.pt.utils.dataloader.BufferedIterator method), 616
__repr__() (deepmd.tf.utils.path.DPPath method), 973
__repr__() (deepmd.utils.path.DPPath method), 1046
__sample() (deepmd.pt.optimizer.KFOptimizerWrapper method), 608
__sample() (deepmd.pt.optimizer.KFWrapper.KFOptimizerWrapper method), 606
__setitem__() (deepmd.dpmodel.atomic_model.BaseAtomicModel method), 372
__setitem__() (deepmd.dpmodel.atomic_model.base_atomic_model method), 261
__setitem__() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 301
__setitem__() (deepmd.dpmodel.descriptor.DescrptSeA method), 306
__setitem__() (deepmd.dpmodel.descriptor.DescrptSeR method), 308
__setitem__() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 method), 285
__setitem__() (deepmd.dpmodel.descriptor.dpa1.NeighborGatedAtt method), 286
__setitem__() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA method), 292
__setitem__() (deepmd.dpmodel.descriptor.se_r.DescrptSeR method), 295
__setitem__() (deepmd.dpmodel.fitting.PolarFitting method), 337
__setitem__() (deepmd.dpmodel.fitting.general_fitting.GeneralFitti method), 319
__setitem__() (deepmd.dpmodel.fitting.polarizability_fitting.PolarF method), 326
__setitem__() (deepmd.dpmodel.utils.NativeLayer method), 369
__setitem__() (deepmd.dpmodel.utils.NetworkCollection method), 370
__setitem__() (deepmd.dpmodel.utils.network.LayerNorm method), 358
__setitem__() (deepmd.dpmodel.utils.network.NativeLayer method), 357
__setitem__() (deepmd.dpmodel.utils.network.NetworkCollection

method), 359
__setitem__() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 451
__setitem__() (deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel method), 435
__setitem__() (deepmd.pt.model.descriptor.DescrptBlockSeA method), 523
__setitem__() (deepmd.pt.model.descriptor.DescrptBlockSeAtten method), 507
__setitem__() (deepmd.pt.model.descriptor.DescrptSeR method), 527
__setitem__() (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA method), 495
__setitem__() (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtten method), 497
__setitem__() (deepmd.pt.model.descriptor.se_atten.NeighborGatedAttention method), 499
__setitem__() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 503
__setitem__() (deepmd.pt.model.task.PolarFittingNet method), 604
__setitem__() (deepmd.pt.model.task.fitting.GeneralFitting method), 590
__setitem__() (deepmd.pt.model.task.polarizability.PolarFittingNet method), 595
__split_weights() (deepmd.pt.optimizer.LKF.LKFOptimizer method), 607
__split_weights() (deepmd.pt.optimizer.LKFOptimizer method), 608
__str__() (deepmd.tf.utils.path.DPH5Path method), 970
__str__() (deepmd.tf.utils.path.DPOSPath method), 972
__str__() (deepmd.tf.utils.path.DPPath method), 973
__str__() (deepmd.utils.path.DPH5Path method), 1050
__str__() (deepmd.utils.path.DPOSPath method), 1048
__str__() (deepmd.utils.path.DPPath method), 1046
__truediv__() (deepmd.tf.utils.path.DPH5Path method), 970
__truediv__() (deepmd.tf.utils.path.DPOSPath method), 972
__truediv__() (deepmd.tf.utils.path.DPPath method), 973
__truediv__() (deepmd.utils.path.DPH5Path method), 1050
__truediv__() (deepmd.utils.path.DPOSPath method), 1048
__truediv__() (deepmd.utils.path.DPPath method), 1046
__update() (deepmd.pt.optimizer.LKF.LKFOptimizer method), 607
__update() (deepmd.pt.optimizer.LKFOptimizer method), 608
__version__ (in module deepmd.dpmodel.descriptor.dpa1), 281
__version__ (in module deepmd.dpmodel.descriptor.se_e2_a), 291
__version__ (in module deepmd.dpmodel.descriptor.se_r), 294
__version__ (in module deepmd.dpmodel.utils.network), 356
__version__ (in module deepmd.dpmodel.utils.serialization), 363
__version__ (in module deepmd.main), 1064
__version__ (in module deepmd.pt.model.network.mlp), 551
__version__ (in module deepmd.tf.__about__), 994
_adjust_batch_size() (deepmd.utils.batch_size.AutoBatchSize method), 1021
_all_excluded() (deepmd.tf.utils.tabulate.DPTabulate method), 981
_apply_bn() (deepmd.pt.model.descriptor.repformer_layer.Repform method), 488
_apply_bn_comp() (deepmd.pt.model.descriptor.repformer_layer.Rep method), 488
_apply_bn_uni() (deepmd.pt.model.descriptor.repformer_layer.Repf method), 488
_apply_h_norm() (in module deepmd.pt.model.descriptor.repformer_layer), 484
_apply_nb_1() (deepmd.pt.model.descriptor.repformer_layer.Repfor method), 488
_apply_nb_2() (deepmd.pt.model.descriptor.repformer_layer.Repfor method), 488
_apply_nlist_mask() (in module deepmd.pt.model.descriptor.repformer_layer), 484
_apply_switch() (in module deepmd.pt.model.descriptor.repformer_layer), 484
_attention_layers() (deepmd.tf.descriptor.DescrptSeAtten method), 733
_attention_layers() (deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 681
_bn_layer() (deepmd.pt.model.descriptor.repformer_layer.Repforme method), 489
_build_fv_graph_inner() (deepmd.tf.DipoleChargeModifier method), 1012
_build_fv_graph_inner()

(deepmd.tf.infer.DipoleChargeModifier method), 821
_build_fv_graph_inner() (deepmd.tf.infer.data_modifier.DipoleChargeModifier method), 794
_build_loss() (deepmd.tf.train.trainer.DPTrainer method), 935
_build_lower() (deepmd.tf.fit.DOSFitting method), 782
_build_lower() (deepmd.tf.fit.DipoleFittingSeA method), 779
_build_lower() (deepmd.tf.fit.EnerFitting method), 786
_build_lower() (deepmd.tf.fit.PolarFittingSeA method), 792
_build_lower() (deepmd.tf.fit.dipole.DipoleFittingSeA method), 761
_build_lower() (deepmd.tf.fit.dos.DOSFitting method), 765
_build_lower() (deepmd.tf.fit.ener.EnerFitting method), 769
_build_lower() (deepmd.tf.fit.polar.PolarFittingSeA method), 774
_build_lower() (deepmd.tf.utils.tabulate.DPTabulate method), 981
_build_lr() (deepmd.tf.train.trainer.DPTrainer method), 935
_build_network() (deepmd.tf.train.trainer.DPTrainer method), 935
_build_optimizer() (deepmd.tf.train.trainer.DPTrainer method), 935
_build_training() (deepmd.tf.train.trainer.DPTrainer method), 935
_cal_grrg() (deepmd.pt.model.descriptor.repformer_layer.RepformerLayer method), 488
_cal_h2g2() (deepmd.pt.model.descriptor.repformer_layer.RepformerLayer method), 488
_calculate_ener() (deepmd.dpmodel.atomic_model.PairTabAtomicModel static method), 280
_calculate_ener() (deepmd.dpmodel.atomic_model.pairtab_atomic_model.PairTabAtomicModel static method), 271
_calculate_ener() (deepmd.pt.model.atomic_model.PairTabAtomicModel static method), 463
_calculate_ener() (deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel static method), 449
_call() (deepmd.dpmodel.utils.EnvMat method), 367
_call() (deepmd.dpmodel.utils.env_mat.EnvMat method), 353
_call_common() (deepmd.dpmodel.fitting.general_fitting.GeneralFitting method), 320
_change_energy_bias() (deepmd.tf.train.trainer.DPTrainer method), 935
_change_sub_config() (in module deepmd.tf.utils.multi_init), 960
_check_mixed_types() (deepmd.infer.deep_eval.DeepEvalBackend method), 392
_check_mode() (deepmd.tf.utils.DeepmdData method), 988
_check_mode() (deepmd.tf.utils.data.DeepmdData method), 947
_check_mode() (deepmd.utils.data.DeepmdData method), 1028
_check_pbc() (deepmd.tf.utils.DeepmdData method), 988
_check_pbc() (deepmd.tf.utils.data.DeepmdData method), 947
_check_pbc() (deepmd.utils.data.DeepmdData method), 1028
_check_shape_consistency() (deepmd.dpmodel.utils.network.LayerNorm method), 358
_check_table_upper_boundary() (deepmd.tf.utils.PairTab method), 992
_check_table_upper_boundary() (deepmd.tf.utils.pair_tab.PairTab method), 966
_check_table_upper_boundary() (deepmd.utils.pair_tab.PairTab method), 1044
_check_tmaps() (in module deepmd.infer.model_devi), 404
_check_type_map_consistency() (deepmd.tf.utils.DeepmdDataSystem method), 991
_check_type_map_consistency() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 950
_check_type_map_consistency() (deepmd.utils.data_system.DeepmdDataSystem method), 1033
_compute_dstats_sys_nonsmth() (deepmd.dpmodel.descriptor.DescrptLocFrame method), 713
_compute_dstats_sys_nonsmth() (deepmd.tf.descriptor.DescrptLocFrame method), 655
_compute_dstats_sys_se_r() (deepmd.tf.descriptor.DescrptSeR method), 741
_compute_dstats_sys_se_r() (deepmd.tf.descriptor.se_r.DescrptSeR method), 693
_compute_dstats_sys_smth() (deepmd.tf.descriptor.DescrptSeA method), 718

_compute_dstats_sys_smth() (deepmd.tf.descriptor.DescrptSeAEfLower method), 726
_compute_dstats_sys_smth() (deepmd.tf.descriptor.DescrptSeAtten method), 733
_compute_dstats_sys_smth() (deepmd.tf.descriptor.DescrptSeT method), 745
_compute_dstats_sys_smth() (deepmd.tf.descriptor.se_a.DescrptSeA method), 664
_compute_dstats_sys_smth() (deepmd.tf.descriptor.se_a_ef.DescrptSeAEfLower method), 673
_compute_dstats_sys_smth() (deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 680
_compute_dstats_sys_smth() (deepmd.tf.descriptor.se_t.DescrptSeT method), 698
_compute_input_stat() (deepmd.tf.model.DOSModel method), 889
_compute_input_stat() (deepmd.tf.model.EnerModel method), 892
_compute_input_stat() (deepmd.tf.model.MultiModel method), 894
_compute_input_stat() (deepmd.tf.model.dos.DOSModel method), 851
_compute_input_stat() (deepmd.tf.model.ener.EnerModel method), 854
_compute_input_stat() (deepmd.tf.model.multi.MultiModel method), 870
_compute_input_stat() (deepmd.tf.model.tensor.TensorModel method), 878
_compute_model_predict() (in module deepmd.pt.utils.stat), 635
_compute_output_stat() (deepmd.tf.model.DOSModel method), 889
_compute_output_stat() (deepmd.tf.model.EnerModel method), 892
_compute_output_stat() (deepmd.tf.model.MultiModel method), 894
_compute_output_stat() (deepmd.tf.model.dos.DOSModel method), 851
_compute_output_stat() (deepmd.tf.model.ener.EnerModel method), 854
_compute_output_stat() (deepmd.tf.model.multi.MultiModel method), 870
_compute_output_stat() (deepmd.tf.model.tensor.TensorModel method), 878
_compute_output_stats() (deepmd.tf.fit.DOSFitting method), 782
_compute_output_stats() (deepmd.tf.fit.EnerFitting method), 785
_compute_output_stats() (deepmd.tf.fit.dos.DOSFitting method), 764
_compute_output_stats() (deepmd.tf.fit.ener.EnerFitting method), 769
_compute_std() (deepmd.tf.descriptor.DescrptLocFrame method), 713
_compute_std() (deepmd.tf.descriptor.DescrptSeA method), 718
_compute_std() (deepmd.tf.descriptor.DescrptSeR method), 741
_compute_std() (deepmd.tf.descriptor.DescrptSeT method), 745
_compute_std() (deepmd.tf.descriptor.loc_frame.DescrptLocFrame method), 655
_compute_std() (deepmd.tf.descriptor.se_a.DescrptSeA method), 664
_compute_std() (deepmd.tf.descriptor.se_r.DescrptSeR method), 693
_compute_std() (deepmd.tf.descriptor.se_t.DescrptSeT method), 698
_compute_std() (deepmd.tf.fit.DOSFitting method), 782
_compute_std() (deepmd.tf.fit.EnerFitting method), 785
_compute_std() (deepmd.tf.fit.dos.DOSFitting method), 765
_compute_std() (deepmd.tf.fit.ener.EnerFitting method), 769
_compute_weight() (deepmd.dpmodel.atomic_model.DPZBLLinearE method), 275
_compute_weight() (deepmd.dpmodel.atomic_model.LinearEnergyA method), 277
_compute_weight() (deepmd.dpmodel.atomic_model.linear_atomic_ method), 267
_compute_weight() (deepmd.dpmodel.atomic_model.linear_atomic_ method), 266
_compute_weight() (deepmd.pt.model.atomic_model.DPZBLLinearE

method), 458
_compute_weight() (deepmd.pt.model.atomic_model.LinearEnergyAtomicModel method), 460
_compute_weight() (deepmd.pt.model.atomic_model.linear_atomic_model.DPZBLLinearEnergyAtomicModel method), 446
_compute_weight() (deepmd.pt.model.atomic_model.linear_atomic_model.LinearEnergyAtomicModel method), 444
_concat_type_embedding() (deepmd.tf.descriptor.DescrptSeA method), 718
_concat_type_embedding() (deepmd.tf.descriptor.se_a.DescrptSeA method), 664
_connect_path() (deepmd.tf.utils.path.DPH5Path method), 970
_connect_path() (deepmd.utils.path.DPH5Path method), 1050
_convert_key() (deepmd.dpmodel.utils.NetworkCollection method), 370
_convert_key() (deepmd.dpmodel.utils.network.NetworkCollection method), 359
_convert_numpy_to_tensor() (deepmd.tf.utils.tabulate.DPTabulate method), 981
_create_consumer() (deepmd.pt.utils.dataloader.BufferedIterator method), 616
_default_bias() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 454
_default_bias() (deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel method), 438
_default_std() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 454
_default_std() (deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel method), 438
_ebd_filter() (deepmd.tf.descriptor.DescrptSeAEbd method), 721
_ebd_filter() (deepmd.tf.descriptor.se_a_ebd.DescrptSeAEbd method), 667
_embedding_net() (deepmd.tf.descriptor.DescrptSeAEbd method), 721
_embedding_net() (deepmd.tf.descriptor.se_a_ebd.DescrptSeAEbd method), 667
_enrich() (deepmd.tf.DipoleChargeModifier method), 1012
_enrich() (deepmd.tf.infer.DipoleChargeModifier method), 821
_enrich() (deepmd.tf.infer.data_modifier.DipoleChargeModifier method), 794
_eval_descriptor_inner() (deepmd.tf.DeepEval method), 1011
_eval_descriptor_inner() (deepmd.tf.infer.DeepEval method), 829
_eval_descriptor_inner() (deepmd.tf.infer.deep_eval.DeepEval method), 804
_eval_energy() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 340
_eval_func() (deepmd.pt.infer.deep_eval.DeepEval method), 418
_eval_func() (deepmd.tf.DeepEval method), 1009
_eval_func() (deepmd.tf.infer.DeepEval method), 828
_eval_func() (deepmd.tf.infer.deep_eval.DeepEval method), 802
_eval_fv() (deepmd.tf.DipoleChargeModifier method), 1012
_eval_fv() (deepmd.tf.infer.DipoleChargeModifier method), 821
_eval_fv() (deepmd.tf.infer.data_modifier.DipoleChargeModifier method), 795
_eval_inner() (deepmd.tf.DeepEval method), 1010
_eval_inner() (deepmd.tf.infer.DeepEval method), 829
_eval_inner() (deepmd.tf.infer.deep_eval.DeepEval method), 803
_eval_model() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 341
_eval_model() (deepmd.pt.infer.deep_eval.DeepEval method), 418
_eval_model_spin() (deepmd.pt.infer.deep_eval.DeepEval method), 418
_execute() (deepmd.pt.utils.neighbor_stat.NeighborStat method), 686
_execute() (deepmd.tf.utils.neighbor_stat.NeighborStat method), 962
_expande_atype() (deepmd.infer.DeepEval method), 407
_expande_atype() (deepmd.infer.deep_eval.DeepEval method), 393
_extend_a_avg_std() (deepmd.pt.model.task.fitting.GeneralFitting method), 591
_extend_f_avg_std() (deepmd.pt.model.task.fitting.GeneralFitting method), 591
_extend_system() (deepmd.tf.DipoleChargeModifier method), 1012
_extend_system() (deepmd.tf.infer.DipoleChargeModifier method), 821
_extend_system() (deepmd.tf.infer.data_modifier.DipoleChargeMod method), 795
_extract_spline_coefficient() (deepmd.dpmodel.atomic_model.PairTabAtomicModel static method), 279
_extract_spline_coefficient() (deepmd.dpmodel.atomic_model.pairtab_atomic_model.Pai static method), 270
_extract_spline_coefficient()

(deepmd.pt.model.atomic_model.PairTabAtomicModel static method), 463
_extract_spline_coefficient() (deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel static method), 449
_extrapolate_table() (deepmd.tf.utils.PairTab method), 993
_extrapolate_table() (deepmd.tf.utils.pair_tab.PairTab method), 967
_extrapolate_table() (deepmd.utils.pair_tab.PairTab method), 1044
_fetch_out_stat() (deepmd.dpmodel.atomic_model.BaseAtomicModel method), 273
_fetch_out_stat() (deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicModel method), 262
_fetch_out_stat() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 454
_fetch_out_stat() (deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel method), 438
_file_keys() (deepmd.tf.utils.path.DPH5Path class method), 970
_file_keys() (deepmd.utils.path.DPH5Path class method), 1050
_fill_stat_with_global() (in module deepmd.pt.utils.stat), 635
_filter() (deepmd.tf.descriptor.DescrptSeA method), 719
_filter() (deepmd.tf.descriptor.DescrptSeAtten method), 733
_filter() (deepmd.tf.descriptor.DescrptSeT method), 746
_filter() (deepmd.tf.descriptor.se_a.DescrptSeA method), 664
_filter() (deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 681
_filter() (deepmd.tf.descriptor.se_t.DescrptSeT method), 698
_filter_lower() (deepmd.tf.descriptor.DescrptSeA method), 719
_filter_lower() (deepmd.tf.descriptor.DescrptSeAtten method), 733
_filter_lower() (deepmd.tf.descriptor.se_a.DescrptSeA method), 664
_filter_lower() (deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 681
_filter_r() (deepmd.tf.descriptor.DescrptSeR method), 741
_filter_r() (deepmd.tf.descriptor.se_r.DescrptSeR method), 693
_fitting_net() (in module deepmd.utils.compat), 1023
_format_name_length() (in module deepmd.utils.data_system), 1033
_forward_common() (deepmd.pt.model.task.fitting.GeneralFitting method), 591
_gelu() (in module deepmd.tf.op._gelu), 925
_gelu_grad_custom_cc() (in module deepmd.tf.op._gelu), 925
_get_bias() (deepmd.tf.utils.tabulate.DPTabulate method), 981
_get_bias_index() (deepmd.dpmodel.atomic_model.BaseAtomicModel method), 273
_get_bias_index() (deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicModel method), 262
_get_bias_index() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 454
_get_bias_index() (deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel method), 438
_get_data_type() (deepmd.tf.utils.tabulate.DPTabulate method), 981
_get_forward_wrapper_func() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 454
_get_forward_wrapper_func() (deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel method), 438
_get_last_layer_size() (deepmd.tf.utils.tabulate.DPTabulate method), 981
_get_layer_size() (deepmd.tf.utils.tabulate.DPTabulate method), 981
_get_matrix() (deepmd.tf.utils.tabulate.DPTabulate method), 981
_get_natoms_2() (deepmd.tf.utils.DeepmdData method), 987
_get_natoms_2() (deepmd.tf.utils.data.DeepmdData method), 946
_get_natoms_2() (deepmd.utils.data.DeepmdData method), 1028
_get_natoms_and_nframes() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 341
_get_natoms_and_nframes() (deepmd.infer.DeepEval method), 407
_get_natoms_and_nframes() (deepmd.infer.deep_eval.DeepEval method), 393
_get_natoms_and_nframes() (deepmd.pt.infer.deep_eval.DeepEval method), 418
_get_natoms_and_nframes() (deepmd.tf.DeepEval method), 1009

_get_natoms_and_nframes() (deepmd.tf.infer.DeepEval method), 828
_get_natoms_and_nframes() (deepmd.tf.infer.deep_eval.DeepEval method), 802
_get_nframes() (deepmd.tf.utils.DeepmdData method), 988
_get_nframes() (deepmd.tf.utils.data.DeepmdData method), 946
_get_nframes() (deepmd.utils.data.DeepmdData method), 1028
_get_output_shape() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 341
_get_output_shape() (deepmd.pt.infer.deep_eval.DeepEval method), 418
_get_output_shape() (deepmd.tf.DeepEval method), 1010
_get_output_shape() (deepmd.tf.infer.DeepEval method), 829
_get_output_shape() (deepmd.tf.infer.deep_eval.DeepEval method), 803
_get_pairwise_dist() (deepmd.dpmodel.atomic_model.PairTabAtomicModel static method), 279
_get_pairwise_dist() (deepmd.dpmodel.atomic_model.pairtab_atomic_model.PairTabAtomicModel static method), 270
_get_pairwise_dist() (deepmd.pt.model.atomic_model.PairTabAtomicModel static method), 463
_get_pairwise_dist() (deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel static method), 448
_get_place_holders() (deepmd.tf.train.trainer.DPTrainer method), 935
_get_request_defs() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 340
_get_request_defs() (deepmd.pt.infer.deep_eval.DeepEval method), 418
_get_sel_natoms() (deepmd.infer.DeepEval method), 408
_get_sel_natoms() (deepmd.infer.deep_eval.DeepEval method), 394
_get_subdata() (deepmd.tf.utils.DeepmdData method), 987
_get_subdata() (deepmd.tf.utils.data.DeepmdData method), 946
_get_subdata() (deepmd.utils.data.DeepmdData method), 1028
_get_table_size() (deepmd.tf.utils.tabulate.DPTabulate method), 981
_get_tensor() (deepmd.tf.DeepEval method), 1006
_get_tensor() (deepmd.tf.infer.DeepEval method), 825
_get_tensor() (deepmd.tf.infer.deep_eval.DeepEval method), 799
_get_tensor() (deepmd.tf.infer.deep_eval.DeepEvalOld method), 805
_glorot_uniform_init() (deepmd.pt.model.network.network.Linear method), 558
_graph_compatable() (deepmd.tf.DeepEval method), 1006
_graph_compatable() (deepmd.tf.infer.DeepEval method), 825
_graph_compatable() (deepmd.tf.infer.deep_eval.DeepEval method), 799
_graph_compatable() (deepmd.tf.infer.deep_eval.DeepEvalOld method), 805
_identity_tensors() (deepmd.tf.descriptor.se.DescrptSe method), 656
_idx_map_sel() (deepmd.tf.utils.DeepmdData method), 987
_idx_map_sel() (deepmd.tf.utils.data.DeepmdData method), 946
_idx_map_sel() (deepmd.utils.data.DeepmdData method), 1028
_import_graph_def_from_ckpt_meta() (deepmd.tf.model.model.Model method), 862
_import_graph_def_from_frz_model() (deepmd.tf.model.model.Model method), 864
_init_attr() (deepmd.tf.DeepEval method), 1006
_init_attr() (deepmd.tf.infer.DeepEval method), 824
_init_attr() (deepmd.tf.infer.deep_eval.DeepEval method), 799
_init_distributed() (deepmd.tf.train.run_options.RunOptions method), 934
_init_from_ckpt() (deepmd.tf.train.trainer.DPTrainer method), 935
_init_from_frz_model() (deepmd.tf.train.trainer.DPTrainer method), 935
_init_from_pretrained_model() (deepmd.tf.train.trainer.DPTrainer method), 935

_init_param() (deepmd.tf.train.trainer.DPTrainer method), 934
_init_serial() (deepmd.tf.train.run_options.RunOptions method), 934
_init_session() (deepmd.tf.train.trainer.DPTrainer method), 935
_init_tensors() (deepmd.tf.DeepEval method), 1006
_init_tensors() (deepmd.tf.infer.DeepEval method), 824
_init_tensors() (deepmd.tf.infer.deep_eval.DeepEval method), 799
_is_subdir() (in module deepmd.tf.train.trainer), 934
_jcopy() (in module deepmd.utils.compat), 1024
_keys (deepmd.tf.utils.path.DPH5Path property), 969
_keys (deepmd.utils.path.DPH5Path property), 1049
_layer_0() (deepmd.tf.utils.tabulate.DPTabulate method), 981
_layer_0() (in module deepmd.tf.utils.compress), 940
_layer_1() (deepmd.tf.utils.tabulate.DPTabulate method), 981
_layer_1() (in module deepmd.tf.utils.compress), 940
_learning_rate() (in module deepmd.utils.compat), 1023
_load_batch_set() (deepmd.tf.utils.DeepmdData method), 987
_load_batch_set() (deepmd.tf.utils.data.DeepmdData method), 946
_load_batch_set() (deepmd.utils.data.DeepmdData method), 1028
_load_data() (deepmd.tf.utils.DeepmdData method), 988
_load_data() (deepmd.tf.utils.data.DeepmdData method), 946
_load_data() (deepmd.utils.data.DeepmdData method), 1028
_load_graph() (deepmd.tf.DeepEval static method), 1006
_load_graph() (deepmd.tf.infer.DeepEval static method), 825
_load_graph() (deepmd.tf.infer.deep_eval.DeepEval static method), 799
_load_graph() (deepmd.tf.infer.deep_eval.DeepEvalOld static method), 805
_load_h5py() (deepmd.tf.utils.path.DPH5Path class method), 969
_load_h5py() (deepmd.utils.path.DPH5Path class method), 1049
_load_set() (deepmd.tf.utils.DeepmdData method), 988
_load_set() (deepmd.tf.utils.data.DeepmdData method), 946
_load_set() (deepmd.utils.data.DeepmdData method), 1028
_load_sub_graph() (deepmd.tf.utils.tabulate.DPTabulate method), 981
_load_test() (deepmd.tf.utils.DeepmdDataSystem method), 988
_load_test() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 947
_load_test() (deepmd.utils.data_system.DeepmdDataSystem method), 1030
_load_test_set() (deepmd.tf.utils.DeepmdData method), 988
_load_test_set() (deepmd.tf.utils.data.DeepmdData method), 946
_load_test_set() (deepmd.utils.data.DeepmdData method), 1028
_load_type() (deepmd.tf.utils.DeepmdData method), 988
_load_type() (deepmd.tf.utils.data.DeepmdData method), 947
_load_type() (deepmd.utils.data.DeepmdData method), 1028
_load_type_map() (deepmd.tf.utils.DeepmdData method), 988
_load_type_map() (deepmd.tf.utils.data.DeepmdData method), 947
_load_type_map() (deepmd.utils.data.DeepmdData method), 1028
_load_type_mix() (deepmd.tf.utils.DeepmdData method), 988
_load_type_mix() (deepmd.tf.utils.data.DeepmdData method), 947
_load_type_mix() (deepmd.utils.data.DeepmdData method), 1028
_log_handles_already_set (deepmd.tf.train.run_options.RunOptions attribute), 933
_lookup_type_embedding() (deepmd.tf.descriptor.DescrptSeAtten method), 733
_lookup_type_embedding() (deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 680
_loss() (in module deepmd.utils.compat), 1024
_make_all_stat_ref() (in module deepmd.tf.model.model_stat), 868
_make_all_stat_ref() (in module deepmd.utils.model_stat), 1040
_make_auto_bs() (deepmd.tf.utils.DeepmdDataSystem method), 991
_make_auto_bs() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 950

_make_auto_bs() (deepmd.utils.data_system.DeepmdDataSystem method), 1033
_make_auto_ts() (deepmd.tf.utils.DeepmdDataSystem method), 991
_make_auto_ts() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 950
_make_auto_ts() (deepmd.utils.data_system.DeepmdDataSystem method), 1033
_make_data() (deepmd.tf.utils.PairTab method), 993
_make_data() (deepmd.tf.utils.pair_tab.PairTab method), 967
_make_data() (deepmd.tf.utils.tabulate.DPTabulate method), 981
_make_data() (deepmd.utils.pair_tab.PairTab method), 1045
_make_env_mat() (in module deepmd.dpmodel.utils.env_mat), 352
_make_env_mat() (in module deepmd.pt.model.descriptor.env_mat), 479
_make_idx_map() (deepmd.tf.utils.DeepmdData method), 988
_make_idx_map() (deepmd.tf.utils.data.DeepmdData method), 947
_make_idx_map() (deepmd.utils.data.DeepmdData method), 1028
_make_nei_g1() (in module deepmd.pt.model.descriptor.repformer_layer), 484
_make_preset_out_bias() (in module deepmd.pt.utils.stat), 635
_merge_batch_data() (deepmd.tf.utils.DeepmdDataSystem method), 990
_merge_batch_data() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 949
_merge_batch_data() (deepmd.utils.data_system.DeepmdDataSystem method), 1032
_model() (in module deepmd.utils.compat), 1022
_model_change_out_bias() (in module deepmd.pt.train.training), 609
_n_all_excluded (deepmd.tf.utils.tabulate.DPTabulate property), 980
_net_out_dim() (deepmd.dpmodel.fitting.DipoleFitting method), 328
_net_out_dim() (deepmd.dpmodel.fitting.InvarFitting method), 335
_net_out_dim() (deepmd.dpmodel.fitting.PolarFitting method), 337
_net_out_dim() (deepmd.dpmodel.fitting.dipole_fitting method), 312
_net_out_dim() (deepmd.dpmodel.fitting.general_fitting.GeneralFitting method), 319
_net_out_dim() (deepmd.dpmodel.fitting.invar_fitting.InvarFitting method), 323
_net_out_dim() (deepmd.dpmodel.fitting.polarizability_fitting.PolarFitting method), 325
_net_out_dim() (deepmd.pt.model.task.DipoleFittingNet method), 599
_net_out_dim() (deepmd.pt.model.task.PolarFittingNet method), 604
_net_out_dim() (deepmd.pt.model.task.dipole.DipoleFittingNet method), 582
_net_out_dim() (deepmd.pt.model.task.fitting.GeneralFitting method), 591
_net_out_dim() (deepmd.pt.model.task.invar_fitting.InvarFitting method), 592
_net_out_dim() (deepmd.pt.model.task.polarizability.PolarFittingNet method), 595
_new_keys (deepmd.tf.utils.path.DPH5Path property), 969
_new_keys (deepmd.utils.path.DPH5Path property), 1049
_nonsmth_descriptor() (in module deepmd.utils.compat), 1023
_normal_init() (deepmd.pt.model.network.network.Embedding method), 561
_normal_init() (deepmd.pt.model.network.network.Linear method), 558
_normalize_3d() (deepmd.tf.descriptor.DescrptSeAEfLower method), 725
_normalize_3d() (deepmd.tf.descriptor.se_a_ef.DescrptSeAEfLower method), 672
_opm() (deepmd.pt.model.network.network.OuterProduct method), 572
_pair_tabulated_inter() (deepmd.dpmodel.atomic_model.PairTabAtomicModel method), 278
_pair_tabulated_inter() (deepmd.dpmodel.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 269
_pair_tabulated_inter() (deepmd.pt.model.atomic_model.PairTabAtomicModel method), 462
_pair_tabulated_inter() (deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 448
_pass_filter() (deepmd.tf.descriptor.DescrptSeA method), 718
_pass_filter() (deepmd.tf.descriptor.DescrptSeAEbd method), 721
_pass_filter() (deepmd.tf.descriptor.DescrptSeAtten method), 733
_pass_filter() (deepmd.tf.descriptor.DescrptSeR method), 741

_pass_filter() (deepmd.tf.descriptor.DescrptSeT method), 745
_pass_filter() (deepmd.tf.descriptor.se_a.DescrptSeA method), 664
_pass_filter() (deepmd.tf.descriptor.se_a_ebd.DescrptSeAEbd method), 667
_pass_filter() (deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 680
_pass_filter() (deepmd.tf.descriptor.se_r.DescrptSeR method), 693
_pass_filter() (deepmd.tf.descriptor.se_t.DescrptSeT method), 698
_post_process_stat() (in module deepmd.pt.utils.stat), 634
_prepare_feed_dict() (deepmd.tf.DeepEval method), 1010
_prepare_feed_dict() (deepmd.tf.infer.DeepEval method), 829
_prepare_feed_dict() (deepmd.tf.infer.deep_eval.DeepEval method), 803
_prod_force_grad_cc() (in module deepmd.tf.op._prod_force_grad), 927
_prod_force_se_a_grad_cc() (in module deepmd.tf.op._prod_force_se_a_grad), 927
_prod_force_se_a_grad_cc() (in module deepmd.tf.op._prod_force_se_r_grad), 928
_prod_force_se_a_mask_grad_cc() (in module deepmd.tf.op._prod_force_se_a_mask_grad), 928
_prod_virial_grad_cc() (in module deepmd.tf.op._prod_virial_grad), 928
_prod_virial_se_a_grad_cc() (in module deepmd.tf.op._prod_virial_se_a_grad), 929
_prod_virial_se_a_grad_cc() (in module deepmd.tf.op._prod_virial_se_r_grad), 929
_program() (in module deepmd.tf.entrypoints.ipi), 749
_restore_from_file() (in module deepmd.pt.utils.stat), 634
_run_default_sess() (deepmd.tf.infer.deep_tensor.DeepTensor method), 812
_save_to_file() (in module deepmd.pt.utils.stat), 634
_scaled_dot_attn() (deepmd.tf.descriptor.DescrptSeAtten method), 733
_scaled_dot_attn() (deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 681
_sentinel (in module deepmd.pt.utils.dataloader), 615
_set_pairtab() (deepmd.pt.model.atomic_model.PairTabAtomicMo method), 461
_set_pairtab() (deepmd.pt.model.atomic_model.pairtab_atomic_m method), 447
_setup_logger() (deepmd.tf.train.run_options.RunOptions method), 933
_shuffle_data() (deepmd.tf.utils.DeepmdData method), 988
_shuffle_data() (deepmd.tf.utils.data.DeepmdData method), 946
_shuffle_data() (deepmd.utils.data.DeepmdData method), 1028
_slice_descrpt_deriv() (deepmd.tf.DipoleChargeModifier method), 1012
_slice_descrpt_deriv() (deepmd.tf.infer.DipoleChargeModifier method), 821
_slice_descrpt_deriv() (deepmd.tf.infer.data_modifier.DipoleChargeModifier method), 794
_smth_descriptor() (in module deepmd.utils.compat), 1023
_soft_min_force_grad_cc() (in module deepmd.tf.op._soft_min_force_grad), 930
_soft_min_virial_grad_cc() (in module deepmd.tf.op._soft_min_virial_grad), 930
_sort_rcuts_sels() (deepmd.dpmodel.atomic_model.LinearEnergy method), 276
_sort_rcuts_sels() (deepmd.dpmodel.atomic_model.linear_atomic method), 265
_sort_rcuts_sels() (deepmd.pt.model.atomic_model.LinearEnergy method), 458
_sort_rcuts_sels() (deepmd.pt.model.atomic_model.linear_atomic method), 443
_spline5_switch() (deepmd.tf.utils.tabulate.DPTabulate method), 981
_standard_input() (deepmd.infer.DeepEval method), 408
_standard_input() (deepmd.infer.deep_eval.DeepEval method), 394
_store_out_stat() (deepmd.pt.model.atomic_model.BaseAtomicMo method), 454
_store_out_stat() (deepmd.pt.model.atomic_model.base_atomic_m method), 438
_tabulate_fusion_se_a_grad_cc() (in module deepmd.tf.op._tabulate_grad), 931
_tabulate_fusion_se_a_grad_grad_cc() (in module deepmd.tf.op._tabulate_grad), 931
_tabulate_fusion_se_atten_grad_cc() (in module deepmd.tf.op._tabulate_grad), 931

_tabulate_fusion_se_atten_grad_grad_cc() (in module deepmd.tf.op._tabulate_grad), 931
_tabulate_fusion_se_r_grad_cc() (in module deepmd.tf.op._tabulate_grad), 931
_tabulate_fusion_se_r_grad_grad_cc() (in module deepmd.tf.op._tabulate_grad), 931
_tabulate_fusion_se_t_grad_cc() (in module deepmd.tf.op._tabulate_grad), 931
_tabulate_fusion_se_t_grad_grad_cc() (in module deepmd.tf.op._tabulate_grad), 931
_to_face_distance() (in module deepmd.pt.utils.region), 632
_training() (in module deepmd.utils.compat), 1024
_transition() (deepmd.pt.model.network.network.TransitionLinear method), 559
_trunc_normal_init() (deepmd.pt.model.network.network.Linear method), 558
_try_init_distrib() (deepmd.tf.train.run_options.RunOptions method), 934
_type_embed() (deepmd.tf.descriptor.DescrptSeAEbd method), 721
_type_embed() (deepmd.tf.descriptor.se_a_ebd.DescrptSeAEbd method), 667
_type_embedding_net_one_side() (deepmd.tf.descriptor.DescrptSeAEbd method), 721
_type_embedding_net_one_side() (deepmd.tf.descriptor.se_a_ebd.DescrptSeAEbd method), 667
_type_embedding_net_one_side_aparam() (deepmd.tf.descriptor.DescrptSeAEbd method), 721
_type_embedding_net_one_side_aparam() (deepmd.tf.descriptor.se_a_ebd.DescrptSeAEbd method), 667
_type_embedding_net_two_sides() (deepmd.tf.descriptor.DescrptSeAEbd method), 721
_type_embedding_net_two_sides() (deepmd.tf.descriptor.se_a_ebd.DescrptSeAEbd method), 667
_update_g1_conv() (deepmd.pt.model.descriptor.repformer_layer.Repformer_layer method), 488
_update_g1_grrg() (deepmd.pt.model.descriptor.repformer_layer.Repformer_layer method), 488
_update_g2_g1g1() (deepmd.pt.model.descriptor.repformer_layer.Repformer_layer method), 488
_update_h2() (deepmd.pt.model.descriptor.repformer_layer.Repformer_layer method), 488
_varsize() (deepmd.dpmodel.atomic_model.BaseAtomicModel method), 273
_varsize() (deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicModel method), 262
_varsize() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 454
_varsize() (deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel method), 438
_warning_input_v0_v1() (in module deepmd.utils.compat), 1022
_warning_input_v1_v2() (in module deepmd.utils.compat), 1024
_zero_init() (deepmd.pt.model.network.network.Linear method), 558

A

ACTIVATION_FN_DICT (in module deepmd.tf.common), 1000
ACTIVATION_FN_DICT (in module deepmd.utils.argcheck), 1016
activation_function:
    model/type_embedding/activation_function (Argument), 112
    model[standard]/descriptor[se_a_ebd_v2]/activation_function (Argument), 136
    model[standard]/descriptor[se_a_mask]/activation_function (Argument), 138
    model[standard]/descriptor[se_a_tpe]/activation_function (Argument), 121
    model[standard]/descriptor[se_atten_v2]/activation_function (Argument), 130
    model[standard]/descriptor[se_atten]/activation_function (Argument), 126
    model[standard]/descriptor[se_e2_a]/activation_function (Argument), 117
    model[standard]/descriptor[se_e2_r]/activation_function (Argument), 124
    model[standard]/descriptor[se_e3]/activation_function (Argument), 120
    model[standard]/fitting_net[dipole]/activation_function (Argument), 144
    model[standard]/fitting_net[dos]/activation_function (Argument), 142
    model[standard]/fitting_net[ener]/activation_function (Argument), 140
    model[standard]/fitting_net[polar]/activation_function (Argument), 143
ActivationFn (class in deepmd.pt.utils.utils), 637
add() (deepmd.utils.data.DeepmdData method), 944
add() (deepmd.utils.data_system.DeepmdDataSystem method), 948
add() (deepmd.tf.utils.DeepmdData method), 985
add() (deepmd.tf.utils.DeepmdDataSystem method), 988
add() (deepmd.utils.data.DeepmdData method), 1026

add() (deepmd.utils.data_system.DeepmdDataSystem method), 1030
add_data_requirement() (deepmd.pt.utils.dataloader.DpLoaderSet method), 615
add_data_requirement() (deepmd.pt.utils.dataset.DeepmdDataSetForLoader method), 616
add_data_requirement() (in module deepmd.common), 1060
add_data_requirement() (in module deepmd.tf.common), 997
add_dict() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 948
add_dict() (deepmd.tf.utils.DeepmdDataSystem method), 988
add_dict() (deepmd.utils.data_system.DeepmdDataSystem method), 1030
add_flt_nvnmd (C++ function), 1280
add_flt_nvnmd() (in module deepmd.tf.env.op_module), 1095
AddFltNvnmd() (in module deepmd.tf.env.op_module), 1069
Annotated (class in deepmd.tf.env.op_grads_module), 1122
Annotated (class in deepmd.tf.env.op_module), 1069
Any (class in deepmd.tf.env.op_grads_module), 1123
Any (class in deepmd.tf.env.op_module), 1070
append_neighbors() (in module deepmd.pt.utils.preprocess), 631
apply_operation() (in module deepmd.dpmodel.output_def), 378
apply_out_stat() (deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicModel method), 262
apply_out_stat() (deepmd.dpmodel.atomic_model.BaseAtomicModel method), 273
apply_out_stat() (deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel method), 437
apply_out_stat() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 453
apply_out_stat() (deepmd.pt.model.atomic_model.dipole_atomic_model.DPDipoleAtomicModel method), 438
apply_out_stat() (deepmd.pt.model.atomic_model.DPDipoleAtomicModel method), 455
apply_out_stat() (deepmd.pt.model.atomic_model.DPPolarAtomicModel method), 464
apply_out_stat() (deepmd.pt.model.atomic_model.linear_atomic_model.LinearEnergyAtomicModel method), 444
apply_out_stat() (deepmd.pt.model.atomic_model.LinearEnergyAtomicModel method), 459
apply_out_stat() (deepmd.pt.model.atomic_model.polar_atomic_model.PolarAtomicModel method), 450
ArgsPlugin (class in deepmd.utils.argcheck), 1016
atom_ener:
    model[standard]/fitting_net[ener]/atom_ener (Argument), 141
atom_exclude_types:
    model/atom_exclude_types (Argument), 112
AtomAttention (class in deepmd.pt.model.network.network), 572
AtomExcludeMask (class in deepmd.dpmodel.utils), 367
AtomExcludeMask (class in deepmd.dpmodel.utils.exclude_mask), 353
AtomExcludeMask (class in deepmd.pt.utils), 638
AtomExcludeMask (class in deepmd.pt.utils.exclude_mask), 620
atomic_output_def() (deepmd.dpmodel.atomic_model.base_atomic_model.BaseA method), 261
atomic_output_def() (deepmd.dpmodel.atomic_model.BaseAtomicModel method), 272
atomic_output_def() (deepmd.pt.model.atomic_model.base_atomic_model.BaseA method), 436
atomic_output_def() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 452
atomic_virial_corr() (in module deepmd.pt.model.model.transform_output), 540
Atten2EquiVarApply (class in deepmd.pt.model.descriptor.repformer_layer), 485
Atten2Map (class in deepmd.pt.model.descriptor.repformer_layer), 484
Atten2MultiHeadApply (class in deepmd.pt.model.descriptor.repformer_layer), 485
Attention (class in deepmd.pt.model.network.network), 572
ATTENTION_LAYER_PATTERN (in module deepmd.tf.utils.learning_rate), 1002
attn:
    model[standard]/descriptor[se_atten_v2]/attn (Argument), 131
    model[standard]/descriptor[se_atten]/attn (Argument), 127
attn_dotr:
    model[standard]/descriptor[se_atten_v2]/attn_dotr (Argument), 131
    model[standard]/descriptor[se_atten]/attn_dotr (Argument), 127
attn_layer:
    model[standard]/descriptor[se_atten_v2]/attn_layer (Argument), 131

model[standard]/descriptor[se_atten]/attn_layer (Argument), 127
attn_mask:
model[standard]/descriptor[se_atten_v2]/attn_mask (Argument), 131
model[standard]/descriptor[se_atten]/attn_mask (Argument), 127
auto_prob:
training/training_data/auto_prob (Argument), 155
training/validation_data/auto_prob (Argument), 156
AutoBatchSize (class in deepmd.dpmodel.utils.batch_size), 351
AutoBatchSize (class in deepmd.pt.utils.auto_batch_size), 613
AutoBatchSize (class in deepmd.tf.utils.batch_size), 937
AutoBatchSize (class in deepmd.utils.batch_size), 1020
avg() (deepmd.tf.utils.data.DeepmdData method), 946
avg() (deepmd.tf.utils.DeepmdData method), 987
avg() (deepmd.utils.data.DeepmdData method), 1028
axis_neuron:
model[standard]/descriptor[se_a_ebd_v2]/axis_neuron (Argument), 136
model[standard]/descriptor[se_a_mask]/axis_neuron (Argument), 138
model[standard]/descriptor[se_a_tpe]/axis_neuron (Argument), 121
model[standard]/descriptor[se_atten_v2]/axis_neuron (Argument), 130
model[standard]/descriptor[se_atten]/axis_neuron (Argument), 126
model[standard]/descriptor[se_e2_a]/axis_neuron (Argument), 117
axis_rule:
model[standard]/descriptor[loc_frame]/axis_rule (Argument), 116

B

b_to_face_distance() (in module deepmd.dpmodel.utils.region), 362
b_to_face_distance() (in module deepmd.pt.utils.region), 632
b_to_face_distance() (in module deepmd.tf.utils.region), 977
BackBone (class in deepmd.pt.model.backbone), 467
BackBone (class in deepmd.pt.model.backbone.backbone), 465
Backend (class in deepmd.backend.backend), 253
Backend.Feature (class in deepmd.backend.backend), 254
BACKEND_TABLE (in module deepmd.main), 1065
BackendOption (class in deepmd.main), 1065
BACKENDS (in module deepmd.main), 1065
BackgroundConsumer (class in deepmd.pt.utils.dataloader), 615
BaseAtomicModel (class in deepmd.dpmodel.atomic_model), 272
BaseAtomicModel (class in deepmd.dpmodel.atomic_model.base_atomic_model), 261
BaseAtomicModel (class in deepmd.pt.model.atomic_model), 451
BaseAtomicModel (class in deepmd.pt.model.atomic_model.base_atomic_model), 434
BaseAtomicModel_ (in module deepmd.dpmodel.atomic_model.base_atomic_model), 261
BaseAtomicModel_ (in module deepmd.pt.model.atomic_model.base_atomic_model), 434
BaseDescriptor (in module deepmd.dpmodel.descriptor.base_descriptor), 281
BaseDescriptor (in module deepmd.pt.model.descriptor), 505
BaseDescriptor (in module deepmd.pt.model.descriptor.base_descriptor), 468
BaseFitting (in module deepmd.dpmodel.fitting.base_fitting), 310
BaseFitting (in module deepmd.pt.model.task), 597
BaseFitting (in module deepmd.pt.model.task.base_fitting), 579
BaseModel (class in deepmd.dpmodel.model.base_model), 342
BaseModel (class in deepmd.pt.model.model), 545
BaseModel (class in deepmd.pt.model.model.model), 535
BaseUpdateSel (class in deepmd.utils.update_sel), 1056
batch_size:
training/training_data/batch_size (Argument), 154
training/validation_data/batch_size (Argument), 155
bias (deepmd.pt.model.network.network.Linear attribute), 558
bin2hex() (deepmd.tf.nvnmd.utils.Encode method), 922
bin2hex() (deepmd.tf.nvnmd.utils.encode.Encode

method), 915
bin2hex_str() (deepmd.tf.nvnmd.utils.Encode method), 922
bin2hex_str() (deepmd.tf.nvnmd.utils.encode.Encode method), 915
border_op() (in module deepmd.pt.model.descriptor.repformers), 490
BufferedIterator (class in deepmd.pt.utils.dataloader), 616
BUILD (deepmd.utils.summary.SummaryPrinter attribute), 1056
build() (deepmd.tf.descriptor.Descriptor method), 701
build() (deepmd.tf.descriptor.descriptor.Descriptor method), 643
build() (deepmd.tf.descriptor.DescrptHybrid method), 707
build() (deepmd.tf.descriptor.DescrptLocFrame method), 712
build() (deepmd.tf.descriptor.DescrptSeA method), 717
build() (deepmd.tf.descriptor.DescrptSeAEbd method), 720
build() (deepmd.tf.descriptor.DescrptSeAEf method), 724
build() (deepmd.tf.descriptor.DescrptSeAEfLower method), 725
build() (deepmd.tf.descriptor.DescrptSeAMask method), 728
build() (deepmd.tf.descriptor.DescrptSeAtten method), 732
build() (deepmd.tf.descriptor.DescrptSeR method), 740
build() (deepmd.tf.descriptor.DescrptSeT method), 744
build() (deepmd.tf.descriptor.hybrid.DescrptHybrid method), 650
build() (deepmd.tf.descriptor.loc_frame.DescrptLocFrame method), 654
build() (deepmd.tf.descriptor.se_a.DescrptSeA method), 663
build() (deepmd.tf.descriptor.se_a_ebd.DescrptSeAEbd method), 666
build() (deepmd.tf.descriptor.se_a_ef.DescrptSeAEf method), 670
build() (deepmd.tf.descriptor.se_a_ef.DescrptSeAEfLower method), 672
build() (deepmd.tf.descriptor.se_a_mask.DescrptSeAMask method), 675
build() (deepmd.tf.descriptor.se_atten.DescrptDPA1Compat method), 686
build() (deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 680
build() (deepmd.tf.descriptor.se_r.DescrptSeR method), 692
build() (deepmd.tf.descriptor.se_t.DescrptSeT method), 697
build() (deepmd.tf.fit.dipole.DipoleFittingSeA method), 761
build() (deepmd.tf.fit.DipoleFittingSeA method), 779
build() (deepmd.tf.fit.dos.DOSFitting method), 765
build() (deepmd.tf.fit.DOSFitting method), 782
build() (deepmd.tf.fit.ener.EnerFitting method), 769
build() (deepmd.tf.fit.EnerFitting method), 786
build() (deepmd.tf.fit.GlobalPolarFittingSeA method), 790
build() (deepmd.tf.fit.polar.GlobalPolarFittingSeA method), 776
build() (deepmd.tf.fit.polar.PolarFittingSeA method), 774
build() (deepmd.tf.fit.PolarFittingSeA method), 792
build() (deepmd.tf.loss.dos.DOSLoss method), 837
build() (deepmd.tf.loss.DOSLoss method), 845
build() (deepmd.tf.loss.ener.EnerDipoleLoss method), 841
build() (deepmd.tf.loss.ener.EnerSpinLoss method), 841
build() (deepmd.tf.loss.ener.EnerStdLoss method), 840
build() (deepmd.tf.loss.EnerDipoleLoss method), 846
build() (deepmd.tf.loss.EnerSpinLoss method), 847
build() (deepmd.tf.loss.EnerStdLoss method), 849
build() (deepmd.tf.loss.loss.Loss method), 842
build() (deepmd.tf.loss.tensor.TensorLoss method), 844
build() (deepmd.tf.loss.TensorLoss method), 849
build() (deepmd.tf.model.dos.DOSModel method), 851
build() (deepmd.tf.model.DOSModel method), 889
build() (deepmd.tf.model.ener.EnerModel method), 854
build() (deepmd.tf.model.EnerModel method), 892
build() (deepmd.tf.model.frozen.FrozenModel method), 856
build() (deepmd.tf.model.FrozenModel method), 881
build() (deepmd.tf.model.linear.LinearEnergyModel method), 859
build() (deepmd.tf.model.LinearEnergyModel method), 883
build() (deepmd.tf.model.model.Model method), 862
build() (deepmd.tf.model.multi.MultiModel method), 870

build() (deepmd.tf.model.MultiModel method), 894
build() (deepmd.tf.model.pairtab.PairTabModel method), 872
build() (deepmd.tf.model.PairTabModel method), 884
build() (deepmd.tf.model.pairwise_dprc.PairwiseDPRc method), 874
build() (deepmd.tf.model.PairwiseDPRc method), 886
build() (deepmd.tf.model.tensor.TensorModel method), 878
build() (deepmd.tf.train.trainer.DatasetLoader method), 936
build() (deepmd.tf.train.trainer.DPTrainer method), 934
build() (deepmd.tf.utils.learning_rate.LearningRateExp method), 959
build() (deepmd.tf.utils.LearningRateExp method), 991
build() (deepmd.tf.utils.neighbor_stat.NeighborStat method), 961
build() (deepmd.tf.utils.neighbor_stat.NeighborStatOP method), 961
build() (deepmd.tf.utils.spin.Spin method), 979
build() (deepmd.tf.utils.tabulate.DPTabulate method), 980
build() (deepmd.tf.utils.type_embed.TypeEmbedNet method), 983
build_davg_dstd() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 904
build_davg_dstd() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
build_davg_dstd() (in module deepmd.tf.nvnmd.descriptor.se_a), 899
build_davg_dstd() (in module deepmd.tf.nvnmd.descriptor.se_atten), 900
build_descrpt() (deepmd.tf.model.model.Model method), 862
build_embedding_net() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 904
build_embedding_net() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
build_fv_graph() (deepmd.tf.DipoleChargeModifier method), 1011
build_fv_graph() (deepmd.tf.infer.data_modifier.DipoleChargeModifier method), 794
build_fv_graph() (deepmd.tf.infer.DipoleChargeModifier method), 821
build_grad() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 903
build_grad() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
build_inside_clist() (in module deepmd.pt.utils.preprocess), 631
build_map() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 903
build_map() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
build_map_coef() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 903
build_map_coef() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
build_multiple_neighbor_list() (in module deepmd.dpmodel.utils), 370
build_multiple_neighbor_list() (in module deepmd.dpmodel.utils.nlist), 360
build_multiple_neighbor_list() (in module deepmd.pt.utils.nlist), 627
build_neighbor_list() (deepmd.tf.DeepEval method), 1008
build_neighbor_list() (deepmd.tf.infer.deep_eval.DeepEval method), 801
build_neighbor_list() (deepmd.tf.infer.deep_eval.DeepEvalOld method), 807
build_neighbor_list() (deepmd.tf.infer.DeepEval method), 826
build_neighbor_list() (in module deepmd.dpmodel.utils), 371
build_neighbor_list() (in module deepmd.dpmodel.utils.nlist), 359
build_neighbor_list() (in module deepmd.pt.utils.nlist), 626
build_neighbor_list() (in module deepmd.pt.utils.preprocess), 631
build_nlist (C++ function), 1280, 1281
build_op_descriptor() (in module deepmd.tf.nvnmd.descriptor.se_a), 899
build_op_descriptor() (in module deepmd.tf.nvnmd.descriptor.se_atten), 900
build_s2g() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 903
build_s2g() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
build_s2g_grad() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 903
build_s2g_grad() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
build_t2g() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 903
build_t2g() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
build_type_embedding()

(deepmd.tf.model.model.Model method), 863
build_type_exclude_mask() (deepmd.dpmodel.utils.AtomExcludeMask method), 368
build_type_exclude_mask() (deepmd.dpmodel.utils.exclude_mask.AtomExcludeMask method), 353
build_type_exclude_mask() (deepmd.dpmodel.utils.exclude_mask.PairExcludeMask method), 353
build_type_exclude_mask() (deepmd.dpmodel.utils.PairExcludeMask method), 368
build_type_exclude_mask() (deepmd.tf.descriptor.Descriptor method), 704
build_type_exclude_mask() (deepmd.tf.descriptor.descriptor.Descriptor method), 646
build_type_exclude_mask_mixed() (deepmd.tf.descriptor.DescrptSeAtten method), 734
build_type_exclude_mask_mixed() (deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 681
build_u2s() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 903
build_u2s() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
build_u2s_grad() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 903
build_u2s_grad() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
byte2hex() (deepmd.tf.nvnmd.utils.Encode method), 921
byte2hex() (deepmd.tf.nvnmd.utils.encode.Encode method), 914

C

CACHE_PER_SYS (in module deepmd.pt.utils.env), 618
cal_1_dim() (deepmd.pt.model.descriptor.repformer_layer.RepformerLayer method), 488
cal_coef4() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 903
cal_coef4() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
cal_g() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 301
cal_g() (deepmd.dpmodel.descriptor.DescrptSeA method), 306
cal_g() (deepmd.dpmodel.descriptor.DescrptSeR method), 309
cal_g() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 method), 285
cal_g() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA method), 293
cal_g() (deepmd.dpmodel.descriptor.se_r.DescrptSeR method), 296
cal_g_strip() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 301
cal_g_strip() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 method), 285
calc_model_devi() (in module deepmd.infer), 410
calc_model_devi() (in module deepmd.infer.model_devi), 404
calc_model_devi() (in module deepmd.tf.infer), 834
calc_model_devi() (in module deepmd.tf.infer.model_devi), 816
calc_model_devi_e() (in module deepmd.infer.model_devi), 402
calc_model_devi_e() (in module deepmd.tf.infer.model_devi), 817
calc_model_devi_f() (in module deepmd.infer.model_devi), 402
calc_model_devi_f() (in module deepmd.tf.infer.model_devi), 817
calc_model_devi_v() (in module deepmd.infer.model_devi), 403
calc_model_devi_v() (in module deepmd.tf.infer.model_devi), 818
calculate() (deepmd.calculator.DP method), 1059
calculate() (deepmd.pt.utils.ase_calc.DPCalculator method), 612
calculate() (deepmd.tf.calculator.DP method), 995
call() (deepmd.dpmodel.common.NativeOP method), 374
call() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 301
call() (deepmd.dpmodel.descriptor.DescrptHybrid method), 303
call() (deepmd.dpmodel.descriptor.DescrptSeA method), 306
call() (deepmd.dpmodel.descriptor.DescrptSeR method), 309
call() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 method), 285
call() (deepmd.dpmodel.descriptor.dpa1.GatedAttentionLayer method), 288
call() (deepmd.dpmodel.descriptor.dpa1.NeighborGatedAttention method), 286
call() (deepmd.dpmodel.descriptor.dpa1.NeighborGatedAttentionLa method), 287
call() (deepmd.dpmodel.descriptor.hybrid.DescrptHybrid method), 289
call() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA method), 293

call() (deepmd.dpmodel.descriptor.se_r.DescrptSeR method), 296
call() (deepmd.dpmodel.fitting.dipole_fitting.DipoleFitting method), 312
call() (deepmd.dpmodel.fitting.DipoleFitting method), 329
call() (deepmd.dpmodel.fitting.invar_fitting.InvarFitting method), 323
call() (deepmd.dpmodel.fitting.InvarFitting method), 335
call() (deepmd.dpmodel.fitting.PolarFitting method), 338
call() (deepmd.dpmodel.fitting.polarizability_fitting.PolarFitting method), 326
call() (deepmd.dpmodel.model.spin_model.SpinModel method), 346
call() (deepmd.dpmodel.model.SpinModel method), 350
call() (deepmd.dpmodel.NativeOP method), 380
call() (deepmd.dpmodel.utils.env_mat.EnvMat method), 352
call() (deepmd.dpmodel.utils.EnvMat method), 367
call() (deepmd.dpmodel.utils.NativeLayer method), 369
call() (deepmd.dpmodel.utils.neighbor_stat.NeighborStat method), 354
call() (deepmd.dpmodel.utils.network.LayerNorm method), 358
call() (deepmd.dpmodel.utils.network.NativeLayer method), 357
call() (deepmd.dpmodel.utils.type_embed.TypeEmbedNet method), 365
call_lower() (deepmd.dpmodel.model.spin_model.SpinModel method), 346
call_lower() (deepmd.dpmodel.model.SpinModel method), 350
cast_precision() (in module deepmd.tf.common), 1001
change_energy_bias() (deepmd.tf.fit.ener.EnerFitting method), 770
change_energy_bias() (deepmd.tf.fit.EnerFitting method), 786
change_energy_bias() (deepmd.tf.model.ener.EnerModel method), 855
change_energy_bias() (deepmd.tf.model.EnerModel method), 893
change_energy_bias() (deepmd.tf.model.model.Model method), 864
change_energy_bias_lower() (in module deepmd.utils.finetune), 1039
change_finetune_model_params() (in module deepmd.pt.utils.finetune), 621
change_finetune_model_params_single() (in module deepmd.pt.utils.finetune), 621
change_out_bias() (deepmd.pt.model.atomic_model.base_atomic_m method), 437
change_out_bias() (deepmd.pt.model.atomic_model.BaseAtomicMo method), 454
check_batch_size() (deepmd.tf.utils.data.DeepmdData method), 945
check_batch_size() (deepmd.tf.utils.DeepmdData method), 986
check_batch_size() (deepmd.utils.data.DeepmdData method), 1027
check_completeness() (deepmd.dpmodel.utils.network.NetworkCollection method), 359
check_completeness() (deepmd.dpmodel.utils.NetworkCollection method), 370
check_dec() (deepmd.tf.nvnmd.utils.Encode method), 921
check_dec() (deepmd.tf.nvnmd.utils.encode.Encode method), 914
check_operation_applied() (in module deepmd.dpmodel.output_def), 379
check_shape() (in module deepmd.dpmodel.output_def), 375
check_shape_consistency() (deepmd.dpmodel.utils.NativeLayer method), 369
check_shape_consistency() (deepmd.dpmodel.utils.network.NativeLayer method), 357
check_switch_range() (in module deepmd.tf.nvnmd.descriptor.se_a), 899
check_switch_range() (in module deepmd.tf.nvnmd.descriptor.se_atten), 900
check_test_size() (deepmd.tf.utils.data.DeepmdData method), 945
check_test_size() (deepmd.tf.utils.DeepmdData method), 986
check_test_size() (deepmd.utils.data.DeepmdData method), 1027
check_type_consistency() (deepmd.dpmodel.utils.NativeLayer method), 369
check_type_consistency() (deepmd.dpmodel.utils.network.NativeLayer method), 357
check_type_consistency() (deepmd.pt.model.network.mlp.MLPLayer method), 551

check_var() (in module deepmd.dpmodel.output_def), 375
check_version_compatibility() (in module deepmd.utils.version), 1057
checkpoint_sequential() (in module deepmd.pt.model.network.network), 555
choice() (in module deepmd.pt.utils.dp_random), 617
choice() (in module deepmd.tf.utils.random), 975
choice() (in module deepmd.utils.random), 1052
CITATION (deepmd.utils.summary.SummaryPrinter attribute), 1056
clear_session() (in module deepmd.tf.common), 1001
collate_batch() (in module deepmd.pt.utils.dataloader), 616
communicate_extended_output() (in module deepmd.dpmodel.model.transform_output), 347
communicate_extended_output() (in module deepmd.pt.model.model.transform_output), 541
compress() (in module deepmd.tf.entrypoints), 756
compress() (in module deepmd.tf.entrypoints.compress), 746
compress:
    model/compress (Argument), 114
compute_avg() (deepmd.utils.env_mat_stat.StatItem method), 1036
compute_descriptor (C++ function), 1282
compute_descriptor_se_a_ef_para (C++ function), 1282
compute_descriptor_se_a_ef_vert (C++ function), 1283
compute_descriptor_se_a_extf (C++ function), 1283
compute_energy_shift() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 947
compute_energy_shift() (deepmd.tf.utils.DeepmdDataSystem method), 988
compute_energy_shift() (deepmd.utils.data_system.DeepmdDataSystem method), 1030
compute_input_stats() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 301
compute_input_stats() (deepmd.dpmodel.descriptor.DescrptHybrid method), 303
compute_input_stats() (deepmd.dpmodel.descriptor.DescrptSeA method), 306
compute_input_stats() (deepmd.dpmodel.descriptor.DescrptSeR method), 309
compute_input_stats() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 method), 285
compute_input_stats() (deepmd.dpmodel.descriptor.hybrid.DescrptHybrid method), 289
compute_input_stats() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA method), 293
compute_input_stats() (deepmd.dpmodel.descriptor.se_r.DescrptSeR method), 296
compute_input_stats() (deepmd.pt.model.descriptor.descriptor.DescriptorBlock method), 469
compute_input_stats() (deepmd.pt.model.descriptor.DescriptorBlock method), 506
compute_input_stats() (deepmd.pt.model.descriptor.DescrptBlockRepformers method), 522
compute_input_stats() (deepmd.pt.model.descriptor.DescrptBlockSeA method), 523
compute_input_stats() (deepmd.pt.model.descriptor.DescrptBlockSeAtten method), 508
compute_input_stats() (deepmd.pt.model.descriptor.DescrptDPA1 method), 512
compute_input_stats() (deepmd.pt.model.descriptor.DescrptDPA2 method), 515
compute_input_stats() (deepmd.pt.model.descriptor.DescrptGaussianLcc method), 518
compute_input_stats() (deepmd.pt.model.descriptor.DescrptHybrid method), 519
compute_input_stats() (deepmd.pt.model.descriptor.DescrptSeA method), 524
compute_input_stats() (deepmd.pt.model.descriptor.DescrptSeR method), 527
compute_input_stats() (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 method), 473
compute_input_stats() (deepmd.pt.model.descriptor.dpa2.DescrptDPA2 method), 477

compute_input_stats() (deepmd.pt.model.descriptor.gaussian_lcc.DescrptGaussianLcc method), 481
compute_input_stats() (deepmd.pt.model.descriptor.hybrid.DescrptHybrid method), 482
compute_input_stats() (deepmd.pt.model.descriptor.repformers.DescrptBlockRepformers method), 491
compute_input_stats() (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA method), 495
compute_input_stats() (deepmd.pt.model.descriptor.se_a.DescrptSeA method), 493
compute_input_stats() (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtten method), 497
compute_input_stats() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 503
compute_input_stats() (deepmd.tf.descriptor.Descriptor method), 700
compute_input_stats() (deepmd.tf.descriptor.descriptor.Descriptor method), 642
compute_input_stats() (deepmd.tf.descriptor.DescrptHybrid method), 706
compute_input_stats() (deepmd.tf.descriptor.DescrptLocFrame method), 711
compute_input_stats() (deepmd.tf.descriptor.DescrptSeA method), 715
compute_input_stats() (deepmd.tf.descriptor.DescrptSeAEf method), 723
compute_input_stats() (deepmd.tf.descriptor.DescrptSeAEfLower method), 725
compute_input_stats() (deepmd.tf.descriptor.DescrptSeAMask method), 727
compute_input_stats() (deepmd.tf.descriptor.DescrptSeAtten method), 731
compute_input_stats() (deepmd.tf.descriptor.DescrptSeR method), 739
compute_input_stats() (deepmd.tf.descriptor.DescrptSeT method), 743
compute_input_stats() (deepmd.tf.descriptor.hybrid.DescrptHybrid method), 649
compute_input_stats() (deepmd.tf.descriptor.loc_frame.DescrptLocFrame method), 654
compute_input_stats() (deepmd.tf.descriptor.se_a.DescrptSeA method), 661
compute_input_stats() (deepmd.tf.descriptor.se_a_ef.DescrptSeAEf method), 670
compute_input_stats() (deepmd.tf.descriptor.se_a_ef.DescrptSeAEfLower method), 671
compute_input_stats() (deepmd.tf.descriptor.se_a_mask.DescrptSeAMask method), 674
compute_input_stats() (deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 678
compute_input_stats() (deepmd.tf.descriptor.se_r.DescrptSeR method), 691
compute_input_stats() (deepmd.tf.descriptor.se_t.DescrptSeT method), 695
compute_input_stats() (deepmd.tf.fit.dos.DOSFitting method), 764
compute_input_stats() (deepmd.tf.fit.DOSFitting method), 782
compute_input_stats() (deepmd.tf.fit.ener.EnerFitting method), 769
compute_input_stats() (deepmd.tf.fit.EnerFitting method), 785
compute_or_load_out_stat() (deepmd.pt.model.atomic_model.base_atomic_model.BaseA method), 437
compute_or_load_out_stat() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 453
compute_or_load_out_stat() (deepmd.pt.model.atomic_model.linear_atomic_model.Linea method), 445
compute_or_load_out_stat() (deepmd.pt.model.atomic_model.LinearEnergyAtomicMode method), 460
compute_or_load_stat() (deepmd.pt.model.atomic_model.base_atomic_model.BaseA method), 436
compute_or_load_stat() (deepmd.pt.model.atomic_model.BaseAtomicModel

method), 453
compute_or_load_stat()
(deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel method), 441
compute_or_load_stat()
(deepmd.pt.model.atomic_model.DPAtomicModel method), 456
compute_or_load_stat()
(deepmd.pt.model.atomic_model.linear_atomic_model.LinearAtomicModel method), 445
compute_or_load_stat()
(deepmd.pt.model.atomic_model.LinearEnergyAtomicModel method), 460
compute_or_load_stat()
(deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 447
compute_or_load_stat()
(deepmd.pt.model.atomic_model.PairTabAtomicModel method), 462
compute_or_load_stat()
(deepmd.pt.model.model.BaseModel method), 546
compute_or_load_stat()
(deepmd.pt.model.model.model.BaseModel method), 536
compute_or_load_stat()
(deepmd.pt.model.model.spin_model.SpinModel method), 539
compute_or_load_stat()
(deepmd.pt.model.model.SpinModel method), 548
compute_output_stats()
(deepmd.dpmodel.fitting.invar_fitting.InvarFitting method), 323
compute_output_stats()
(deepmd.dpmodel.fitting.InvarFitting method), 335
compute_output_stats()
(deepmd.pt.model.task.dipole.DipoleFittingNet method), 582
compute_output_stats()
(deepmd.pt.model.task.DipoleFittingNet method), 599
compute_output_stats()
(deepmd.tf.fit.dos.DOSFitting method), 764
compute_output_stats() (deepmd.tf.fit.DOSFitting method), 781
compute_output_stats()
(deepmd.tf.fit.ener.EnerFitting method), 768
compute_output_stats()
(deepmd.tf.fit.EnerFitting method), 785
compute_output_stats()
(deepmd.tf.fit.polar.PolarFittingSeA method), 774
compute_output_stats()
(deepmd.tf.fit.PolarFittingSeA method), 792
compute_output_stats() (in module deepmd.pt.utils.stat), 635
compute_output_stats_atomic() (in module deepmd.pt.utils.stat), 636
compute_output_stats_global() (in module deepmd.pt.utils.stat), 636
compute_shift() (in module deepmd.pt.utils.preprocess), 631
compute_prec:
training/mixed_precision/compute_prec (Argument), 156
compute_serial_cid() (in module deepmd.pt.utils.preprocess), 630
compute_smooth_weight() (in module deepmd.dpmodel.utils.env_mat), 352
compute_smooth_weight() (in module deepmd.pt.utils.preprocess), 631
compute_stat() (deepmd.pt.utils.env_mat_stat.EnvMatStat method), 619
compute_stats() (deepmd.utils.env_mat_stat.EnvMatStat method), 1037
compute_stats_from_atomic() (in module deepmd.utils.out_stat), 1043
compute_stats_from_redu() (in module deepmd.utils.out_stat), 1042
compute_std() (deepmd.utils.env_mat_stat.StatItem method), 1036
concat_output_tebd:
model[standard]/descriptor[dpa2]/concat_output_tebd (Argument), 132
model[standard]/descriptor[se_atten]/concat_output_tebd (Argument), 129
concat_switch_virtual()
(deepmd.dpmodel.model.spin_model.SpinModel static method), 345
concat_switch_virtual()
(deepmd.dpmodel.model.SpinModel static method), 349
concat_switch_virtual()
(deepmd.pt.model.model.spin_model.SpinModel static method), 538
concat_switch_virtual()
(deepmd.pt.model.model.SpinModel static method), 547
conf_keys (in module deepmd.utils.econf_embd), 1035
config_file:
nvnmd/config_file (Argument), 160
convert() (in module deepmd.tf.entrypoints), 756

convert() (in module deepmd.tf.entrypoints.convert), 747
convert_012_to_21() (in module deepmd.tf.utils.convert), 942
convert_10_to_21() (in module deepmd.tf.utils.convert), 942
convert_12_to_21() (in module deepmd.tf.utils.convert), 942
convert_13_to_21() (in module deepmd.tf.utils.convert), 941
convert_20_to_21() (in module deepmd.tf.utils.convert), 942
convert_backend() (in module deepmd.entrypoints.convert_backend), 382
convert_dp012_to_dp10() (in module deepmd.tf.utils.convert), 943
convert_dp10_to_dp11() (in module deepmd.tf.utils.convert), 943
convert_dp12_to_dp13() (in module deepmd.tf.utils.convert), 943
convert_dp13_to_dp20() (in module deepmd.tf.utils.convert), 943
convert_dp20_to_dp21() (in module deepmd.tf.utils.convert), 943
convert_forward_map() (in module deepmd.tf.env.op_module), 1096
convert_input_v0_v1() (in module deepmd.tf.utils.compat), 938
convert_input_v0_v1() (in module deepmd.utils.compat), 1022
convert_input_v1_v2() (in module deepmd.tf.utils.compat), 939
convert_input_v1_v2() (in module deepmd.utils.compat), 1024
convert_pb_to_pbtxt() (in module deepmd.tf.utils.convert), 942
convert_pbtxt_to_pb() (in module deepmd.tf.utils.convert), 942
convert_tensor_to_ndarray_in_dict() (in module deepmd.tf.utils.graph), 958
convert_to_21() (in module deepmd.tf.utils.convert), 941
ConvertForwardMap() (in module deepmd.tf.env.op_module), 1070
copy_coord (C++ function), 1284
copy_flt_nvnmd() (in module deepmd.tf.env.op_module), 1096
CopyFltNvnmd() (in module deepmd.tf.env.op_module), 1070
Counter (class in deepmd.dpmodel.utils.serialization), 363
create_file_path() (deepmd.tf.nvnmd.utils.fio.Fio method), 915
cuda_library_paths (in module deepmd.tf.lmp), 1005

D

data_bias_nsample:
    model/data_bias_nsample (Argument), 111
data_dict:
    training/data_dict (Argument), 158
data_requirement (in module deepmd.common), 1060
data_requirement (in module deepmd.tf.common), 997
data_stat() (deepmd.tf.model.dos.DOSModel method), 851
data_stat() (deepmd.tf.model.DOSModel method), 889
data_stat() (deepmd.tf.model.ener.EnerModel method), 854
data_stat() (deepmd.tf.model.EnerModel method), 892
data_stat() (deepmd.tf.model.frozen.FrozenModel method), 857
data_stat() (deepmd.tf.model.FrozenModel method), 882
data_stat() (deepmd.tf.model.linear.LinearModel method), 859
data_stat() (deepmd.tf.model.model.Model method), 865
data_stat() (deepmd.tf.model.multi.MultiModel method), 870
data_stat() (deepmd.tf.model.MultiModel method), 894
data_stat() (deepmd.tf.model.pairtab.PairTabModel method), 873
data_stat() (deepmd.tf.model.PairTabModel method), 886
data_stat() (deepmd.tf.model.pairwise_dprc.PairwiseDPRc method), 875
data_stat() (deepmd.tf.model.PairwiseDPRc method), 887
data_stat() (deepmd.tf.model.tensor.TensorModel method), 878
data_stat_nbatch:
    model/data_stat_nbatch (Argument), 111
data_stat_protect:
    model/data_stat_protect (Argument), 111
DataRequirementItem (class in deepmd.utils.data), 1028
DatasetLoader (class in deepmd.tf.train.trainer), 936
dec2bin() (deepmd.tf.nvnmd.utils.Encode method), 922
dec2bin() (deepmd.tf.nvnmd.utils.encode.Encode method), 914
decay_rate:

learning_rate[exp]/decay_rate (Argument), 147
decay_steps:
learning_rate[exp]/decay_steps (Argument), 147
deep_eval (deepmd.backend.backend.Backend property), 254
DEEP_EVAL (deepmd.backend.backend.Backend.Feature attribute), 254
deep_eval (deepmd.backend.dpmodel.DPModelBackend property), 256
deep_eval (deepmd.backend.pytorch.PyTorchBackend property), 257
deep_eval (deepmd.backend.tensorflow.TensorFlowBackend property), 259
DeepDipole (class in deepmd.infer.deep_dipole), 387
DeepDipole (class in deepmd.tf.infer), 822
DeepDipole (class in deepmd.tf.infer.deep_dipole), 795
DeepDOS (class in deepmd.infer.deep_dos), 388
DeepDOS (class in deepmd.tf.infer), 822
DeepDOS (class in deepmd.tf.infer.deep_dos), 796
DeepEval (class in deepmd.dpmodel.infer.deep_eval), 338
DeepEval (class in deepmd.infer), 406
DeepEval (class in deepmd.infer.deep_eval), 392
DeepEval (class in deepmd.pt.infer.deep_eval), 416
DeepEval (class in deepmd.tf), 1005
DeepEval (class in deepmd.tf.infer), 823
DeepEval (class in deepmd.tf.infer.deep_eval), 798
DeepEvalBackend (class in deepmd.infer.deep_eval), 389
DeepEvalOld (class in deepmd.tf.infer.deep_eval), 804
DeepGlobalPolar (class in deepmd.infer.deep_polar), 395
DeepGlobalPolar (class in deepmd.tf.infer), 829
DeepGlobalPolar (class in deepmd.tf.infer.deep_polar), 808
deepmd
module, 253
deepmd.__main__
module, 1058
deepmd.backend
module, 253
deepmd.backend.backend
module, 253
deepmd.backend.dpmodel
module, 256
deepmd.backend.pytorch
module, 257
deepmd.backend.suffix
module, 258
deepmd.backend.tensorflow
module, 259
deepmd.calculator
module, 1058
deepmd.common
module, 1060
deepmd.dpmodel
module, 260
deepmd.dpmodel.atomic_model
module, 260
deepmd.dpmodel.atomic_model.base_atomic_model
module, 261
deepmd.dpmodel.atomic_model.dp_atomic_model
module, 263
deepmd.dpmodel.atomic_model.linear_atomic_model
module, 264
deepmd.dpmodel.atomic_model.make_base_atomic_model
module, 268
deepmd.dpmodel.atomic_model.pairtab_atomic_model
module, 268
deepmd.dpmodel.common
module, 374
deepmd.dpmodel.descriptor
module, 281
deepmd.dpmodel.descriptor.base_descriptor
module, 281
deepmd.dpmodel.descriptor.dpa1
module, 281
deepmd.dpmodel.descriptor.hybrid
module, 288
deepmd.dpmodel.descriptor.make_base_descriptor
module, 290
deepmd.dpmodel.descriptor.se_e2_a
module, 290
deepmd.dpmodel.descriptor.se_r
module, 294
deepmd.dpmodel.fitting
module, 310
deepmd.dpmodel.fitting.base_fitting
module, 310
deepmd.dpmodel.fitting.dipole_fitting
module, 310
deepmd.dpmodel.fitting.dos_fitting
module, 313
deepmd.dpmodel.fitting.ener_fitting
module, 315
deepmd.dpmodel.fitting.general_fitting
module, 317
deepmd.dpmodel.fitting.invar_fitting
module, 320
deepmd.dpmodel.fitting.make_base_fitting
module, 323
deepmd.dpmodel.fitting.polarizability_fitting
module, 324
deepmd.dpmodel.infer

module, 338
deepmd.dpmodel.infer.deep_eval
module, 338
deepmd.dpmodel.model
module, 341
deepmd.dpmodel.model.base_model
module, 341
deepmd.dpmodel.model.dp_model
module, 342
deepmd.dpmodel.model.ener_model
module, 343
deepmd.dpmodel.model.make_model
module, 343
deepmd.dpmodel.model.model
module, 344
deepmd.dpmodel.model.spin_model
module, 344
deepmd.dpmodel.model.transform_output
module, 347
deepmd.dpmodel.output_def
module, 374
deepmd.dpmodel.utils
module, 351
deepmd.dpmodel.utils.batch_size
module, 351
deepmd.dpmodel.utils.env_mat
module, 352
deepmd.dpmodel.utils.exclude_mask
module, 353
deepmd.dpmodel.utils.neighbor_stat
module, 354
deepmd.dpmodel.utils.network
module, 355
deepmd.dpmodel.utils.nlist
module, 359
deepmd.dpmodel.utils.region
module, 361
deepmd.dpmodel.utils.serialization
module, 363
deepmd.dpmodel.utils.type_embed
module, 364
deepmd.dpmodel.utils.update_sel
module, 365
deepmd.driver
module, 1063
deepmd.entrypoints
module, 382
deepmd.entrypoints.convert_backend
module, 382
deepmd.entrypoints.doc
module, 383
deepmd.entrypoints.gui
module, 383
deepmd.entrypoints.main
module, 384
deepmd.entrypoints.neighbor_stat
module, 384
deepmd.entrypoints.test
module, 386
deepmd.env
module, 1063
deepmd.infer
module, 387
deepmd.infer.deep_dipole
module, 387
deepmd.infer.deep_dos
module, 388
deepmd.infer.deep_eval
module, 389
deepmd.infer.deep_polar
module, 395
deepmd.infer.deep_pot
module, 397
deepmd.infer.deep_tensor
module, 399
deepmd.infer.deep_wfc
module, 401
deepmd.infer.model_devi
module, 402
deepmd.loggers
module, 411
deepmd.loggers.loggers
module, 412
deepmd.loggers.training
module, 413
deepmd.main
module, 1064
deepmd.pt
module, 414
deepmd.pt.cxx_op
module, 639
deepmd.pt.entrypoints
module, 414
deepmd.pt.entrypoints.main
module, 414
deepmd.pt.infer
module, 416
deepmd.pt.infer.deep_eval
module, 416
deepmd.pt.infer.inference
module, 419
deepmd.pt.loss
module, 419
deepmd.pt.loss.denoise
module, 419
deepmd.pt.loss.dos
module, 420
deepmd.pt.loss.ener

module, 422
deepmd.pt.loss.ener_spin
module, 423
deepmd.pt.loss.loss
module, 425
deepmd.pt.loss.tensor
module, 426
deepmd.pt.model
module, 434
deepmd.pt.model.atomic_model
module, 434
deepmd.pt.model.atomic_model.base_atomic_model
module, 434
deepmd.pt.model.atomic_model.dipole_atomic_model
module, 438
deepmd.pt.model.atomic_model.dos_atomic_model
module, 439
deepmd.pt.model.atomic_model.dp_atomic_model
module, 439
deepmd.pt.model.atomic_model.energy_atomic_model
module, 442
deepmd.pt.model.atomic_model.linear_atomic_model
module, 442
deepmd.pt.model.atomic_model.pairtab_atomic_model
module, 446
deepmd.pt.model.atomic_model.polar_atomic_model
module, 450
deepmd.pt.model.backbone
module, 464
deepmd.pt.model.backbone.backbone
module, 464
deepmd.pt.model.backbone.evoformer2b
module, 465
deepmd.pt.model.descriptor
module, 468
deepmd.pt.model.descriptor.base_descriptor
module, 468
deepmd.pt.model.descriptor.descriptor
module, 468
deepmd.pt.model.descriptor.dpa1
module, 470
deepmd.pt.model.descriptor.dpa2
module, 475
deepmd.pt.model.descriptor.env_mat
module, 479
deepmd.pt.model.descriptor.gaussian_lcc
module, 479
deepmd.pt.model.descriptor.hybrid
module, 481
deepmd.pt.model.descriptor.repformer_layer
module, 483
deepmd.pt.model.descriptor.repformers
module, 489
deepmd.pt.model.descriptor.se_a
module, 492
deepmd.pt.model.descriptor.se_atten
module, 496
deepmd.pt.model.descriptor.se_r
module, 502
deepmd.pt.model.model
module, 528
deepmd.pt.model.model.dipole_model
module, 529
deepmd.pt.model.model.dos_model
module, 529
deepmd.pt.model.model.dp_model
module, 530
deepmd.pt.model.model.dp_zbl_model
module, 531
deepmd.pt.model.model.ener_model
module, 532
deepmd.pt.model.model.frozen
module, 533
deepmd.pt.model.model.make_hessian_model
module, 534
deepmd.pt.model.model.make_model
module, 535
deepmd.pt.model.model.model
module, 535
deepmd.pt.model.model.polar_model
module, 537
deepmd.pt.model.model.spin_model
module, 537
deepmd.pt.model.model.transform_output
module, 540
deepmd.pt.model.network
module, 548
deepmd.pt.model.network.layernorm
module, 548
deepmd.pt.model.network.mlp
module, 550
deepmd.pt.model.network.network
module, 552
deepmd.pt.model.task
module, 578
deepmd.pt.model.task.atten_lcc
module, 578
deepmd.pt.model.task.base_fitting
module, 579
deepmd.pt.model.task.denoise
module, 579
deepmd.pt.model.task.dipole
module, 580
deepmd.pt.model.task.dos
module, 582
deepmd.pt.model.task.ener
module, 584
deepmd.pt.model.task.fitting

module, 588
deepmd.pt.model.task.invar_fitting
module, 591
deepmd.pt.model.task.polarizability
module, 593
deepmd.pt.model.task.task
module, 595
deepmd.pt.model.task.type_predict
module, 595
deepmd.pt.optimizer
module, 606
deepmd.pt.optimizer.KFWrapper
module, 606
deepmd.pt.optimizer.LKF
module, 606
deepmd.pt.train
module, 609
deepmd.pt.train.training
module, 609
deepmd.pt.train.wrapper
module, 610
deepmd.pt.utils
module, 611
deepmd.pt.utils.ase_calc
module, 611
deepmd.pt.utils.auto_batch_size
module, 613
deepmd.pt.utils.cache
module, 614
deepmd.pt.utils.dataloader
module, 614
deepmd.pt.utils.dataset
module, 616
deepmd.pt.utils.dp_random
module, 617
deepmd.pt.utils.env
module, 618
deepmd.pt.utils.env_mat_stat
module, 618
deepmd.pt.utils.exclude_mask
module, 620
deepmd.pt.utils.finetune
module, 621
deepmd.pt.utils.learning_rate
module, 622
deepmd.pt.utils.multi_task
module, 623
deepmd.pt.utils.neighbor_stat
module, 624
deepmd.pt.utils.nlist
module, 626
deepmd.pt.utils.plugin
module, 628
deepmd.pt.utils.preprocess
module, 630
deepmd.pt.utils.region
module, 631
deepmd.pt.utils.serialization
module, 633
deepmd.pt.utils.stat
module, 634
deepmd.pt.utils.update_sel
module, 636
deepmd.pt.utils.utils
module, 636
deepmd.tf
module, 639
deepmd.tf.__about__
module, 994
deepmd.tf.__main__
module, 994
deepmd.tf.calculator
module, 994
deepmd.tf.cluster
module, 639
deepmd.tf.cluster.local
module, 639
deepmd.tf.common
module, 996
deepmd.tf.descriptor
module, 640
deepmd.tf.descriptor.descriptor
module, 640
deepmd.tf.descriptor.hybrid
module, 647
deepmd.tf.descriptor.loc_frame
module, 653
deepmd.tf.descriptor.se
module, 656
deepmd.tf.descriptor.se_a
module, 659
deepmd.tf.descriptor.se_a_ebd
module, 665
deepmd.tf.descriptor.se_a_ebd_v2
module, 667
deepmd.tf.descriptor.se_a_ef
module, 668
deepmd.tf.descriptor.se_a_mask
module, 673
deepmd.tf.descriptor.se_atten
module, 676
deepmd.tf.descriptor.se_atten_v2
module, 688
deepmd.tf.descriptor.se_r
module, 689
deepmd.tf.descriptor.se_t
module, 694
deepmd.tf.entrypoints

module, 746
deepmd.tf.entrypoints.compress
module, 746
deepmd.tf.entrypoints.convert
module, 747
deepmd.tf.entrypoints.doc
module, 747
deepmd.tf.entrypoints.freeze
module, 747
deepmd.tf.entrypoints.gui
module, 748
deepmd.tf.entrypoints.ipi
module, 749
deepmd.tf.entrypoints.main
module, 750
deepmd.tf.entrypoints.neighbor_stat
module, 751
deepmd.tf.entrypoints.test
module, 752
deepmd.tf.entrypoints.train
module, 753
deepmd.tf.entrypoints.transfer
module, 754
deepmd.tf.env
module, 1001
deepmd.tf.env.op_grads_module
module, 1122
deepmd.tf.env.op_module
module, 1069
deepmd.tf.fit
module, 760
deepmd.tf.fit.dipole
module, 760
deepmd.tf.fit.dos
module, 763
deepmd.tf.fit.ener
module, 766
deepmd.tf.fit.fitting
module, 771
deepmd.tf.fit.polar
module, 773
deepmd.tf.infer
module, 794
deepmd.tf.infer.data_modifier
module, 794
deepmd.tf.infer.deep_dipole
module, 795
deepmd.tf.infer.deep_dos
module, 796
deepmd.tf.infer.deep_eval
module, 798
deepmd.tf.infer.deep_polar
module, 808
deepmd.tf.infer.deep_pot
module, 810
deepmd.tf.infer.deep_tensor
module, 812
deepmd.tf.infer.deep_wfc
module, 814
deepmd.tf.infer.ewald_recp
module, 815
deepmd.tf.infer.model_devi
module, 816
deepmd.tf.lmp
module, 1004
deepmd.tf.loggers
module, 835
deepmd.tf.loggers.loggers
module, 835
deepmd.tf.loss
module, 837
deepmd.tf.loss.dos
module, 837
deepmd.tf.loss.ener
module, 838
deepmd.tf.loss.loss
module, 842
deepmd.tf.loss.tensor
module, 844
deepmd.tf.model
module, 850
deepmd.tf.model.dos
module, 850
deepmd.tf.model.ener
module, 853
deepmd.tf.model.frozen
module, 856
deepmd.tf.model.linear
module, 858
deepmd.tf.model.model
module, 860
deepmd.tf.model.model_stat
module, 868
deepmd.tf.model.multi
module, 869
deepmd.tf.model.pairtab
module, 872
deepmd.tf.model.pairwise_dprc
module, 874
deepmd.tf.model.tensor
module, 877
deepmd.tf.nvnmd
module, 898
deepmd.tf.nvnmd.data
module, 898
deepmd.tf.nvnmd.data.data
module, 898
deepmd.tf.nvnmd.descriptor

module, 899
deepmd.tf.nvnmd.descriptor.se_a
module, 899
deepmd.tf.nvnmd.descriptor.se_atten
module, 900
deepmd.tf.nvnmd.entrypoints
module, 901
deepmd.tf.nvnmd.entrypoints.freeze
module, 901
deepmd.tf.nvnmd.entrypoints.mapt
module, 902
deepmd.tf.nvnmd.entrypoints.train
module, 904
deepmd.tf.nvnmd.entrypoints.wrap
module, 905
deepmd.tf.nvnmd.fit
module, 910
deepmd.tf.nvnmd.fit.ener
module, 910
deepmd.tf.nvnmd.utils
module, 910
deepmd.tf.nvnmd.utils.argcheck
module, 910
deepmd.tf.nvnmd.utils.config
module, 911
deepmd.tf.nvnmd.utils.encode
module, 913
deepmd.tf.nvnmd.utils.fio
module, 915
deepmd.tf.nvnmd.utils.network
module, 916
deepmd.tf.nvnmd.utils.op
module, 918
deepmd.tf.nvnmd.utils.weight
module, 918
deepmd.tf.op
module, 923
deepmd.tf.op._add_flt_nvnmd_grad
module, 924
deepmd.tf.op._copy_flt_nvnmd_grad
module, 924
deepmd.tf.op._dotmul_flt_nvnmd_grad
module, 924
deepmd.tf.op._flt_nvnmd_grad
module, 925
deepmd.tf.op._gelu
module, 925
deepmd.tf.op._map_flt_nvnmd_grad
module, 926
deepmd.tf.op._matmul_fitnet_nvnmd_grad
module, 926
deepmd.tf.op._matmul_flt2fix_nvnmd
module, 926
deepmd.tf.op._matmul_flt_nvnmd_grad
module, 926
deepmd.tf.op._mul_flt_nvnmd_grad
module, 927
deepmd.tf.op._prod_force_grad
module, 927
deepmd.tf.op._prod_force_se_a_grad
module, 927
deepmd.tf.op._prod_force_se_a_mask_grad
module, 928
deepmd.tf.op._prod_force_se_r_grad
module, 928
deepmd.tf.op._prod_virial_grad
module, 928
deepmd.tf.op._prod_virial_se_a_grad
module, 929
deepmd.tf.op._prod_virial_se_r_grad
module, 929
deepmd.tf.op._quantize_nvnmd_grad
module, 929
deepmd.tf.op._soft_min_force_grad
module, 930
deepmd.tf.op._soft_min_virial_grad
module, 930
deepmd.tf.op._tabulate_grad
module, 930
deepmd.tf.op._tanh4_flt_nvnmd_grad
module, 931
deepmd.tf.train
module, 932
deepmd.tf.train.run_options
module, 932
deepmd.tf.train.trainer
module, 934
deepmd.tf.utils
module, 936
deepmd.tf.utils.argcheck
module, 936
deepmd.tf.utils.batch_size
module, 937
deepmd.tf.utils.compat
module, 938
deepmd.tf.utils.compress
module, 939
deepmd.tf.utils.convert
module, 940
deepmd.tf.utils.data
module, 943
deepmd.tf.utils.data_system
module, 947
deepmd.tf.utils.errors
module, 950
deepmd.tf.utils.finetune
module, 951
deepmd.tf.utils.graph

module, 952
deepmd.tf.utils.learning_rate
module, 958
deepmd.tf.utils.multi_init
module, 959
deepmd.tf.utils.neighbor_stat
module, 960
deepmd.tf.utils.network
module, 962
deepmd.tf.utils.nlist
module, 965
deepmd.tf.utils.pair_tab
module, 965
deepmd.tf.utils.parallel_op
module, 967
deepmd.tf.utils.path
module, 968
deepmd.tf.utils.plugin
module, 974
deepmd.tf.utils.random
module, 975
deepmd.tf.utils.region
module, 977
deepmd.tf.utils.serialization
module, 977
deepmd.tf.utils.sess
module, 978
deepmd.tf.utils.spin
module, 978
deepmd.tf.utils.tabulate
module, 979
deepmd.tf.utils.type_embed
module, 981
deepmd.tf.utils.update_sel
module, 984
deepmd.tf.utils.weight_avg
module, 984
deepmd.utils
module, 1013
deepmd.utils.argcheck
module, 1013
deepmd.utils.argcheck_nvnmd
module, 1019
deepmd.utils.batch_size
module, 1019
deepmd.utils.compat
module, 1022
deepmd.utils.data
module, 1025
deepmd.utils.data_system
module, 1029
deepmd.utils.econf_embd
module, 1034
deepmd.utils.env_mat_stat
module, 1036
deepmd.utils.errors
module, 1038
deepmd.utils.finetune
module, 1038
deepmd.utils.hostlist
module, 1039
deepmd.utils.model_stat
module, 1040
deepmd.utils.neighbor_stat
module, 1041
deepmd.utils.out_stat
module, 1042
deepmd.utils.pair_tab
module, 1043
deepmd.utils.path
module, 1045
deepmd.utils.plugin
module, 1050
deepmd.utils.random
module, 1052
deepmd.utils.spin
module, 1054
deepmd.utils.summary
module, 1055
deepmd.utils.update_sel
module, 1056
deepmd.utils.version
module, 1057
deepmd.utils.weight_avg
module, 1057
deepmd::AtomMap (C++ class), 1136
deepmd::AtomMap::AtomMap (C++ function), 1136
deepmd::AtomMap::backward (C++ function), 1136
deepmd::AtomMap::forward (C++ function), 1136
deepmd::AtomMap::get_bkw_map (C++ function), 1136
deepmd::AtomMap::get_fwd_map (C++ function), 1136
deepmd::AtomMap::get_type (C++ function), 1136
deepmd::build_nlist_cpu (C++ function), 1285
deepmd::build_nlist_gpu (C++ function), 1285
deepmd::check_status (C++ function), 1175
deepmd::compute_cell_info (C++ function), 1285
deepmd::convert_nlist (C++ function), 1285
deepmd::convert_nlist_gpu_device (C++ function), 1286
deepmd::convert_pbtxt_to_pb (C++ function), 1176
deepmd::convert_to_inter_cpu (C++ function), 1286
deepmd::convert_to_inter_gpu (C++ function), 1286

deepmd::convert_to_phys_cpu (C++ function), 1287
deepmd::convert_to_phys_gpu (C++ function), 1287
deepmd::copy_coord_cpu (C++ function), 1287
deepmd::copy_coord_gpu (C++ function), 1287
deepmd::cos_switch (C++ function), 1288
deepmd::cprod (C++ function), 1288
deepmd::cum_sum (C++ function), 1288
deepmd::deepmd_exception (C++ struct), 1134, 1269
deepmd::deepmd_exception::deepmd_exception (C++ function), 1269
deepmd::deepmd_exception_oom (C++ struct), 1270
deepmd::deepmd_exception_oom::deepmd_exception_oom (C++ function), 1270
deepmd::DeePot (C++ class), 1137
deepmd::DeePot::~DeePot (C++ function), 1142
deepmd::DeePot::compute (C++ function), 1137--1140
deepmd::DeePot::compute_mixed_type (C++ function), 1140, 1141
deepmd::DeePot::cutoff (C++ function), 1142
deepmd::DeePot::DeePot (C++ function), 1142
deepmd::DeePot::dim_aparam (C++ function), 1142
deepmd::DeePot::dim_fparam (C++ function), 1142
deepmd::DeePot::get_type_map (C++ function), 1143
deepmd::DeePot::init (C++ function), 1142
deepmd::DeePot::is_aparam_nall (C++ function), 1143
deepmd::DeePot::numb_types (C++ function), 1142
deepmd::DeePot::numb_types_spin (C++ function), 1142
deepmd::DeePot::print_summary (C++ function), 1142
deepmd::DeePotBase (C++ class), 1143
deepmd::DeePotBase::~DeePotBase (C++ function), 1146
deepmd::DeePotBase::computew (C++ function), 1143--1145
deepmd::DeePotBase::computew_mixed_type (C++ function), 1145, 1146
deepmd::DeePotBase::cutoff (C++ function), 1146
deepmd::DeePotBase::DeePotBase (C++ function), 1146
deepmd::DeePotBase::dim_aparam (C++ function), 1147
deepmd::DeePotBase::dim_fparam (C++ function), 1147
deepmd::DeePotBase::get_type_map (C++ function), 1147
deepmd::DeePotBase::init (C++ function), 1146
deepmd::DeePotBase::is_aparam_nall (C++ function), 1147
deepmd::DeePotBase::numb_types (C++ function), 1146
deepmd::DeePotBase::numb_types_spin (C++ function), 1146
deepmd::DeePotModelDevi (C++ class), 1147
deepmd::DeePotModelDevi::~DeePotModelDevi (C++ function), 1147
deepmd::DeePotModelDevi::compute (C++ function), 1148, 1149
deepmd::DeePotModelDevi::compute_avg (C++ function), 1151
deepmd::DeePotModelDevi::compute_relative_std (C++ function), 1151
deepmd::DeePotModelDevi::compute_relative_std_f (C++ function), 1152
deepmd::DeePotModelDevi::compute_std (C++ function), 1151
deepmd::DeePotModelDevi::compute_std_e (C++ function), 1151
deepmd::DeePotModelDevi::compute_std_f (C++ function), 1152
deepmd::DeePotModelDevi::cutoff (C++ function), 1150
deepmd::DeePotModelDevi::DeePotModelDevi (C++ function), 1147
deepmd::DeePotModelDevi::dim_aparam (C++ function), 1151
deepmd::DeePotModelDevi::dim_fparam (C++ function), 1150
deepmd::DeePotModelDevi::init (C++ function), 1148
deepmd::DeePotModelDevi::is_aparam_nall (C++ function), 1152
deepmd::DeePotModelDevi::numb_types (C++ function), 1150
deepmd::DeePotModelDevi::numb_types_spin (C++ function), 1150
deepmd::DeePotPT (C++ class), 1153
deepmd::DeePotPT::~DeePotPT (C++ function), 1153
deepmd::DeePotPT::computew (C++ function), 1154, 1155
deepmd::DeePotPT::computew_mixed_type (C++ function), 1155, 1156
deepmd::DeePotPT::cutoff (C++ function), 1153
deepmd::DeePotPT::DeePotPT (C++ function), 1153
deepmd::DeePotPT::dim_aparam (C++ function), 1154

deepmd::DeepPotPT::dim_fparam (C++ function), 1153
deepmd::DeepPotPT::get_type_map (C++ function), 1154
deepmd::DeepPotPT::init (C++ function), 1153
deepmd::DeepPotPT::is_aparam_nall (C++ function), 1154
deepmd::DeepPotPT::numb_types (C++ function), 1153
deepmd::DeepPotPT::numb_types_spin (C++ function), 1153
deepmd::DeepPotTF (C++ class), 1157
deepmd::DeepPotTF::~DeepPotTF (C++ function), 1157
deepmd::DeepPotTF::computew (C++ function), 1158--1160
deepmd::DeepPotTF::computew_mixed_type (C++ function), 1160
deepmd::DeepPotTF::cutoff (C++ function), 1157
deepmd::DeepPotTF::DeepPotTF (C++ function), 1157
deepmd::DeepPotTF::dim_aparam (C++ function), 1158
deepmd::DeepPotTF::dim_fparam (C++ function), 1158
deepmd::DeepPotTF::get_type_map (C++ function), 1158
deepmd::DeepPotTF::init (C++ function), 1157
deepmd::DeepPotTF::is_aparam_nall (C++ function), 1158
deepmd::DeepPotTF::numb_types (C++ function), 1157
deepmd::DeepPotTF::numb_types_spin (C++ function), 1158
deepmd::DeepTensor (C++ class), 1161
deepmd::DeepTensor::~DeepTensor (C++ function), 1161
deepmd::DeepTensor::compute (C++ function), 1161--1163
deepmd::DeepTensor::cutoff (C++ function), 1163
deepmd::DeepTensor::DeepTensor (C++ function), 1161
deepmd::DeepTensor::get_type_map (C++ function), 1164
deepmd::DeepTensor::init (C++ function), 1161
deepmd::DeepTensor::numb_types (C++ function), 1164
deepmd::DeepTensor::output_dim (C++ function), 1164
deepmd::DeepTensor::print_summary (C++ function), 1161
deepmd::DeepTensor::sel_types (C++ function), 1164
deepmd::DeepTensorBase (C++ class), 1164
deepmd::DeepTensorBase::~DeepTensorBase (C++ function), 1166
deepmd::DeepTensorBase::computew (C++ function), 1165, 1166
deepmd::DeepTensorBase::cutoff (C++ function), 1166
deepmd::DeepTensorBase::DeepTensorBase (C++ function), 1166
deepmd::DeepTensorBase::get_type_map (C++ function), 1167
deepmd::DeepTensorBase::init (C++ function), 1166
deepmd::DeepTensorBase::numb_types (C++ function), 1166
deepmd::DeepTensorBase::output_dim (C++ function), 1166
deepmd::DeepTensorBase::sel_types (C++ function), 1167
deepmd::DeepTensorTF (C++ class), 1167
deepmd::DeepTensorTF::~DeepTensorTF (C++ function), 1169
deepmd::DeepTensorTF::computew (C++ function), 1167, 1168
deepmd::DeepTensorTF::cutoff (C++ function), 1169
deepmd::DeepTensorTF::DeepTensorTF (C++ function), 1169
deepmd::DeepTensorTF::get_type_map (C++ function), 1169
deepmd::DeepTensorTF::init (C++ function), 1169
deepmd::DeepTensorTF::numb_types (C++ function), 1169
deepmd::DeepTensorTF::output_dim (C++ function), 1169
deepmd::DeepTensorTF::sel_types (C++ function), 1169
deepmd::delete_device_memory (C++ function), 1289
deepmd::DipoleChargeModifier (C++ class), 1170
deepmd::DipoleChargeModifier::~DipoleChargeModifier (C++ function), 1170
deepmd::DipoleChargeModifier::compute (C++ function), 1170
deepmd::DipoleChargeModifier::cutoff (C++ function), 1171
deepmd::DipoleChargeModifier::DipoleChargeModifier (C++ function), 1170
deepmd::DipoleChargeModifier::init (C++ function), 1170
deepmd::DipoleChargeModifier::numb_types (C++ function), 1171
deepmd::DipoleChargeModifier::print_summary

(C++ function), 1170
deepmd::DipoleChargeModifier::sel_types (C++ function), 1171
deepmd::DipoleChargeModifierBase (C++ class), 1171
deepmd::DipoleChargeModifierBase::~DipoleChargeModifierBase (C++ function), 1172
deepmd::DipoleChargeModifierBase::computew (C++ function), 1172
deepmd::DipoleChargeModifierBase::cutoff (C++ function), 1173
deepmd::DipoleChargeModifierBase::DipoleChargeModifierBase (C++ function), 1172
deepmd::DipoleChargeModifierBase::init (C++ function), 1172
deepmd::DipoleChargeModifierBase::numb_types (C++ function), 1173
deepmd::DipoleChargeModifierBase::sel_types (C++ function), 1173
deepmd::DipoleChargeModifierTF (C++ class), 1173
deepmd::DipoleChargeModifierTF::~DipoleChargeModifierTF (C++ function), 1174
deepmd::DipoleChargeModifierTF::computew (C++ function), 1174
deepmd::DipoleChargeModifierTF::cutoff (C++ function), 1174
deepmd::DipoleChargeModifierTF::DipoleChargeModifierTF (C++ function), 1173
deepmd::DipoleChargeModifierTF::init (C++ function), 1174
deepmd::DipoleChargeModifierTF::numb_types (C++ function), 1174
deepmd::DipoleChargeModifierTF::sel_types (C++ function), 1174
deepmd::dot1 (C++ function), 1289
deepmd::dot2 (C++ function), 1289
deepmd::dot3 (C++ function), 1289
deepmd::dot4 (C++ function), 1289
deepmd::dotmv3 (C++ function), 1290
deepmd::DPBackend (C++ enum), 1175
deepmd::DPBackend::Paddle (C++ enumerator), 1175
deepmd::DPBackend::PyTorch (C++ enumerator), 1175
deepmd::DPBackend::TensorFlow (C++ enumerator), 1175
deepmd::DPBackend::Unknown (C++ enumerator), 1175
deepmd::DPGetDeviceCount (C++ function), 1290
deepmd::dprc_pairwise_map_cpu (C++ function), 1290
deepmd::DPSetDevice (C++ function), 1291
deepmd::ElectrostaticConvertion (C++ member), 1320
deepmd::ENERGYTYPE (C++ type), 1184
deepmd::env_mat_a_cpu (C++ function), 1291
deepmd::env_mat_a_nvnmd_quantize_cpu (C++ function), 1291
deepmd::env_mat_nbor_update (C++ function), 1292
deepmd::env_mat_r_cpu (C++ function), 1292
deepmd::ewald_recp (C++ function), 1292
deepmd::EwaldParameters (C++ struct), 1270
deepmd::EwaldParameters::beta (C++ member), 1270
deepmd::EwaldParameters::rcut (C++ member), 1270
deepmd::EwaldParameters::spacing (C++ member), 1270
deepmd::filter_ftype_gpu (C++ function), 1292
deepmd::format_nbor_list_gpu (C++ function), 1293
deepmd::format_nlist_cpu (C++ function), 1293
deepmd::free_nlist_gpu_device (C++ function), 1293
deepmd::gelu_cpu (C++ function), 1294
deepmd::gelu_gpu (C++ function), 1294
deepmd::gelu_grad_cpu (C++ function), 1294
deepmd::gelu_grad_gpu (C++ function), 1294
deepmd::gelu_grad_grad_cpu (C++ function), 1294
deepmd::gelu_grad_grad_gpu (C++ function), 1295
deepmd::get_env_nthreads (C++ function), 1176
deepmd::group_atoms_cpu (C++ function), 1295
deepmd::hpp::convert_nlist (C++ function), 1216
deepmd::hpp::convert_pbtxt_to_pb (C++ function), 1217
deepmd::hpp::deepmd_exception (C++ struct), 1187
deepmd::hpp::deepmd_exception::deepmd_exception (C++ function), 1187
deepmd::hpp::DeepPot (C++ class), 1191
deepmd::hpp::DeepPot::~DeepPot (C++ function), 1191
deepmd::hpp::DeepPot::compute (C++ function), 1191--1193
deepmd::hpp::DeepPot::compute_mixed_type (C++ function), 1194, 1195
deepmd::hpp::DeepPot::cutoff (C++ function), 1196
deepmd::hpp::DeepPot::DeepPot (C++ function), 1191
deepmd::hpp::DeepPot::dim_aparam (C++ function), 1196
deepmd::hpp::DeepPot::dim_fparam (C++ func-

tion), 1196
deepmd::hpp::DeepPot::get_type_map (C++ function), 1196
deepmd::hpp::DeepPot::init (C++ function), 1191
deepmd::hpp::DeepPot::numb_types (C++ function), 1196
deepmd::hpp::DeepPot::numb_types_spin (C++ function), 1196
deepmd::hpp::DeepPot::print_summary (C++ function), 1196
deepmd::hpp::DeepPotModelDevi (C++ class), 1197
deepmd::hpp::DeepPotModelDevi::~DeepPotModelDevi (C++ function), 1197
deepmd::hpp::DeepPotModelDevi::compute (C++ function), 1197--1199
deepmd::hpp::DeepPotModelDevi::compute_avg (C++ function), 1200
deepmd::hpp::DeepPotModelDevi::compute_relative_std (C++ function), 1200
deepmd::hpp::DeepPotModelDevi::compute_relative_std_f (C++ function), 1201
deepmd::hpp::DeepPotModelDevi::compute_std (C++ function), 1200
deepmd::hpp::DeepPotModelDevi::compute_std_f (C++ function), 1201
deepmd::hpp::DeepPotModelDevi::cutoff (C++ function), 1200
deepmd::hpp::DeepPotModelDevi::DeepPotModelDevi (C++ function), 1197
deepmd::hpp::DeepPotModelDevi::dim_aparam (C++ function), 1200
deepmd::hpp::DeepPotModelDevi::dim_fparam (C++ function), 1200
deepmd::hpp::DeepPotModelDevi::init (C++ function), 1197
deepmd::hpp::DeepPotModelDevi::numb_types (C++ function), 1200
deepmd::hpp::DeepPotModelDevi::numb_types_spin (C++ function), 1200
deepmd::hpp::DeepTensor (C++ class), 1201
deepmd::hpp::DeepTensor::~DeepTensor (C++ function), 1202
deepmd::hpp::DeepTensor::compute (C++ function), 1202--1204
deepmd::hpp::DeepTensor::cutoff (C++ function), 1204
deepmd::hpp::DeepTensor::DeepTensor (C++ function), 1202
deepmd::hpp::DeepTensor::get_type_map (C++ function), 1205
deepmd::hpp::DeepTensor::init (C++ function), 1202
deepmd::hpp::DeepTensor::numb_types (C++ function), 1204
deepmd::hpp::DeepTensor::output_dim (C++ function), 1204
deepmd::hpp::DeepTensor::print_summary (C++ function), 1205
deepmd::hpp::DeepTensor::sel_types (C++ function), 1204
deepmd::hpp::DipoleChargeModifier (C++ class), 1205
deepmd::hpp::DipoleChargeModifier::~DipoleChargeModifier (C++ function), 1205
deepmd::hpp::DipoleChargeModifier::compute (C++ function), 1205
deepmd::hpp::DipoleChargeModifier::cutoff (C++ function), 1206
deepmd::hpp::DipoleChargeModifier::DipoleChargeModifier (C++ function), 1205
deepmd::hpp::DipoleChargeModifier::init (C++ function), 1205
deepmd::hpp::DipoleChargeModifier::numb_types (C++ function), 1206
deepmd::hpp::DipoleChargeModifier::print_summary (C++ function), 1206
deepmd::hpp::DipoleChargeModifier::sel_types (C++ function), 1206
deepmd::hpp::InputNlist (C++ struct), 1187
deepmd::hpp::InputNlist::~InputNlist (C++ function), 1187
deepmd::hpp::InputNlist::firstneigh (C++ member), 1188
deepmd::hpp::InputNlist::ilist (C++ member), 1188
deepmd::hpp::InputNlist::InputNlist (C++ function), 1187
deepmd::hpp::InputNlist::inum (C++ member), 1188
deepmd::hpp::InputNlist::nl (C++ member), 1188
deepmd::hpp::InputNlist::numneigh (C++ member), 1188
deepmd::hpp::read_file_to_string (C++ function), 1217
deepmd::hpp::select_by_type (C++ function), 1217
deepmd::hpp::select_map (C++ function), 1218
deepmd::init_region_cpu (C++ function), 1295
deepmd::InputNlist (C++ struct), 1271
deepmd::InputNlist::~InputNlist (C++ function), 1271
deepmd::InputNlist::firstneigh (C++ member), 1271
deepmd::InputNlist::firstrecv (C++ member), 1272
deepmd::InputNlist::ilist (C++ member), 1271

deepmd::InputNlist::InputNlist (C++ function), 1271
deepmd::InputNlist::inum (C++ member), 1271
deepmd::InputNlist::nswap (C++ member), 1271
deepmd::InputNlist::numneigh (C++ member), 1271
deepmd::InputNlist::recvnum (C++ member), 1272
deepmd::InputNlist::recvproc (C++ member), 1272
deepmd::InputNlist::sendlist (C++ member), 1272
deepmd::InputNlist::sendnum (C++ member), 1271
deepmd::InputNlist::sendproc (C++ member), 1272
deepmd::InputNlist::world (C++ member), 1272
deepmd::invsqrt (C++ function), 1296
deepmd::invsqrt<double> (C++ function), 1296
deepmd::invsqrt<float> (C++ function), 1296
deepmd::load_op_library (C++ function), 1176
deepmd::malloc_device_memory (C++ function), 1296, 1297
deepmd::malloc_device_memory_sync (C++ function), 1297
deepmd::map_aparam_cpu (C++ function), 1298
deepmd::max_numneigh (C++ function), 1298
deepmd::memcpy_device_to_host (C++ function), 1298, 1299
deepmd::memcpy_host_to_device (C++ function), 1299
deepmd::memset_device_memory (C++ function), 1299
deepmd::model_compatable (C++ function), 1177
deepmd::name_prefix (C++ function), 1177
deepmd::neighbor_stat_gpu (C++ function), 1300
deepmd::NeighborListData (C++ struct), 1134
deepmd::NeighborListData::copy_from_nlist (C++ function), 1135
deepmd::NeighborListData::firstneigh (C++ member), 1135
deepmd::NeighborListData::ilist (C++ member), 1135
deepmd::NeighborListData::jlist (C++ member), 1135
deepmd::NeighborListData::make_inlist (C++ function), 1135
deepmd::NeighborListData::numneigh (C++ member), 1135
deepmd::NeighborListData::padding (C++ function), 1135
deepmd::NeighborListData::shuffle (C++ function), 1135
deepmd::NeighborListData::shuffle_exclude_empty (C++ function), 1135
deepmd::normalize_coord_cpu (C++ function), 1300
deepmd::normalize_coord_gpu (C++ function), 1300
deepmd::pair_tab_cpu (C++ function), 1300
deepmd::print_summary (C++ function), 1177
deepmd::prod_env_mat_a_cpu (C++ function), 1301
deepmd::prod_env_mat_a_gpu (C++ function), 1301
deepmd::prod_env_mat_a_nvnmd_quantize_cpu (C++ function), 1301
deepmd::prod_env_mat_r_cpu (C++ function), 1302
deepmd::prod_env_mat_r_gpu (C++ function), 1302
deepmd::prod_force_a_cpu (C++ function), 1302, 1303
deepmd::prod_force_a_gpu (C++ function), 1304
deepmd::prod_force_grad_a_cpu (C++ function), 1304
deepmd::prod_force_grad_a_gpu (C++ function), 1304
deepmd::prod_force_grad_r_cpu (C++ function), 1304
deepmd::prod_force_grad_r_gpu (C++ function), 1305
deepmd::prod_force_r_cpu (C++ function), 1305
deepmd::prod_force_r_gpu (C++ function), 1305
deepmd::prod_virial_a_cpu (C++ function), 1305
deepmd::prod_virial_a_gpu (C++ function), 1306
deepmd::prod_virial_grad_a_cpu (C++ function), 1306
deepmd::prod_virial_grad_a_gpu (C++ function), 1306
deepmd::prod_virial_grad_r_cpu (C++ function), 1306
deepmd::prod_virial_grad_r_gpu (C++ function), 1307
deepmd::prod_virial_r_cpu (C++ function), 1307
deepmd::prod_virial_r_gpu (C++ function), 1307
deepmd::read_file_to_string (C++ function), 1177
deepmd::Region (C++ struct), 1272
deepmd::Region::~Region (C++ function), 1272
deepmd::Region::boxt (C++ member), 1273
deepmd::Region::rec_boxt (C++ member), 1273
deepmd::Region::Region (C++ function), 1272
deepmd::select_by_type (C++ function), 1178
deepmd::select_map (C++ function), 1178, 1179
deepmd::select_map_inv (C++ function), 1179, 1180
deepmd::select_real_atoms (C++ function), 1180

deepmd::select_real_atoms_coord (C++ function), 1180
deepmd::session_get_dtype (C++ function), 1180
deepmd::session_get_scalar (C++ function), 1181
deepmd::session_get_vector (C++ function), 1181
deepmd::session_input_tensors (C++ function), 1182, 1183
deepmd::session_input_tensors_mixed_type (C++ function), 1183
deepmd::soft_min_switch_cpu (C++ function), 1307
deepmd::soft_min_switch_force_cpu (C++ function), 1308
deepmd::soft_min_switch_force_grad_cpu (C++ function), 1308
deepmd::soft_min_switch_virial_cpu (C++ function), 1308
deepmd::soft_min_switch_virial_grad_cpu (C++ function), 1308
deepmd::spline3_switch (C++ function), 1309
deepmd::spline5_switch (C++ function), 1309
deepmd::STRINGTYPE (C++ type), 1184
deepmd::tabulate_fusion_se_a_cpu (C++ function), 1309
deepmd::tabulate_fusion_se_a_gpu (C++ function), 1309
deepmd::tabulate_fusion_se_a_grad_cpu (C++ function), 1310
deepmd::tabulate_fusion_se_a_grad_gpu (C++ function), 1310
deepmd::tabulate_fusion_se_a_grad_grad_cpu (C++ function), 1310
deepmd::tabulate_fusion_se_a_grad_grad_gpu (C++ function), 1311
deepmd::tabulate_fusion_se_r_cpu (C++ function), 1311
deepmd::tabulate_fusion_se_r_gpu (C++ function), 1311
deepmd::tabulate_fusion_se_r_grad_cpu (C++ function), 1312
deepmd::tabulate_fusion_se_r_grad_gpu (C++ function), 1312
deepmd::tabulate_fusion_se_r_grad_grad_cpu (C++ function), 1312
deepmd::tabulate_fusion_se_r_grad_grad_gpu (C++ function), 1312
deepmd::tabulate_fusion_se_t_cpu (C++ function), 1313
deepmd::tabulate_fusion_se_t_gpu (C++ function), 1313
deepmd::tabulate_fusion_se_t_grad_cpu (C++ function), 1313
deepmd::tabulate_fusion_se_t_grad_gpu (C++ function), 1314
deepmd::tabulate_fusion_se_t_grad_grad_cpu (C++ function), 1314
deepmd::tabulate_fusion_se_t_grad_grad_gpu (C++ function), 1314
deepmd::test_encoding_decoding_nbor_info_gpu (C++ function), 1315
deepmd::tf_exception (C++ struct), 1135
deepmd::tf_exception::tf_exception (C++ function), 1136
deepmd::use_nei_info_cpu (C++ function), 1315
deepmd::use_nei_info_gpu (C++ function), 1315
deepmd::use_nlist_map (C++ function), 1315
deepmd::volume_cpu (C++ function), 1316
deepmd::volume_gpu (C++ function), 1316
DeepmdData (class in deepmd.tf.utils), 985
DeepmdData (class in deepmd.tf.utils.data), 943
DeepmdData (class in deepmd.utils.data), 1025
DeepmdDataSetForLoader (class in deepmd.pt.utils.dataset), 616
DeepmdDataSystem (class in deepmd.tf.utils), 988
DeepmdDataSystem (class in deepmd.tf.utils.data_system), 947
DeepmdDataSystem (class in deepmd.utils.data_system), 1030
DeepPolar (class in deepmd.infer.deep_polar), 395
DeepPolar (class in deepmd.tf.infer), 831
DeepPolar (class in deepmd.tf.infer.deep_polar), 809
DeepPot (class in deepmd.infer), 408
DeepPot (class in deepmd.infer.deep_pot), 397
DeepPot (class in deepmd.tf.infer), 831
DeepPot (class in deepmd.tf.infer.deep_pot), 810
DeepPotential() (in module deepmd), 1066
DeepPotential() (in module deepmd.infer), 411
DeepPotential() (in module deepmd.tf), 1011
DeepPotential() (in module deepmd.tf.infer), 820
DeepTensor (class in deepmd.infer.deep_tensor), 399
DeepTensor (class in deepmd.tf.infer.deep_tensor), 812
DeepWFC (class in deepmd.infer.deep_wfc), 401
DeepWFC (class in deepmd.tf.infer), 833
DeepWFC (class in deepmd.tf.infer.deep_wfc), 814
default_mesh (deepmd.tf.utils.data_system.DeepmdDataSystem property), 947
default_mesh (deepmd.tf.utils.DeepmdDataSystem property), 988
default_mesh (deepmd.utils.data_system.DeepmdDataSystem property), 1030
DEFAULT_PRECISION (in module deepmd.dpmodel), 380
DEFAULT_PRECISION (in module deepmd.dpmodel.common), 374

DEFAULT_PRECISION (in module deepmd.pt.utils.env), 618
default_tf_session_config (in module deepmd.tf.env), 1003
DenoiseLoss (class in deepmd.pt.loss), 427
DenoiseLoss (class in deepmd.pt.loss.denoise), 419
DenoiseNet (class in deepmd.pt.model.task), 597
DenoiseNet (class in deepmd.pt.model.task.denoise), 579
deprecate_argument_extra_check() (in module deepmd.utils.argcheck), 1016
deprecate_numb_test() (in module deepmd.tf.utils.compat), 939
deprecate_numb_test() (in module deepmd.utils.compat), 1024
DeprecateAction (class in deepmd.main), 1065
DERV_C (deepmd.dpmodel.output_def.OutputVariableCategory attribute), 376
DERV_C (deepmd.dpmodel.output_def.OutputVariableOperation attribute), 376
DERV_C_MAG (deepmd.dpmodel.output_def.OutputVariableCategory attribute), 376
DERV_C_REDU (deepmd.dpmodel.output_def.OutputVariableCategory attribute), 376
DERV_R (deepmd.dpmodel.output_def.OutputVariableCategory attribute), 376
DERV_R (deepmd.dpmodel.output_def.OutputVariableOperation attribute), 376
DERV_R_DERV_R (deepmd.dpmodel.output_def.OutputVariableCategory attribute), 376
DERV_R_MAG (deepmd.dpmodel.output_def.OutputVariableCategory attribute), 376
Descriptor (class in deepmd.tf.descriptor), 698
Descriptor (class in deepmd.tf.descriptor.descriptor), 640
descriptor:
    model[multi]/descriptor (Argument), 145
    model[standard]/descriptor (Argument), 115
DescriptorBlock (class in deepmd.pt.model.descriptor), 505
DescriptorBlock (class in deepmd.pt.model.descriptor.descriptor), 468
Descrpt() (in module deepmd.tf.env.op_module), 1071
descrpt() (in module deepmd.tf.env.op_module), 1096
descrpt2r4() (in module deepmd.tf.nvnmd.descriptor.se_a), 899
descrpt2r4() (in module deepmd.tf.nvnmd.descriptor.se_atten), 900
descrpt_args_plugin (in module deepmd.utils.argcheck), 1017
descrpt_dpa2_args() (in module deepmd.utils.argcheck), 1017
descrpt_hybrid_args() (in module deepmd.utils.argcheck), 1017
descrpt_local_frame_args() (in module deepmd.utils.argcheck), 1017
descrpt_norot() (in module deepmd.tf.env.op_module), 1097
descrpt_se_a() (in module deepmd.tf.env.op_module), 1097
descrpt_se_a_args() (in module deepmd.utils.argcheck), 1017
descrpt_se_a_ebd_v2_args() (in module deepmd.utils.argcheck), 1017
descrpt_se_a_ef() (in module deepmd.tf.env.op_module), 1098
descrpt_se_a_ef_para() (in module deepmd.tf.env.op_module), 1098
descrpt_se_a_ef_vert() (in module deepmd.tf.env.op_module), 1099
descrpt_se_a_mask() (in module deepmd.tf.env.op_module), 1100
descrpt_se_a_mask_args() (in module deepmd.utils.argcheck), 1017
descrpt_se_a_tpe_args() (in module deepmd.utils.argcheck), 1017
descrpt_se_atten_args() (in module deepmd.utils.argcheck), 1017
descrpt_se_atten_common_args() (in module deepmd.utils.argcheck), 1017
descrpt_se_atten_v2_args() (in module deepmd.utils.argcheck), 1017
descrpt_se_r() (in module deepmd.tf.env.op_module), 1100
descrpt_se_r_args() (in module deepmd.utils.argcheck), 1017
descrpt_se_t_args() (in module deepmd.utils.argcheck), 1017
descrpt_variant_type_args() (in module deepmd.utils.argcheck), 1017
DescrptBlockRepformers (class in deepmd.pt.model.descriptor), 520
DescrptBlockRepformers (class in deepmd.pt.model.descriptor.repformers), 490
DescrptBlockSeA (class in deepmd.pt.model.descriptor), 522
DescrptBlockSeA (class in deepmd.pt.model.descriptor.se_a), 494
DescrptBlockSeAtten (class in deepmd.pt.model.descriptor), 507
DescrptBlockSeAtten (class in deepmd.pt.model.descriptor.se_atten), 496

DescrptDPA1 (class in deepmd.dpmodel.descriptor), 297
DescrptDPA1 (class in deepmd.dpmodel.descriptor.dpa1), 281
DescrptDPA1 (class in deepmd.pt.model.descriptor), 509
DescrptDPA1 (class in deepmd.pt.model.descriptor.dpa1), 470
DescrptDPA1Compat (class in deepmd.tf.descriptor.se_atten), 684
DescrptDPA2 (class in deepmd.pt.model.descriptor), 514
DescrptDPA2 (class in deepmd.pt.model.descriptor.dpa2), 475
DescrptGaussianLcc (class in deepmd.pt.model.descriptor), 517
DescrptGaussianLcc (class in deepmd.pt.model.descriptor.gaussian_lcc), 479
DescrptHybrid (class in deepmd.dpmodel.descriptor), 302
DescrptHybrid (class in deepmd.dpmodel.descriptor.hybrid), 288
DescrptHybrid (class in deepmd.pt.model.descriptor), 518
DescrptHybrid (class in deepmd.pt.model.descriptor.hybrid), 481
DescrptHybrid (class in deepmd.tf.descriptor), 705
DescrptHybrid (class in deepmd.tf.descriptor.hybrid), 647
DescrptLocFrame (class in deepmd.tf.descriptor), 710
DescrptLocFrame (class in deepmd.tf.descriptor.loc_frame), 653
DescrptNorot() (in module deepmd.tf.env.op_module), 1071
DescrptSe (class in deepmd.tf.descriptor.se), 656
DescrptSeA (class in deepmd.dpmodel.descriptor), 304
DescrptSeA (class in deepmd.dpmodel.descriptor.se_e2_a), 291
DescrptSeA (class in deepmd.pt.model.descriptor), 524
DescrptSeA (class in deepmd.pt.model.descriptor.se_a), 492
DescrptSeA (class in deepmd.tf.descriptor), 713
DescrptSeA (class in deepmd.tf.descriptor.se_a), 659
DescrptSeA() (in module deepmd.tf.env.op_module), 1072
DescrptSeAEbd (class in deepmd.tf.descriptor), 719
DescrptSeAEbd (class in deepmd.tf.descriptor.se_a_ebd), 665
DescrptSeAEbdV2 (class in deepmd.tf.descriptor), 721
DescrptSeAEbdV2 (class in deepmd.tf.descriptor.se_a_ebd_v2), 668
DescrptSeAEf (class in deepmd.tf.descriptor), 722
DescrptSeAEf (class in deepmd.tf.descriptor.se_a_ef), 668
DescrptSeAEf() (in module deepmd.tf.env.op_module), 1072
DescrptSeAEfLower (class in deepmd.tf.descriptor), 725
DescrptSeAEfLower (class in deepmd.tf.descriptor.se_a_ef), 671
DescrptSeAEfPara() (in module deepmd.tf.env.op_module), 1073
DescrptSeAEfVert() (in module deepmd.tf.env.op_module), 1073
DescrptSeAMask (class in deepmd.tf.descriptor), 726
DescrptSeAMask (class in deepmd.tf.descriptor.se_a_mask), 673
DescrptSeAMask() (in module deepmd.tf.env.op_module), 1074
DescrptSeAtten (class in deepmd.tf.descriptor), 729
DescrptSeAtten (class in deepmd.tf.descriptor.se_atten), 676
DescrptSeAttenV2 (class in deepmd.tf.descriptor), 736
DescrptSeAttenV2 (class in deepmd.tf.descriptor.se_atten_v2), 688
DescrptSeR (class in deepmd.dpmodel.descriptor), 307
DescrptSeR (class in deepmd.dpmodel.descriptor.se_r), 294
DescrptSeR (class in deepmd.pt.model.descriptor), 526
DescrptSeR (class in deepmd.pt.model.descriptor.se_r), 502
DescrptSeR (class in deepmd.tf.descriptor), 738
DescrptSeR (class in deepmd.tf.descriptor.se_r), 689
DescrptSeR() (in module deepmd.tf.env.op_module), 1074
DescrptSeRGPUExecuteFunctor (C++ struct), 1273
DescrptSeRGPUExecuteFunctor::operator() (C++ function), 1273
DescrptSeT (class in deepmd.tf.descriptor), 742
DescrptSeT (class in deepmd.tf.descriptor.se_t), 694
deserialize() (deepmd.dpmodel.atomic_model.base_atomic_model class method), 262
deserialize() (deepmd.dpmodel.atomic_model.BaseAtomicModel class method), 273
deserialize() (deepmd.dpmodel.atomic_model.dp_atomic_model.D class method), 264
deserialize() (deepmd.dpmodel.atomic_model.DPAtomicModel class method), 274
deserialize() (deepmd.dpmodel.atomic_model.DPZBLLinearEnerg

class method), 275
deserialize() (deepmd.dpmodel.atomic_model.linear_atomic_model.DPZBLLinearEnergyAtomicModel class method), 267
deserialize() (deepmd.dpmodel.atomic_model.linear_atomic_model.LinearEnergyAtomicModel class method), 266
deserialize() (deepmd.dpmodel.atomic_model.LinearEnergyAtomicModel class method), 277
deserialize() (deepmd.dpmodel.atomic_model.pairtab_atomic_model.PairTabAtomicModel class method), 269
deserialize() (deepmd.dpmodel.atomic_model.PairTabAtomicModel class method), 278
deserialize() (deepmd.dpmodel.descriptor.DescrptDPA1 class method), 302
deserialize() (deepmd.dpmodel.descriptor.DescrptHybrid class method), 304
deserialize() (deepmd.dpmodel.descriptor.DescrptSeA class method), 307
deserialize() (deepmd.dpmodel.descriptor.DescrptSeR class method), 309
deserialize() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 class method), 286
deserialize() (deepmd.dpmodel.descriptor.dpa1.GatedAttentionLayer class method), 288
deserialize() (deepmd.dpmodel.descriptor.dpa1.NeighborGatedAttention class method), 287
deserialize() (deepmd.dpmodel.descriptor.dpa1.NeighborGatedAttentionLayer class method), 287
deserialize() (deepmd.dpmodel.descriptor.hybrid.DescrptHybrid class method), 289
deserialize() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA class method), 294
deserialize() (deepmd.dpmodel.descriptor.se_r.DescrptSeR class method), 297
deserialize() (deepmd.dpmodel.fitting.dipole_fitting.DipoleFitting class method), 312
deserialize() (deepmd.dpmodel.fitting.DipoleFitting class method), 328
deserialize() (deepmd.dpmodel.fitting.dos_fitting.DOSFittingNet class method), 314
deserialize() (deepmd.dpmodel.fitting.DOSFittingNet class method), 331
deserialize() (deepmd.dpmodel.fitting.ener_fitting.EnergyFittingNet class method), 317
deserialize() (deepmd.dpmodel.fitting.EnergyFittingNet class method), 333
deserialize() (deepmd.dpmodel.fitting.general_fitting.GeneralFitting class method), 319
deserialize() (deepmd.dpmodel.fitting.invar_fitting.InvarFitting class method), 322
deserialize() (deepmd.dpmodel.fitting.InvarFitting class method), 335
deserialize() (deepmd.dpmodel.fitting.PolarFitting class method), 337
deserialize() (deepmd.dpmodel.fitting.polarizability_fitting.PolarFitting class method), 326
deserialize() (deepmd.dpmodel.model.dp_zbl_model.DPZBLModel class method), 346
deserialize() (deepmd.dpmodel.model.SpinModel class method), 350
deserialize() (deepmd.dpmodel.utils.env_mat.EnvMat class method), 353
deserialize() (deepmd.dpmodel.utils.EnvMat class method), 367
deserialize() (deepmd.dpmodel.utils.NativeLayer class method), 369
deserialize() (deepmd.dpmodel.utils.network.LayerNorm class method), 358
deserialize() (deepmd.dpmodel.utils.network.NativeLayer class method), 357
deserialize() (deepmd.dpmodel.utils.network.NetworkCollection class method), 359
deserialize() (deepmd.dpmodel.utils.NetworkCollection class method), 370
deserialize() (deepmd.dpmodel.utils.type_embed.TypeEmbedNet class method), 365
deserialize() (deepmd.pt.model.atomic_model.base_atomic_model class method), 436
deserialize() (deepmd.pt.model.atomic_model.BaseAtomicModel class method), 453
deserialize() (deepmd.pt.model.atomic_model.dp_atomic_model.D class method), 440
deserialize() (deepmd.pt.model.atomic_model.DPAtomicModel class method), 456
deserialize() (deepmd.pt.model.atomic_model.DPZBLLinearEnerg class method), 458
deserialize() (deepmd.pt.model.atomic_model.linear_atomic_mod class method), 446
deserialize() (deepmd.pt.model.atomic_model.linear_atomic_mod class method), 444
deserialize() (deepmd.pt.model.atomic_model.LinearEnergyAtomi class method), 460
deserialize() (deepmd.pt.model.atomic_model.pairtab_atomic_mo class method), 447
deserialize() (deepmd.pt.model.atomic_model.PairTabAtomicMod class method), 462
deserialize() (deepmd.pt.model.descriptor.DescrptDPA1 class method), 513
deserialize() (deepmd.pt.model.descriptor.DescrptDPA2 class method), 516
deserialize() (deepmd.pt.model.descriptor.DescrptHybrid class method), 520
deserialize() (deepmd.pt.model.descriptor.DescrptSeA class method), 526
deserialize() (deepmd.pt.model.descriptor.DescrptSeR class method), 528
deserialize() (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 class method), 474
deserialize() (deepmd.pt.model.descriptor.dpa2.DescrptDPA2

class method), 478
deserialize() (deepmd.pt.model.descriptor.hybrid.DescrptHybrid class method), 483
deserialize() (deepmd.pt.model.descriptor.se_a.DescrptSeA class method), 494
deserialize() (deepmd.pt.model.descriptor.se_atten.GatedAttentionLayer class method), 502
deserialize() (deepmd.pt.model.descriptor.se_atten.NeighborGatedAttention class method), 499
deserialize() (deepmd.pt.model.descriptor.se_atten.NeighborGatedAttentionLayer class method), 501
deserialize() (deepmd.pt.model.descriptor.se_r.DescrptSeR class method), 504
deserialize() (deepmd.pt.model.model.frozen.FrozenModel class method), 534
deserialize() (deepmd.pt.model.model.FrozenModel class method), 544
deserialize() (deepmd.pt.model.model.spin_model.SpinModel class method), 539
deserialize() (deepmd.pt.model.model.SpinModel class method), 548
deserialize() (deepmd.pt.model.network.layernorm.LayerNorm class method), 550
deserialize() (deepmd.pt.model.network.mlp.MLPLayer class method), 552
deserialize() (deepmd.pt.model.network.network.TypeEmbedNetConsistent class method), 564
deserialize() (deepmd.pt.model.task.dipole.DipoleFittingNet class method), 582
deserialize() (deepmd.pt.model.task.DipoleFittingNet class method), 599
deserialize() (deepmd.pt.model.task.dos.DOSFittingNet class method), 584
deserialize() (deepmd.pt.model.task.ener.EnergyFittingNet class method), 586
deserialize() (deepmd.pt.model.task.ener.EnergyFittingNetDirect method), 587
deserialize() (deepmd.pt.model.task.EnergyFittingNet class method), 601
deserialize() (deepmd.pt.model.task.EnergyFittingNetDirect method), 602
deserialize() (deepmd.pt.model.task.fitting.GeneralFitting class method), 590
deserialize() (deepmd.pt.model.task.invar_fitting.InvarFitting class method), 593
deserialize() (deepmd.pt.model.task.PolarFittingNet class method), 604
deserialize() (deepmd.pt.model.task.polarizability.PolarFittingNet class method), 595
deserialize() (deepmd.tf.descriptor.Descriptor class method), 705
deserialize() (deepmd.tf.descriptor.descriptor.Descriptor class method), 647
deserialize() (deepmd.tf.descriptor.DescrptHybrid class method), 710
deserialize() (deepmd.tf.descriptor.DescrptSeA class method), 719
deserialize() (deepmd.tf.descriptor.DescrptSeAtten class method), 736
deserialize() (deepmd.tf.descriptor.DescrptSeR class method), 741
deserialize() (deepmd.tf.descriptor.hybrid.DescrptHybrid class method), 652
deserialize() (deepmd.tf.descriptor.se_a.DescrptSeA class method), 664
deserialize() (deepmd.tf.descriptor.se_atten.DescrptDPA1Compat class method), 687
deserialize() (deepmd.tf.descriptor.se_atten.DescrptSeAtten class method), 683
deserialize() (deepmd.tf.descriptor.se_r.DescrptSeR class method), 693
deserialize() (deepmd.tf.fit.dipole.DipoleFittingSeA class method), 762
deserialize() (deepmd.tf.fit.DipoleFittingSeA class method), 780
deserialize() (deepmd.tf.fit.dos.DOSFitting class method), 766
deserialize() (deepmd.tf.fit.DOSFitting class method), 783
deserialize() (deepmd.tf.fit.ener.EnerFitting class method), 770
deserialize() (deepmd.tf.fit.EnerFitting class method), 787
deserialize() (deepmd.tf.fit.Fitting class method), 788
deserialize() (deepmd.tf.fit.fitting.Fitting class method), 771
deserialize() (deepmd.tf.fit.polar.PolarFittingSeA class method), 775
deserialize() (deepmd.tf.fit.PolarFittingSeA class method), 793
deserialize() (deepmd.tf.model.frozen.FrozenModel class method), 858
deserialize() (deepmd.tf.model.FrozenModel class method), 883
deserialize() (deepmd.tf.model.model.Model class method), 866
deserialize() (deepmd.tf.model.model.StandardModel class method), 867
deserialize() (deepmd.tf.utils.pair_tab.PairTab class method), 966
deserialize() (deepmd.tf.utils.PairTab class method), 992
deserialize() (deepmd.tf.utils.type_embed.TypeEmbedNet class method), 983
deserialize() (deepmd.utils.pair_tab.PairTab class method), 1044
deserialize() (deepmd.utils.spin.Spin class

method), 1055
deserialize_attention_layers() (deepmd.tf.descriptor.DescrptSeAtten class method), 735
deserialize_attention_layers() (deepmd.tf.descriptor.se_atten.DescrptSeAtten class method), 683
deserialize_hook (deepmd.backend.backend.Backend property), 254
deserialize_hook (deepmd.backend.dpmodel.DPModelBackend property), 256
deserialize_hook (deepmd.backend.pytorch.PyTorchBackend property), 258
deserialize_hook (deepmd.backend.tensorflow.TensorFlowBackend property), 259
deserialize_network() (deepmd.tf.descriptor.se.DescrptSe class method), 658
deserialize_network() (deepmd.tf.fit.Fitting class method), 789
deserialize_network() (deepmd.tf.fit.fitting.Fitting class method), 772
deserialize_network_strip() (deepmd.tf.descriptor.DescrptSeAtten class method), 736
deserialize_network_strip() (deepmd.tf.descriptor.se_atten.DescrptSeAtten class method), 683
deserialize_to_file() (in module deepmd.pt.utils.serialization), 633
deserialize_to_file() (in module deepmd.tf.utils.serialization), 977
detect_backend_by_model() (deepmd.backend.backend.Backend static method), 255
detect_model_version() (in module deepmd.tf.utils.convert), 941
device (in module deepmd.pt.model.atomic_model.base_atomic_model), 434
device (in module deepmd.pt.model.network.layernorm), 549
device (in module deepmd.pt.model.network.mlp), 550
device (in module deepmd.pt.model.task.dos), 583
device (in module deepmd.pt.model.task.ener), 585
device (in module deepmd.pt.model.task.fitting), 588
device (in module deepmd.pt.model.task.invar_fitting), 591
DEVICE (in module deepmd.pt.utils.env), 618
dict_to_device() (in module deepmd.pt.utils.utils), 637
dim_emb (deepmd.pt.model.descriptor.DescrptBlockRepformers property), 521
dim_emb (deepmd.pt.model.descriptor.DescrptBlockSeAtten property), 507
dim_emb (deepmd.pt.model.descriptor.DescrptDPA1 property), 512
dim_emb (deepmd.pt.model.descriptor.DescrptDPA2 property), 515
dim_emb (deepmd.pt.model.descriptor.DescrptGaussianLcc property), 518
dim_emb (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 property), 473
dim_emb (deepmd.pt.model.descriptor.dpa2.DescrptDPA2 property), 477
dim_emb (deepmd.pt.model.descriptor.gaussian_lcc.DescrptGaussianL property), 481
dim_emb (deepmd.pt.model.descriptor.repformers.DescrptBlockRepfo property), 491
dim_emb (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtten property), 497
dim_in (deepmd.pt.model.descriptor.DescrptBlockRepformers property), 521
dim_in (deepmd.pt.model.descriptor.DescrptBlockSeA property), 522
dim_in (deepmd.pt.model.descriptor.DescrptBlockSeAtten property), 507
dim_in (deepmd.pt.model.descriptor.DescrptGaussianLcc property), 518
dim_in (deepmd.pt.model.descriptor.gaussian_lcc.DescrptGaussianLc property), 481
dim_in (deepmd.pt.model.descriptor.repformers.DescrptBlockRepfor property), 490
dim_in (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA property), 494
dim_in (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtten property), 496
dim_in() (deepmd.dpmodel.utils.NativeLayer method), 369
dim_in() (deepmd.dpmodel.utils.network.NativeLayer method), 357
dim_in() (deepmd.pt.model.network.mlp.MLPLayer method), 551
dim_out (deepmd.dpmodel.descriptor.DescrptDPA1 property), 301
dim_out (deepmd.dpmodel.descriptor.DescrptSeA property), 306
dim_out (deepmd.dpmodel.descriptor.DescrptSeR property), 308
dim_out (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 property), 285
dim_out (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA property), 292
dim_out (deepmd.dpmodel.descriptor.se_r.DescrptSeR property), 295
dim_out (deepmd.pt.model.descriptor.DescrptBlockRepformers

property), 521
dim_out (deepmd.pt.model.descriptor.DescrptBlockSeA property), 522
dim_out (deepmd.pt.model.descriptor.DescrptBlockSeAtten property), 507
dim_out (deepmd.pt.model.descriptor.DescrptDPA1 property), 512
dim_out (deepmd.pt.model.descriptor.DescrptDPA2 property), 515
dim_out (deepmd.pt.model.descriptor.DescrptGaussianLcc property), 518
dim_out (deepmd.pt.model.descriptor.DescrptSeA property), 524
dim_out (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 property), 473
dim_out (deepmd.pt.model.descriptor.dpa2.DescrptDPA2 property), 477
dim_out (deepmd.pt.model.descriptor.gaussian_lcc.DescrptGaussianLcc property), 480
dim_out (deepmd.pt.model.descriptor.repformers.DescrptBlockRepformers property), 490
dim_out (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA property), 494
dim_out (deepmd.pt.model.descriptor.se_a.DescrptSeA property), 492
dim_out (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtten property), 496
dim_out() (deepmd.dpmodel.utils.NativeLayer method), 369
dim_out() (deepmd.dpmodel.utils.network.LayerNorm method), 358
dim_out() (deepmd.dpmodel.utils.network.NativeLayer method), 357
dim_out() (deepmd.pt.model.network.layernorm.LayerNorm method), 549
dim_out() (deepmd.pt.model.network.mlp.MLPLayer method), 551
DipoleChargeModifier (class in deepmd.tf), 1011
DipoleChargeModifier (class in deepmd.tf.infer), 820
DipoleChargeModifier (class in deepmd.tf.infer.data_modifier), 794
DipoleFitting (class in deepmd.dpmodel.fitting), 327
DipoleFitting (class in deepmd.dpmodel.fitting.dipole_fitting), 310
DipoleFittingNet (class in deepmd.pt.model.task), 598
DipoleFittingNet (class in deepmd.pt.model.task.dipole), 580
DipoleFittingSeA (class in deepmd.tf.fit), 778
DipoleFittingSeA (class in deepmd.tf.fit.dipole), 760
DipoleModel (class in deepmd.pt.model.model.dipole_model), 529
DipoleModel (class in deepmd.tf.model), 896
DipoleModel (class in deepmd.tf.model.tensor), 879
disp_file:
    training/disp_file (Argument), 157
disp_freq:
    training/disp_freq (Argument), 157
disp_message() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 913
disp_training:
    training/disp_training (Argument), 157
display_if_exist() (deepmd.pt.loss.loss.TaskLoss static method), 425
display_if_exist() (deepmd.pt.loss.TaskLoss static method), 432
display_if_exist() (deepmd.tf.loss.loss.Loss static method), 843
distributed_indices() (in module deepmd.pt.optimizer.LKF), 606
do_derivative() (in module deepmd.dpmodel.output_def), 379
do_mask() (in module deepmd.dpmodel.output_def), 379
do_reduce() (in module deepmd.dpmodel.output_def), 379
doc_only_pt_supported (in module deepmd.utils.argcheck), 1016
doc_only_tf_supported (in module deepmd.utils.argcheck), 1016
doc_train_input() (in module deepmd.entrypoints.doc), 383
doc_train_input() (in module deepmd.tf.entrypoints), 756
doc_train_input() (in module deepmd.tf.entrypoints.doc), 747
DOSFitting (class in deepmd.tf.fit), 780
DOSFitting (class in deepmd.tf.fit.dos), 763
DOSFittingNet (class in deepmd.dpmodel.fitting), 329
DOSFittingNet (class in deepmd.dpmodel.fitting.dos_fitting), 313
DOSFittingNet (class in deepmd.pt.model.task.dos), 583
DOSLoss (class in deepmd.pt.loss), 428
DOSLoss (class in deepmd.pt.loss.dos), 420
DOSLoss (class in deepmd.tf.loss), 845
DOSLoss (class in deepmd.tf.loss.dos), 837
DOSModel (class in deepmd.pt.model.model.dos_model), 530
DOSModel (class in deepmd.tf.model), 888
DOSModel (class in deepmd.tf.model.dos), 850
dotmul_flt_nvnmd (C++ function), 1316

dotmul_flt_nvnmd() (in module deepmd.tf.env.op_module), 1101
DotmulFltNvnmd() (in module deepmd.tf.env.op_module), 1075
DP (class in deepmd.calculator), 1058
DP (class in deepmd.tf.calculator), 994
DP_C_API_VERSION (C macro), 1259
DP_CHECK_OK (C macro), 1259
DP_ConvertPbtxtToPb (C++ function), 1218
DP_DeepPot (C++ struct), 1188
DP_DeepPot (C++ type), 1260
DP_DeepPot::aparam_nall (C++ member), 1188
DP_DeepPot::daparam (C++ member), 1188
DP_DeepPot::dfparam (C++ member), 1188
DP_DeepPot::dp (C++ member), 1188
DP_DeepPot::DP_DeepPot (C++ function), 1188
DP_DeepPot::exception (C++ member), 1188
DP_DeepPotCheckOK (C++ function), 1219
DP_DeepPotCompute (C++ function), 1219
DP_DeepPotCompute2 (C++ function), 1220
DP_DeepPotComputef (C++ function), 1221
DP_DeepPotComputef2 (C++ function), 1222
DP_DeepPotComputeMixedType (C++ function), 1223
DP_DeepPotComputeMixedTypef (C++ function), 1223
DP_DeepPotComputeNList (C++ function), 1224
DP_DeepPotComputeNList2 (C++ function), 1225
DP_DeepPotComputeNListf (C++ function), 1226
DP_DeepPotComputeNListf2 (C++ function), 1227
DP_DeepPotGetCutoff (C++ function), 1228
DP_DeepPotGetDimAParam (C++ function), 1228
DP_DeepPotGetDimFParam (C++ function), 1228
DP_DeepPotGetNumbTypes (C++ function), 1229
DP_DeepPotGetNumbTypesSpin (C++ function), 1229
DP_DeepPotGetTypeMap (C++ function), 1229
DP_DeepPotIsAParamNAll (C++ function), 1230
DP_DeepPotModelDevi (C++ struct), 1189
DP_DeepPotModelDevi (C++ type), 1260
DP_DeepPotModelDevi::aparam_nall (C++ member), 1189
DP_DeepPotModelDevi::daparam (C++ member), 1189
DP_DeepPotModelDevi::dfparam (C++ member), 1189
DP_DeepPotModelDevi::dp (C++ member), 1189
DP_DeepPotModelDevi::DP_DeepPotModelDevi (C++ function), 1189
DP_DeepPotModelDevi::exception (C++ member), 1189
DP_DeepPotModelDeviCheckOK (C++ function), 1230
DP_DeepPotModelDeviCompute (C++ function), 1230
DP_DeepPotModelDeviCompute2 (C++ function), 1231
DP_DeepPotModelDeviComputef (C++ function), 1232
DP_DeepPotModelDeviComputef2 (C++ function), 1233
DP_DeepPotModelDeviComputeNList (C++ function), 1234
DP_DeepPotModelDeviComputeNList2 (C++ function), 1235
DP_DeepPotModelDeviComputeNListf (C++ function), 1236
DP_DeepPotModelDeviComputeNListf2 (C++ function), 1237
DP_DeepPotModelDeviGetCutoff (C++ function), 1238
DP_DeepPotModelDeviGetDimAParam (C++ function), 1238
DP_DeepPotModelDeviGetDimFParam (C++ function), 1238
DP_DeepPotModelDeviGetNumbTypes (C++ function), 1239
DP_DeepPotModelDeviGetNumbTypesSpin (C++ function), 1239
DP_DeepPotModelDeviIsAParamNAll (C++ function), 1239
DP_DeepTensor (C++ struct), 1189
DP_DeepTensor (C++ type), 1260
DP_DeepTensor::DP_DeepTensor (C++ function), 1189
DP_DeepTensor::dt (C++ member), 1190
DP_DeepTensor::exception (C++ member), 1190
DP_DeepTensorCheckOK (C++ function), 1240
DP_DeepTensorCompute (C++ function), 1240
DP_DeepTensorComputef (C++ function), 1241
DP_DeepTensorComputeNList (C++ function), 1241
DP_DeepTensorComputeNListf (C++ function), 1242
DP_DeepTensorComputeTensor (C++ function), 1243
DP_DeepTensorComputeTensorf (C++ function), 1243
DP_DeepTensorComputeTensorNList (C++ function), 1244
DP_DeepTensorComputeTensorNListf (C++ function), 1244
DP_DeepTensorGetCutoff (C++ function), 1245
DP_DeepTensorGetNumbSelTypes (C++ function), 1245
DP_DeepTensorGetNumbTypes (C++ function), 1246
DP_DeepTensorGetOutputDim (C++ function), 1246
DP_DeepTensorGetSelTypes (C++ function), 1246

DP_DeepTensorGetTypeMap (C++ function), 1247
DP_DeleteChar (C++ function), 1247
DP_DeleteDeepPot (C++ function), 1247
DP_DeleteDeepPotModelDevi (C++ function), 1247
DP_DeleteDeepTensor (C++ function), 1248
DP_DeleteDipoleChargeModifier (C++ function), 1248
DP_DeleteNlist (C++ function), 1248
DP_DipoleChargeModifier (C++ struct), 1190
DP_DipoleChargeModifier (C++ type), 1260
DP_DipoleChargeModifier::dcm (C++ member), 1190
DP_DipoleChargeModifier::DP_DipoleChargeModifier (C++ function), 1190
DP_DipoleChargeModifier::exception (C++ member), 1190
DP_DipoleChargeModifierCheckOK (C++ function), 1249
DP_DipoleChargeModifierComputeNList (C++ function), 1249
DP_DipoleChargeModifierComputeNListf (C++ function), 1250
DP_DipoleChargeModifierGetCutoff (C++ function), 1251
DP_DipoleChargeModifierGetNumbSelTypes (C++ function), 1251
DP_DipoleChargeModifierGetNumbTypes (C++ function), 1251
DP_DipoleChargeModifierGetSelTypes (C++ function), 1252
dp_ipi() (in module deepmd.tf.entrypoints.ipi), 749
DP_NEW_OK (C macro), 1259
DP_NewDeepPot (C++ function), 1252
DP_NewDeepPotModelDevi (C++ function), 1252
DP_NewDeepPotModelDeviWithParam (C++ function), 1253
DP_NewDeepPotWithParam (C++ function), 1253
DP_NewDeepPotWithParam2 (C++ function), 1254
DP_NewDeepTensor (C++ function), 1254
DP_NewDeepTensorWithParam (C++ function), 1254
DP_NewDipoleChargeModifier (C++ function), 1255
DP_NewDipoleChargeModifierWithParam (C++ function), 1255
DP_NewNlist (C++ function), 1256
DP_NewNlist_comm (C++ function), 1256
DP_Nlist (C++ struct), 1190
DP_Nlist (C++ type), 1261
DP_Nlist::DP_Nlist (C++ function), 1191
DP_Nlist::exception (C++ member), 1191
DP_Nlist::nl (C++ member), 1191
DP_NlistCheckOK (C++ function), 1256
DP_PrintSummary (C++ function), 1257
DP_ReadFileToChar (C++ function), 1257
DP_ReadFileToChar2 (C++ function), 1257
DP_REQUIRES_OK (C macro), 1259
DP_SelectByType (C++ function), 1258
DP_SelectMapInt (C++ function), 1258
DPAssert (C++ function), 1316
DPAtomicModel (class in deepmd.dpmodel.atomic_model), 273
DPAtomicModel (class in deepmd.dpmodel.atomic_model.dp_atomic_model), 263
DPAtomicModel (class in deepmd.pt.model.atomic_model), 455
DPAtomicModel (class in deepmd.pt.model.atomic_model.dp_atomic_model), 440
DPCalculator (class in deepmd.pt.utils.ase_calc), 611
DPDipoleAtomicModel (class in deepmd.pt.model.atomic_model), 454
DPDipoleAtomicModel (class in deepmd.pt.model.atomic_model.dipole_atomic_model), 438
DPDOSAtomicModel (class in deepmd.pt.model.atomic_model), 455
DPDOSAtomicModel (class in deepmd.pt.model.atomic_model.dos_atomic_model), 439
DPDOSModel_ (in module deepmd.pt.model.model.dipole_model), 529
DPDOSModel_ (in module deepmd.pt.model.model.dos_model), 530
DPDOSModel_ (in module deepmd.pt.model.model.polar_model), 537
DPDriver (class in deepmd.driver), 1063
DPEnergyAtomicModel (class in deepmd.pt.model.atomic_model), 457
DPEnergyAtomicModel (class in deepmd.pt.model.atomic_model.energy_atomic_model), 442
DPEnergyModel_ (in module deepmd.dpmodel.model.ener_model), 343
DPEnergyModel_ (in module deepmd.pt.model.model.ener_model), 532
DPErrcheck (C macro), 1320
DPH5Path (class in deepmd.tf.utils.path), 969
DPH5Path (class in deepmd.utils.path), 1048
DpLoaderSet (class in deepmd.pt.utils.dataloader), 615
DPModelBackend (class in deepmd.backend.dpmodel), 256
DPModelCommon (class in deepmd.dpmodel), 380

DPModelCommon (class in deepmd.dpmodel.model), 348
DPModelCommon (class in deepmd.dpmodel.model.dp_model), 342
DPModelCommon (class in deepmd.pt.model.model), 541
DPModelCommon (class in deepmd.pt.model.model.dp_model), 530
DPOSPath (class in deepmd.tf.utils.path), 971
DPOSPath (class in deepmd.utils.path), 1047
DPPath (class in deepmd.tf.utils.path), 972
DPPath (class in deepmd.utils.path), 1045
DPPolarAtomicModel (class in deepmd.pt.model.atomic_model), 464
DPPolarAtomicModel (class in deepmd.pt.model.atomic_model.polar_atomic_model), 450
dprc_pairwise_idx() (in module deepmd.tf.env.op_module), 1101
DprcPairwiseIdx() (in module deepmd.tf.env.op_module), 1075
DPTabulate (class in deepmd.tf.utils.tabulate), 979
DPTrainer (class in deepmd.tf.train.trainer), 934
DPZBLLinearEnergyAtomicModel (class in deepmd.dpmodel.atomic_model), 274
DPZBLLinearEnergyAtomicModel (class in deepmd.dpmodel.atomic_model.linear_atomic_model), 266
DPZBLLinearEnergyAtomicModel (class in deepmd.pt.model.atomic_model), 457
DPZBLLinearEnergyAtomicModel (class in deepmd.pt.model.atomic_model.linear_atomic_model), 445
DPZBLModel (class in deepmd.pt.model.model), 542
DPZBLModel (class in deepmd.pt.model.model.dp_zbl_model), 531
DPZBLModel_ (in module deepmd.pt.model.model.dp_zbl_model), 531
Dropout (class in deepmd.pt.model.network.network), 553
DropPath (class in deepmd.pt.model.network.network), 555
dtype (in module deepmd.pt.model.atomic_model.base_atomic_model), 434
dtype (in module deepmd.pt.model.task.dos), 583
dtype (in module deepmd.pt.model.task.ener), 585
dtype (in module deepmd.pt.model.task.fitting), 588
dtype (in module deepmd.pt.model.task.invar_fitting), 591

E

electronic_configuration_embedding (in module deepmd.utils.econf_embd), 1035
embed_atom_type() (in module deepmd.tf.utils.type_embed), 982
Embedding (class in deepmd.pt.model.network.network), 559
embedding_net() (in module deepmd.tf.utils.network), 963
EMBEDDING_NET_PATTERN (in module deepmd.tf.env), 1002
embedding_net_rand_seed_shift() (in module deepmd.tf.utils.network), 963
EmbeddingNet (in module deepmd.dpmodel.utils), 368
EmbeddingNet (in module deepmd.dpmodel.utils.network), 358
EmbeddingNet (in module deepmd.pt.model.network.mlp), 552
empty_t() (in module deepmd.pt.model.network.mlp), 551
enable:
    nvnmd/enable (Argument), 160
enable_atom_ener_coeff:
    loss[ener_spin]/enable_atom_ener_coeff (Argument), 152
    loss[ener]/enable_atom_ener_coeff (Argument), 150
enable_compression() (deepmd.tf.descriptor.Descriptor method), 702
enable_compression() (deepmd.tf.descriptor.descriptor.Descriptor method), 644
enable_compression() (deepmd.tf.descriptor.DescrptHybrid method), 708
enable_compression() (deepmd.tf.descriptor.DescrptSeA method), 716
enable_compression() (deepmd.tf.descriptor.DescrptSeAtten method), 732
enable_compression() (deepmd.tf.descriptor.DescrptSeR method), 740
enable_compression() (deepmd.tf.descriptor.DescrptSeT method), 744
enable_compression() (deepmd.tf.descriptor.hybrid.DescrptHybrid method), 651
enable_compression() (deepmd.tf.descriptor.se_a.DescrptSeA method), 662
enable_compression()

(deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 679
enable_compression() (deepmd.tf.descriptor.se_r.DescrptSeR method), 692
enable_compression() (deepmd.tf.descriptor.se_t.DescrptSeT method), 696
enable_compression() (deepmd.tf.model.frozen.FrozenModel method), 857
enable_compression() (deepmd.tf.model.FrozenModel method), 882
enable_compression() (deepmd.tf.model.linear.LinearModel method), 859
enable_compression() (deepmd.tf.model.model.Model method), 864
enable_compression() (deepmd.tf.model.model.StandardModel method), 867
enable_compression() (deepmd.tf.model.pairtab.PairTabModel method), 873
enable_compression() (deepmd.tf.model.PairTabModel method), 886
enable_compression() (deepmd.tf.model.pairwise_dprc.PairwiseDPRc method), 876
enable_compression() (deepmd.tf.model.PairwiseDPRc method), 888
ENABLE_CUSTOMIZED_OP (in module deepmd.pt), 639
ENABLE_CUSTOMIZED_OP (in module deepmd.pt.cxx_op), 639
enable_mixed_precision() (deepmd.tf.descriptor.Descriptor method), 702
enable_mixed_precision() (deepmd.tf.descriptor.descriptor.Descriptor method), 644
enable_mixed_precision() (deepmd.tf.descriptor.DescrptHybrid method), 709
enable_mixed_precision() (deepmd.tf.descriptor.DescrptSeA method), 717
enable_mixed_precision() (deepmd.tf.descriptor.hybrid.DescrptHybrid method), 651
enable_mixed_precision() (deepmd.tf.descriptor.se_a.DescrptSeA method), 662
enable_mixed_precision() (deepmd.tf.fit.dipole.DipoleFittingSeA method), 762
enable_mixed_precision() (deepmd.tf.fit.DipoleFittingSeA method), 779
enable_mixed_precision() (deepmd.tf.fit.dos.DOSFitting method), 765
enable_mixed_precision() (deepmd.tf.fit.DOSFitting method), 783
enable_mixed_precision() (deepmd.tf.fit.ener.EnerFitting method), 770
enable_mixed_precision() (deepmd.tf.fit.EnerFitting method), 786
enable_mixed_precision() (deepmd.tf.fit.GlobalPolarFittingSeA method), 791
enable_mixed_precision() (deepmd.tf.fit.polar.GlobalPolarFittingSeA method), 777
enable_mixed_precision() (deepmd.tf.fit.polar.PolarFittingSeA method), 775
enable_mixed_precision() (deepmd.tf.fit.PolarFittingSeA method), 793
enable_mixed_precision() (deepmd.tf.model.model.Model method), 864
enable_mixed_precision() (deepmd.tf.model.model.StandardModel method), 866
enable_mixed_precision() (deepmd.tf.model.multi.MultiModel method), 871
enable_mixed_precision() (deepmd.tf.model.MultiModel method), 895
enable_profiler: training/enable_profiler (Argument), 158
Encode (class in deepmd.tf.nvnmd.utils), 920
Encode (class in deepmd.tf.nvnmd.utils.encode), 913
EnerDipoleLoss (class in deepmd.tf.loss), 846
EnerDipoleLoss (class in deepmd.tf.loss.ener), 841
EnerFitting (class in deepmd.tf.fit), 783
EnerFitting (class in deepmd.tf.fit.ener), 767
ENERGY_BIAS_TRAINABLE (in module deepmd.pt.utils.env), 618
EnergyFittingNet (class in deepmd.dpmodel.fitting), 331

EnergyFittingNet (class in deepmd.dpmodel.fitting.ener_fitting), 315
EnergyFittingNet (class in deepmd.pt.model.task), 600
EnergyFittingNet (class in deepmd.pt.model.task.ener), 585
EnergyFittingNetDirect (class in deepmd.pt.model.task), 601
EnergyFittingNetDirect (class in deepmd.pt.model.task.ener), 586
EnergyHead (class in deepmd.pt.model.network.network), 570
EnergyModel (class in deepmd.dpmodel.model.ener_model), 343
EnergyModel (class in deepmd.pt.model.model), 543
EnergyModel (class in deepmd.pt.model.model.ener_model), 532
EnergySpinLoss (class in deepmd.pt.loss), 430
EnergySpinLoss (class in deepmd.pt.loss.ener_spin), 423
EnergyStdLoss (class in deepmd.pt.loss), 429
EnergyStdLoss (class in deepmd.pt.loss.ener), 422
EnerModel (class in deepmd.tf.model), 890
EnerModel (class in deepmd.tf.model.ener), 853
EnerSpinLoss (class in deepmd.tf.loss), 846
EnerSpinLoss (class in deepmd.tf.loss.ener), 840
EnerStdLoss (class in deepmd.tf.loss), 848
EnerStdLoss (class in deepmd.tf.loss.ener), 838
ENTRY_POINT (deepmd.backend.backend.Backend.Feature attribute), 254
entry_point_hook (deepmd.backend.backend.Backend property), 254
entry_point_hook (deepmd.backend.dpmodel.DPModelBackend property), 256
entry_point_hook (deepmd.backend.pytorch.PyTorchBackend property), 257
entry_point_hook (deepmd.backend.tensorflow.TensorFlowBackend property), 259
env_mat_a (C++ function), 1317
env_mat_r (C++ function), 1317
env_protection:
    model[standard]/descriptor[se_a_ebd_v2]/env_protection (Argument), 138
    model[standard]/descriptor[se_a_tpe]/env_protection (Argument), 122
    model[standard]/descriptor[se_e2_a]/env_protection (Argument), 118
EnvMat (class in deepmd.dpmodel.utils), 367
EnvMat (class in deepmd.dpmodel.utils.env_mat), 352
EnvMatStat (class in deepmd.pt.utils.env_mat_stat), 618
EnvMatStat (class in deepmd.utils.env_mat_stat), 1037
EnvMatStatSe (class in deepmd.pt.utils.env_mat_stat), 619
eps (in module deepmd.backend), 260
eval() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 340
eval() (deepmd.infer.deep_dos.DeepDOS method), 388
eval() (deepmd.infer.deep_eval.DeepEvalBackend method), 390
eval() (deepmd.infer.deep_polar.DeepGlobalPolar method), 396
eval() (deepmd.infer.deep_pot.DeepPot method), 397
eval() (deepmd.infer.deep_tensor.DeepTensor method), 399
eval() (deepmd.infer.DeepPot method), 409
eval() (deepmd.pt.infer.deep_eval.DeepEval method), 417
eval() (deepmd.tf.DeepEval method), 1009
eval() (deepmd.tf.DipoleChargeModifier method), 1012
eval() (deepmd.tf.infer.data_modifier.DipoleChargeModifier method), 795
eval() (deepmd.tf.infer.deep_dos.DeepDOS method), 797
eval() (deepmd.tf.infer.deep_eval.DeepEval method), 802
eval() (deepmd.tf.infer.deep_polar.DeepGlobalPolar method), 808
eval() (deepmd.tf.infer.deep_pot.DeepPot method), 810
eval() (deepmd.tf.infer.deep_tensor.DeepTensor method), 812
eval() (deepmd.tf.infer.DeepDOS method), 823
eval() (deepmd.tf.infer.DeepEval method), 828
eval() (deepmd.tf.infer.DeepGlobalPolar method), 830
eval() (deepmd.tf.infer.DeepPot method), 832
eval() (deepmd.tf.infer.DipoleChargeModifier method), 821
eval() (deepmd.tf.infer.ewald_recp.EwaldRecp method), 815
eval() (deepmd.tf.infer.EwaldRecp method), 833
eval() (deepmd.tf.loss.dos.DOSLoss method), 838
eval() (deepmd.tf.loss.DOSLoss method), 845
eval() (deepmd.tf.loss.ener.EnerDipoleLoss method), 842
eval() (deepmd.tf.loss.ener.EnerSpinLoss method), 841
eval() (deepmd.tf.loss.ener.EnerStdLoss method), 840
eval() (deepmd.tf.loss.EnerDipoleLoss method), 846

eval() (deepmd.tf.loss.EnerSpinLoss method), 847
eval() (deepmd.tf.loss.EnerStdLoss method), 849
eval() (deepmd.tf.loss.loss.Loss method), 843
eval() (deepmd.tf.loss.tensor.TensorLoss method), 844
eval() (deepmd.tf.loss.TensorLoss method), 850
eval_descriptor() (deepmd.infer.deep_eval.DeepEval method), 393
eval_descriptor() (deepmd.infer.deep_eval.DeepEvalBackend method), 391
eval_descriptor() (deepmd.infer.DeepEval method), 407
eval_descriptor() (deepmd.tf.DeepEval method), 1010
eval_descriptor() (deepmd.tf.infer.deep_eval.DeepEval method), 803
eval_descriptor() (deepmd.tf.infer.DeepEval method), 829
eval_full() (deepmd.infer.deep_tensor.DeepTensor method), 400
eval_full() (deepmd.tf.infer.deep_tensor.DeepTensor method), 813
eval_model() (in module deepmd.pt.infer.deep_eval), 419
eval_single_list() (deepmd.tf.train.trainer.DPTrainer static method), 935
eval_typeebd() (deepmd.infer.deep_eval.DeepEval method), 394
eval_typeebd() (deepmd.infer.deep_eval.DeepEvalBackend method), 392
eval_typeebd() (deepmd.infer.DeepEval method), 407
eval_typeebd() (deepmd.tf.DeepEval method), 1008
eval_typeebd() (deepmd.tf.infer.deep_eval.DeepEval method), 800
eval_typeebd() (deepmd.tf.infer.deep_eval.DeepEvalOld method), 806
eval_typeebd() (deepmd.tf.infer.DeepEval method), 826
Evoformer2bBackBone (class in deepmd.pt.model.backbone.evoformer2b), 465
Evoformer2bEncoder (class in deepmd.pt.model.network.network), 574
Evoformer3bEncoder (class in deepmd.pt.model.network.network), 577
Evoformer3bEncoderLayer (class in deepmd.pt.model.network.network), 576
EvoformerEncoderLayer (class in deepmd.pt.model.network.network), 574
ewald_beta:
    model/modifier[dipole_charge]/ewald_beta (Argument), 114
ewald_h:
    model/modifier[dipole_charge]/ewald_h (Argument), 114
ewald_recp() (in module deepmd.tf.env.op_module), 1101
EwaldRecp (class in deepmd.tf.infer), 833
EwaldRecp (class in deepmd.tf.infer.ewald_recp), 815
EwaldRecp() (in module deepmd.tf.env.op_module), 1076
exclude_types (deepmd.pt.model.task.dipole.DipoleFittingNet attribute), 581
exclude_types (deepmd.pt.model.task.DipoleFittingNet attribute), 599
exclude_types (deepmd.pt.model.task.dos.DOSFittingNet attribute), 584
exclude_types (deepmd.pt.model.task.ener.EnergyFittingNet attribute), 586
exclude_types (deepmd.pt.model.task.EnergyFittingNet attribute), 601
exclude_types (deepmd.pt.model.task.fitting.GeneralFitting attribute), 590
exclude_types (deepmd.pt.model.task.invar_fitting.InvarFitting attribute), 592
exclude_types (deepmd.pt.model.task.PolarFittingNet attribute), 604
exclude_types (deepmd.pt.model.task.polarizability.PolarFittingNet attribute), 595
exclude_types:
    model[standard]/descriptor[se_a_ebd_v2]/exclude_types (Argument), 137
    model[standard]/descriptor[se_a_mask]/exclude_types (Argument), 139
    model[standard]/descriptor[se_a_tpe]/exclude_types (Argument), 122
    model[standard]/descriptor[se_atten_v2]/exclude_types (Argument), 131
    model[standard]/descriptor[se_atten]/exclude_types (Argument), 126
    model[standard]/descriptor[se_e2_a]/exclude_types (Argument), 118
    model[standard]/descriptor[se_e2_r]/exclude_types (Argument), 124
execute() (deepmd.utils.batch_size.AutoBatchSize method), 1020
execute_all() (deepmd.pt.utils.auto_batch_size.AutoBatchSize method), 613
execute_all() (deepmd.utils.batch_size.AutoBatchSize method), 1021
exits() (deepmd.tf.nvnmd.utils.fio.Fio method), 915
expand_aparam() (deepmd.pt.model.model.spin_model.SpinModel static method), 538
expand_aparam() (deepmd.pt.model.model.SpinModel static method), 547

expand_sys_str() (in module deepmd.common), 1062
expand_sys_str() (in module deepmd.tf.common), 997
explicit_ntypes (deepmd.tf.descriptor.Descriptor property), 699
explicit_ntypes (deepmd.tf.descriptor.descriptor.Descriptor property), 641
explicit_ntypes (deepmd.tf.descriptor.DescrptHybrid property), 705
explicit_ntypes (deepmd.tf.descriptor.DescrptSeA property), 715
explicit_ntypes (deepmd.tf.descriptor.DescrptSeAtten property), 731
explicit_ntypes (deepmd.tf.descriptor.hybrid.DescrptHybrid property), 648
explicit_ntypes (deepmd.tf.descriptor.se_a.DescrptSeA property), 661
explicit_ntypes (deepmd.tf.descriptor.se_atten.DescrptSeAtten property), 678
extend_bin() (deepmd.tf.nvnmd.utils.Encode method), 921
extend_bin() (deepmd.tf.nvnmd.utils.encode.Encode method), 914
extend_coord_with_ghosts() (in module deepmd.dpmodel.utils), 371
extend_coord_with_ghosts() (in module deepmd.dpmodel.utils.nlist), 361
extend_coord_with_ghosts() (in module deepmd.pt.utils.nlist), 627
extend_coord_with_ghosts() (in module deepmd.tf.utils.nlist), 965
extend_hex() (deepmd.tf.nvnmd.utils.Encode method), 921
extend_hex() (deepmd.tf.nvnmd.utils.encode.Encode method), 914
extend_input_and_build_neighbor_list() (in module deepmd.pt.utils.nlist), 626
extend_list() (deepmd.tf.nvnmd.utils.Encode method), 921
extend_list() (deepmd.tf.nvnmd.utils.encode.Encode method), 914
extend_nlist() (deepmd.dpmodel.model.spin_model.SpinModel static method), 345
extend_nlist() (deepmd.dpmodel.model.SpinModel static method), 349
extend_nlist() (deepmd.pt.model.model.spin_model.SpinModel static method), 538
extend_nlist() (deepmd.pt.model.model.SpinModel static method), 547
extra_repr() (deepmd.pt.model.network.network.DropPath method), 555

F

features (deepmd.backend.backend.Backend attribute), 255
features (deepmd.backend.dpmodel.DPModelBackend attribute), 256
features (deepmd.backend.pytorch.PyTorchBackend attribute), 258
features (deepmd.backend.tensorflow.TensorFlowBackend attribute), 260
ffn (deepmd.pt.model.network.network.NeighborWiseAttentionLaye attribute), 568
filter_GR2D() (in module deepmd.tf.nvnmd.descriptor.se_a), 899
filter_GR2D() (in module deepmd.tf.nvnmd.descriptor.se_atten), 900
filter_lower_R42GR() (in module deepmd.tf.nvnmd.descriptor.se_a), 899
filter_lower_R42GR() (in module deepmd.tf.nvnmd.descriptor.se_atten), 900
filter_tensorVariableList() (in module deepmd.tf.nvnmd.entrypoints.freeze), 901
find_libpython (in module deepmd.tf.lmp), 1004
find_max_expo (C++ function), 1317
find_max_expo() (deepmd.tf.nvnmd.utils.Encode method), 921
find_max_expo() (deepmd.tf.nvnmd.utils.encode.Encode method), 913
Fio (class in deepmd.tf.nvnmd.utils.fio), 915
FioBin (class in deepmd.tf.nvnmd.utils), 922
FioBin (class in deepmd.tf.nvnmd.utils.fio), 916
FioDic (class in deepmd.tf.nvnmd.utils), 922
FioDic (class in deepmd.tf.nvnmd.utils.fio), 915
FioJsonDic (class in deepmd.tf.nvnmd.utils.fio), 916
FioNpyDic (class in deepmd.tf.nvnmd.utils.fio), 916
FioTxt (class in deepmd.tf.nvnmd.utils), 922
FioTxt (class in deepmd.tf.nvnmd.utils.fio), 916
fit_diag:
    model[standard]/fitting_net[polar]/fit_diag (Argument), 144
fit_output_to_model_output() (in module deepmd.dpmodel.model.transform_output), 347
fit_output_to_model_output() (in module deepmd.pt.model.model.transform_output), 540
Fitting (class in deepmd.pt.model.task), 602
Fitting (class in deepmd.pt.model.task.fitting), 588
Fitting (class in deepmd.tf.fit), 787
Fitting (class in deepmd.tf.fit.fitting), 771
fitting_args_plugin (in module deepmd.utils.argcheck), 1017

fitting_check_output() (in module deepmd.dpmodel), 382
fitting_check_output() (in module deepmd.dpmodel.output_def), 375
fitting_dipole() (in module deepmd.utils.argcheck), 1018
fitting_dos() (in module deepmd.utils.argcheck), 1017
fitting_ener() (in module deepmd.utils.argcheck), 1017
fitting_net: model[standard]/fitting_net (Argument), 139
fitting_net_dict: model[multi]/fitting_net_dict (Argument), 145
FITTING_NET_PATTERN (in module deepmd.tf.env), 1002
fitting_output_def() (deepmd.dpmodel.atomic_model.dp_atomic_model.DPAtomicModel method), 263
fitting_output_def() (deepmd.dpmodel.atomic_model.DPAtomicModel method), 273
fitting_output_def() (deepmd.dpmodel.atomic_model.linear_atomic_model.LinearEnergyAtomicModel method), 266
fitting_output_def() (deepmd.dpmodel.atomic_model.LinearEnergyAtomicModel method), 277
fitting_output_def() (deepmd.dpmodel.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 269
fitting_output_def() (deepmd.dpmodel.atomic_model.PairTabAtomicModel method), 278
fitting_output_def() (deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel method), 440
fitting_output_def() (deepmd.pt.model.atomic_model.DPAtomicModel method), 455
fitting_output_def() (deepmd.pt.model.atomic_model.linear_atomic_model.LinearEnergyAtomicModel method), 444
fitting_output_def() (deepmd.pt.model.atomic_model.LinearEnergyAtomicModel method), 459
fitting_output_def() (deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 447
fitting_output_def() (deepmd.pt.model.atomic_model.PairTabAtomicModel method), 461
fitting_output_def() (deepmd.pt.model.model.frozen.FrozenModel method), 533
fitting_output_def() (deepmd.pt.model.model.FrozenModel method), 543
fitting_polar() (in module deepmd.utils.argcheck), 1017
fitting_variant_type_args() (in module deepmd.utils.argcheck), 1018
fitting_weight: training/fitting_weight (Argument), 158
FittingNet (in module deepmd.dpmodel.utils), 368
FittingNet (in module deepmd.dpmodel.utils.network), 358
FittingNet (in module deepmd.pt.model.network.mlp), 552
FittingNetAttenLcc (class in deepmd.pt.model.task), 597
FittingNetAttenLcc (class in deepmd.pt.model.task.atten_lcc), 578
FittingOutputDef (class in deepmd.dpmodel), 380
FittingOutputDef (class in deepmd.dpmodel.output_def), 377
flatten_list_vec() (in module deepmd.utils.argcheck), 1035
flt2bin() (deepmd.tf.nvnmd.utils.Encode method), 921
flt2bin() (deepmd.tf.nvnmd.utils.encode.Encode method), 914
flt2bin_one() (deepmd.tf.nvnmd.utils.Encode method), 921
flt2bin_one() (deepmd.tf.nvnmd.utils.encode.Encode method), 913
FLT_MAX (C macro), 1320
flt_nvnmd() (in module deepmd.tf.env.op_module), 1102
FLTNVNMD() (in module deepmd.tf.env.op_module), 1076
format_model_suffix() (in module deepmd.backend.suffix), 258
format_nlist_i_cpu (C++ function), 1318
format_nlist_i_fill_a (C++ function), 1318
format_training_message() (in module deepmd.loggers.training), 413
format_training_message_per_task() (in module deepmd.loggers.training), 413
forward (deepmd.pt.model.network.mlp.MLP attribute), 552
forward() (deepmd.pt.loss.denoise.DenoiseLoss method), 420
forward() (deepmd.pt.loss.DenoiseLoss method), 428
forward() (deepmd.pt.loss.dos.DOSLoss method),

421
forward() (deepmd.pt.loss.DOSLoss method), 429
forward() (deepmd.pt.loss.ener.EnergyStdLoss method), 423
forward() (deepmd.pt.loss.ener_spin.EnergySpinLoss method), 424
forward() (deepmd.pt.loss.EnergySpinLoss method), 431
forward() (deepmd.pt.loss.EnergyStdLoss method), 430
forward() (deepmd.pt.loss.loss.TaskLoss method), 425
forward() (deepmd.pt.loss.TaskLoss method), 432
forward() (deepmd.pt.loss.tensor.TensorLoss method), 426
forward() (deepmd.pt.loss.TensorLoss method), 433
forward() (deepmd.pt.model.backbone.BackBone method), 467
forward() (deepmd.pt.model.backbone.backbone.BackBone method), 465
forward() (deepmd.pt.model.backbone.evoformer2b.Evoformer2bBackbone method), 466
forward() (deepmd.pt.model.descriptor.descriptor.Descriptor method), 469
forward() (deepmd.pt.model.descriptor.DescriptorBlock method), 506
forward() (deepmd.pt.model.descriptor.DescrptBlockRepformers method), 522
forward() (deepmd.pt.model.descriptor.DescrptBlockSeA method), 523
forward() (deepmd.pt.model.descriptor.DescrptBlockSeAtten method), 508
forward() (deepmd.pt.model.descriptor.DescrptDPA1 method), 513
forward() (deepmd.pt.model.descriptor.DescrptDPA2 method), 516
forward() (deepmd.pt.model.descriptor.DescrptGaussianLcc method), 518
forward() (deepmd.pt.model.descriptor.DescrptHybrid method), 519
forward() (deepmd.pt.model.descriptor.DescrptSeA method), 525
forward() (deepmd.pt.model.descriptor.DescrptSeR method), 527
forward() (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 method), 474
forward() (deepmd.pt.model.descriptor.dpa2.DescrptDPA2 method), 478
forward() (deepmd.pt.model.descriptor.gaussian_lcc.DescrptGaussianLcc method), 481
forward() (deepmd.pt.model.descriptor.hybrid.DescrptHybrid method), 482
forward() (deepmd.pt.model.descriptor.repformer_layer.Atten2 method), 486
forward() (deepmd.pt.model.descriptor.repformer_layer.Atten2Map method), 485
forward() (deepmd.pt.model.descriptor.repformer_layer.Atten2Mult method), 485
forward() (deepmd.pt.model.descriptor.repformer_layer.LocalAtten method), 487
forward() (deepmd.pt.model.descriptor.repformer_layer.RepformerL method), 488
forward() (deepmd.pt.model.descriptor.repformers.DescrptBlockRep method), 491
forward() (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA method), 495
forward() (deepmd.pt.model.descriptor.se_a.DescrptSeA method), 493
forward() (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtte method), 498
forward() (deepmd.pt.model.descriptor.se_atten.GatedAttentionLay method), 501
forward() (deepmd.pt.model.descriptor.se_atten.NeighborGatedAtte method), 499
forward() (deepmd.pt.model.descriptor.se_atten.NeighborGatedAtte method), 500
forward() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 504
forward() (deepmd.pt.model.model.dipole_model.DipoleModel method), 529
forward() (deepmd.pt.model.model.dos_model.DOSModel method), 530
forward() (deepmd.pt.model.model.dp_zbl_model.DPZBLModel method), 531
forward() (deepmd.pt.model.model.DPZBLModel method), 542
forward() (deepmd.pt.model.model.ener_model.EnergyModel method), 532
forward() (deepmd.pt.model.model.EnergyModel method), 543
forward() (deepmd.pt.model.model.frozen.FrozenModel method), 533
forward() (deepmd.pt.model.model.FrozenModel method), 544
forward() (deepmd.pt.model.model.polar_model.PolarModel method), 537
forward() (deepmd.pt.model.model.spin_model.SpinEnergyModel method), 539
forward() (deepmd.pt.model.model.SpinEnergyModel method), 546
forward() (deepmd.pt.model.network.layernorm.LayerNorm method), 549
forward() (deepmd.pt.model.network.mlp.MLPLayer method), 551
forward() (deepmd.pt.model.network.network.AtomAttention method), 573
forward() (deepmd.pt.model.network.network.Attention method), 572

forward() (deepmd.pt.model.network.network.DropoutNode method), 554
forward() (deepmd.pt.model.network.network.DropPath method), 555
forward() (deepmd.pt.model.network.network.EnergyHead method), 571
forward() (deepmd.pt.model.network.network.EvoformerEncoder method), 575
forward() (deepmd.pt.model.network.network.EvoformerEncoderLayer method), 577
forward() (deepmd.pt.model.network.network.EvoformerFFNBlock method), 577
forward() (deepmd.pt.model.network.network.GatedSelfAttention method), 574
forward() (deepmd.pt.model.network.network.GaussianEmbedding method), 569
forward() (deepmd.pt.model.network.network.GaussianKernel method), 566
forward() (deepmd.pt.model.network.network.Identity method), 565
forward() (deepmd.pt.model.network.network.LocalSelfMultiheadAttention method), 570
forward() (deepmd.pt.model.network.network.MaskLMHead method), 562
forward() (deepmd.pt.model.network.network.NeighborWiseAttention method), 567
forward() (deepmd.pt.model.network.network.NeighborWiseAttentionLayer method), 568
forward() (deepmd.pt.model.network.network.NodeTaskHead method), 570
forward() (deepmd.pt.model.network.network.NonLinear method), 562
forward() (deepmd.pt.model.network.network.NonLinearHead method), 561
forward() (deepmd.pt.model.network.network.OuterProduct method), 572
forward() (deepmd.pt.model.network.network.ResidualDeep method), 563
forward() (deepmd.pt.model.network.network.ResidualLinear method), 556
forward() (deepmd.pt.model.network.network.SimpleLinear method), 557
forward() (deepmd.pt.model.network.network.TransitionLayer method), 559
forward() (deepmd.pt.model.network.network.TriangleMultiplication method), 574
forward() (deepmd.pt.model.network.network.TypeEmbedNet method), 563
forward() (deepmd.pt.model.network.network.TypeEmbedNetConsistent method), 564
forward() (deepmd.pt.model.network.network.TypeFilter method), 556
forward() (deepmd.pt.model.task.atten_lcc.FittingNetAttenLcc method), 579
forward() (deepmd.pt.model.task.denoise.DenoiseNet method), 580
forward() (deepmd.pt.model.task.DenoiseNet method), 598
forward() (deepmd.pt.model.task.dipole.DipoleFittingNet method), 582
forward() (deepmd.pt.model.task.DipoleFittingNet method), 600
forward() (deepmd.pt.model.task.ener.EnergyFittingNetDirect method), 587
forward() (deepmd.pt.model.task.EnergyFittingNetDirect method), 602
forward() (deepmd.pt.model.task.FittingNetAttenLcc method), 597
forward() (deepmd.pt.model.task.invar_fitting.InvarFitting method), 593
forward() (deepmd.pt.model.task.PolarFittingNet method), 605
forward() (deepmd.pt.model.task.polarizability.PolarFittingNet method), 595
forward() (deepmd.pt.model.task.type_predict.TypePredictNet method), 596
forward() (deepmd.pt.model.task.TypePredictNet method), 605
forward() (deepmd.pt.train.wrapper.ModelWrapper method), 611
forward() (deepmd.pt.utils.AtomExcludeMask method), 638
forward() (deepmd.pt.utils.exclude_mask.AtomExcludeMask method), 620
forward() (deepmd.pt.utils.exclude_mask.PairExcludeMask method), 620
forward() (deepmd.pt.utils.neighbor_stat.NeighborStatOP method), 625
forward() (deepmd.pt.utils.PairExcludeMask method), 638
forward() (deepmd.pt.utils.utils.ActivationFn method), 637
forward_atomic() (deepmd.dpmodel.atomic_model.dp_atomic_mod method), 263
forward_atomic() (deepmd.dpmodel.atomic_model.DPAtomicModel method), 274
forward_atomic() (deepmd.dpmodel.atomic_model.linear_atomic_m method), 265
forward_atomic() (deepmd.dpmodel.atomic_model.LinearEnergyAt method), 276
forward_atomic() (deepmd.dpmodel.atomic_model.pairtab_atomic_ method), 269
forward_atomic() (deepmd.dpmodel.atomic_model.PairTabAtomicM method), 278
forward_atomic() (deepmd.pt.model.atomic_model.dp_atomic_mod method), 440

forward_atomic() (deepmd.pt.model.atomic_model.DPAtomicModel method), 456
forward_atomic() (deepmd.pt.model.atomic_model.linear_atomic_model.LinearEnergyAtomicModel method), 443
forward_atomic() (deepmd.pt.model.atomic_model.LinearEnergyAtomicModel method), 459
forward_atomic() (deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 448
forward_atomic() (deepmd.pt.model.atomic_model.PairTabAtomicModel method), 462
forward_common() (deepmd.pt.model.model.spin_model.SpinModel method), 539
forward_common() (deepmd.pt.model.model.SpinModel method), 548
forward_common_atomic() (deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicModel method), 261
forward_common_atomic() (deepmd.dpmodel.atomic_model.BaseAtomicModel method), 272
forward_common_atomic() (deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel method), 436
forward_common_atomic() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 452
forward_common_lower() (deepmd.pt.model.model.spin_model.SpinModel method), 539
forward_common_lower() (deepmd.pt.model.model.SpinModel method), 548
forward_lower() (deepmd.pt.model.model.dipole_model.DipoleModel method), 529
forward_lower() (deepmd.pt.model.model.dos_model.DOSModel method), 530
forward_lower() (deepmd.pt.model.model.dp_zbl_model.DPZBLModel method), 532
forward_lower() (deepmd.pt.model.model.DPZBLModel method), 542
forward_lower() (deepmd.pt.model.model.ener_model.EnergyModel method), 532
forward_lower() (deepmd.pt.model.model.EnergyModel method), 543
forward_lower() (deepmd.pt.model.model.polar_model.PolarModel method), 537
forward_lower() (deepmd.pt.model.model.spin_model.SpinEnergyModel method), 540
forward_lower() (deepmd.pt.model.model.SpinEnergyModel method), 546
freeze() (in module deepmd.pt.entrypoints.main), 415
freeze() (in module deepmd.tf.entrypoints), 756
freeze() (in module deepmd.tf.entrypoints.freeze), 768
frozen_model_args() (in module deepmd.pt.utils.preprocess), 1018
FrozenModel (class in deepmd.pt.model.model), 543
FrozenModel (class in deepmd.pt.model.model.frozen), 533
FrozenModel (class in deepmd.tf.model.frozen), 881
FrozenModel (class in deepmd.tf.model.frozen), 856

G

GatedAttentionLayer (class in deepmd.dpmodel.descriptor.dpa1), 287
GatedAttentionLayer (class in deepmd.pt.model.descriptor.se_atten), 501
GatedSelfAttention (class in deepmd.pt.model.network.network), 568
gather_placeholder() (in module deepmd.tf.model.pairwise_dprc), 877
gaussian() (in module deepmd.pt.model.network.network), 565
GaussianEmbedding (class in deepmd.pt.model.network.network), 565
GaussianKernel (class in deepmd.pt.model.network.network), 565
gelu (in module deepmd.tf.op._gelu), 925
gelu() (in module deepmd.tf.common), 999
Gelu() (in module deepmd.tf.env.op_module), 1076
gelu() (in module deepmd.tf.env.op_module), 1102
gelu_custom() (in module deepmd.tf.env.op_module), 1102
gelu_grad() (in module deepmd.tf.env.op_module), 1102
gelu_grad_custom() (in module deepmd.tf.env.op_module), 1102
gelu_grad_grad() (in module deepmd.tf.env.op_module), 1103
gelu_grad_grad_custom() (in module deepmd.tf.env.op_module), 1103
gelu_tf() (in module deepmd.tf.common), 999
GeluCustom() (in module deepmd.tf.env.op_module), 1076
GeluGPUExecuteFunctor (C++ struct), 1273
GeluGPUExecuteFunctor::operator() (C++ function), 1273
GeluGrad() (in module deepmd.tf.env.op_module), 1076
GeluGradCustom() (in module deepmd.tf.env.op_module), 1077
GeluGradGPUExecuteFunctor (C++ struct), 1274
GeluGradGPUExecuteFunctor::operator() (C++ function), 1274
GeluGradGrad() (in module deepmd.tf.env.op_module), 1077

GeluGradGradCustom() (in module deepmd.tf.env.op_module), 1077
GeluGradGradGPUExecuteFunctor (C++ struct), 1274
GeluGradGradGPUExecuteFunctor::operator() (C++ function), 1274
gen_args() (in module deepmd.tf.utils.argcheck), 937
gen_args() (in module deepmd.utils.argcheck), 1019
gen_doc() (in module deepmd.tf.utils.argcheck), 937
gen_doc() (in module deepmd.utils.argcheck), 1019
gen_json() (in module deepmd.tf.utils.argcheck), 937
gen_json() (in module deepmd.utils.argcheck), 1019
GeneralFitting (class in deepmd.dpmodel.fitting.general_fitting), 317
GeneralFitting (class in deepmd.pt.model.task.fitting), 589
generate() (deepmd.tf.utils.parallel_op.ParallelOp method), 968
get() (deepmd.tf.nvnmd.utils.fio.FioDic method), 916
get() (deepmd.tf.nvnmd.utils.FioDic method), 922
get() (deepmd.tf.utils.pair_tab.PairTab method), 967
get() (deepmd.tf.utils.PairTab method), 993
get() (deepmd.utils.pair_tab.PairTab method), 1044
get_activation_fn() (in module deepmd.dpmodel.utils.network), 357
get_activation_func() (in module deepmd.tf.common), 1000
get_all_argument() (deepmd.utils.argcheck.ArgsPlugin method), 1017
get_atom_axis() (in module deepmd.pt.model.model.transform_output), 540
get_atom_exclude_types() (deepmd.utils.spin.Spin method), 1055
get_atom_exclude_types_placeholder() (deepmd.utils.spin.Spin method), 1055
get_atom_type() (deepmd.tf.utils.data.DeepmdData method), 946
get_atom_type() (deepmd.tf.utils.DeepmdData method), 987
get_atom_type() (deepmd.utils.data.DeepmdData method), 1027
get_attention_layer_nodes_from_graph_def() (in module deepmd.tf.utils.graph), 957
get_attention_layer_variables_from_graph_def() (in module deepmd.tf.utils.graph), 957
get_avg() (deepmd.utils.env_mat_stat.EnvMatStat method), 1037
get_backend() (deepmd.backend.backend.Backend static method), 255
get_backend_info() (deepmd.pt.entrypoints.main.SummaryPrinter method), 415
get_backend_info() (deepmd.utils.summary.SummaryPrinter method), 1056
get_backends() (deepmd.backend.backend.Backend static method), 255
get_backends_by_feature() (deepmd.backend.backend.Backend static method), 255
get_batch() (deepmd.tf.utils.data.DeepmdData method), 945
get_batch() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 949
get_batch() (deepmd.tf.utils.DeepmdData method), 986
get_batch() (deepmd.tf.utils.DeepmdDataSystem method), 989
get_batch() (deepmd.utils.data.DeepmdData method), 1027
get_batch() (deepmd.utils.data_system.DeepmdDataSystem method), 1031
get_batch_mixed() (deepmd.tf.utils.data_system.DeepmdDataSyste method), 949
get_batch_mixed() (deepmd.tf.utils.DeepmdDataSystem method), 990
get_batch_mixed() (deepmd.utils.data_system.DeepmdDataSystem method), 1032
get_batch_size() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 950
get_batch_size() (deepmd.tf.utils.DeepmdDataSystem method), 991
get_batch_size() (deepmd.utils.data_system.DeepmdDataSystem method), 1033
get_batch_standard() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 949
get_batch_standard() (deepmd.tf.utils.DeepmdDataSystem method), 990
get_batch_standard() (deepmd.utils.data_system.DeepmdDataSystem method), 1032
get_class_name() (in module deepmd.pt.utils.multi_task), 624
get_compute_device() (deepmd.pt.entrypoints.main.SummaryPrinter method), 415
get_compute_device() (deepmd.utils.summary.SummaryPrinter method), 1056
get_constant_initializer() (in module deepmd.tf.nvnmd.utils.weight), 920
get_data() (deepmd.dpmodel.FittingOutputDef

method), 381
get_data() (deepmd.dpmodel.ModelOutputDef method), 381
get_data() (deepmd.dpmodel.output_def.FittingOutputDef method), 377
get_data() (deepmd.dpmodel.output_def.ModelOutputDef method), 378
get_data() (deepmd.pt.train.training.Trainer method), 609
get_data() (in module deepmd.utils.data_system), 1034
get_data_dict() (deepmd.tf.train.trainer.DatasetLoader method), 936
get_data_dict() (deepmd.tf.utils.data.DeepmdData method), 945
get_data_dict() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 948
get_data_dict() (deepmd.tf.utils.DeepmdData method), 986
get_data_dict() (deepmd.tf.utils.DeepmdDataSystem method), 989
get_data_dict() (deepmd.utils.data.DeepmdData method), 1027
get_data_dict() (deepmd.utils.data_system.DeepmdDataSystem method), 1031
get_deepmd_jdata() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 913
get_deriv_name() (in module deepmd.dpmodel), 382
get_deriv_name() (in module deepmd.dpmodel.output_def), 378
get_deriv_name_mag() (in module deepmd.dpmodel.output_def), 378
get_descriptor() (deepmd.pt.model.model.dp_model.DPModel method), 530
get_descriptor() (deepmd.pt.model.model.DPModelCommon method), 542
get_device_id() (deepmd.pt.optimizer.LKF.LKFOptimizer method), 607
get_device_id() (deepmd.pt.optimizer.LKFOptimizer method), 608
get_dim_aparam() (deepmd.dpmodel.atomic_model.dp_atomic_model.DPAtomicModel method), 264
get_dim_aparam() (deepmd.dpmodel.atomic_model.DPAtomicModel method), 274
get_dim_aparam() (deepmd.dpmodel.atomic_model.linear_atomic_model.LinearAtomicModel method), 266
get_dim_aparam() (deepmd.dpmodel.atomic_model.LinearAtomicModel method), 277
get_dim_aparam() (deepmd.dpmodel.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 271
get_dim_aparam() (deepmd.dpmodel.atomic_model.PairTabAtomicModel method), 280
get_dim_aparam() (deepmd.dpmodel.fitting.general_fitting.GeneralFitting method), 319
get_dim_aparam() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 339
get_dim_aparam() (deepmd.dpmodel.model.spin_model.SpinModel method), 345
get_dim_aparam() (deepmd.dpmodel.model.SpinModel method), 349
get_dim_aparam() (deepmd.infer.deep_eval.DeepEval method), 393
get_dim_aparam() (deepmd.infer.deep_eval.DeepEvalBackend method), 391
get_dim_aparam() (deepmd.infer.DeepEval method), 407
get_dim_aparam() (deepmd.pt.infer.deep_eval.DeepEval method), 417
get_dim_aparam() (deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel method), 441
get_dim_aparam() (deepmd.pt.model.atomic_model.DPAtomicModel method), 456
get_dim_aparam() (deepmd.pt.model.atomic_model.linear_atomic_model.LinearAtomicModel method), 444
get_dim_aparam() (deepmd.pt.model.atomic_model.LinearEnergyAtomicModel method), 460
get_dim_aparam() (deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 449
get_dim_aparam() (deepmd.pt.model.atomic_model.PairTabAtomicModel method), 464
get_dim_aparam() (deepmd.pt.model.model.frozen.FrozenModel method), 533
get_dim_aparam() (deepmd.pt.model.model.FrozenModel method), 543
get_dim_aparam() (deepmd.pt.model.model.spin_model.SpinModel method), 538
get_dim_aparam() (deepmd.pt.model.model.SpinModel method), 547
get_dim_aparam() (deepmd.pt.model.task.fitting.GeneralFitting method), 590
get_dim_aparam() (deepmd.tf.DeepEval method), 1009
get_dim_aparam() (deepmd.tf.infer.deep_eval.DeepEval method), 802
get_dim_aparam() (deepmd.tf.infer.deep_tensor.DeepTensor method), 812
get_dim_aparam() (deepmd.tf.infer.DeepEval method), 827
get_dim_atom_type() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 301
get_dim_emb() (deepmd.dpmodel.descriptor.DescrptHybrid method), 303
get_dim_emb() (deepmd.dpmodel.descriptor.DescrptSeA method), 306
get_dim_emb() (deepmd.dpmodel.descriptor.DescrptSeR method), 308
get_dim_emb() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1

method), 285
get_dim_emb() (deepmd.dpmodel.descriptor.hybrid.DescrptHybrid method), 288
get_dim_emb() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA method), 293
get_dim_emb() (deepmd.dpmodel.descriptor.se_r.DescrptSeR method), 295
get_dim_emb() (deepmd.pt.model.descriptor.descriptor.Descriptor method), 469
get_dim_emb() (deepmd.pt.model.descriptor.DescriptorBlock method), 506
get_dim_emb() (deepmd.pt.model.descriptor.DescrptBlockRepformers method), 521
get_dim_emb() (deepmd.pt.model.descriptor.DescrptBlockSeA method), 523
get_dim_emb() (deepmd.pt.model.descriptor.DescrptBlockSeAtten method), 507
get_dim_emb() (deepmd.pt.model.descriptor.DescrptDPA1 method), 512
get_dim_emb() (deepmd.pt.model.descriptor.DescrptDPA2 method), 515
get_dim_emb() (deepmd.pt.model.descriptor.DescrptHybrid method), 519
get_dim_emb() (deepmd.pt.model.descriptor.DescrptSeA method), 524
get_dim_emb() (deepmd.pt.model.descriptor.DescrptSeR method), 526
get_dim_emb() (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 method), 473
get_dim_emb() (deepmd.pt.model.descriptor.dpa2.DescrptDPA2 method), 477
get_dim_emb() (deepmd.pt.model.descriptor.hybrid.DescrptHybrid method), 482
get_dim_emb() (deepmd.pt.model.descriptor.repformers.DescrptBlockRepformers method), 491
get_dim_emb() (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA method), 495
get_dim_emb() (deepmd.pt.model.descriptor.se_a.DescrptSeA method), 492
get_dim_emb() (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtten method), 497
get_dim_emb() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 503
get_dim_fparam() (deepmd.dpmodel.atomic_model.dp_atomic_model.DPAtomicModel method), 264
get_dim_fparam() (deepmd.dpmodel.atomic_model.DPAtomicModel method), 274
get_dim_fparam() (deepmd.dpmodel.atomic_model.linear_atomic_model.LinearEnergyAtomicModel method), 266
get_dim_fparam() (deepmd.dpmodel.atomic_model.LinearEnergyAtomicModel method), 277
get_dim_fparam() (deepmd.dpmodel.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 271
get_dim_fparam() (deepmd.dpmodel.atomic_model.PairTabAtomicModel method), 280
get_dim_fparam() (deepmd.dpmodel.fitting.general_fitting.GeneralFitting method), 319
get_dim_fparam() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 339
get_dim_fparam() (deepmd.dpmodel.model.spin_model.SpinModel method), 345
get_dim_fparam() (deepmd.dpmodel.model.SpinModel method), 349
get_dim_fparam() (deepmd.infer.deep_eval.DeepEval method), 393
get_dim_fparam() (deepmd.infer.deep_eval.DeepEvalBackend method), 391
get_dim_fparam() (deepmd.infer.DeepEval method), 407
get_dim_fparam() (deepmd.pt.infer.deep_eval.DeepEval method), 417
get_dim_fparam() (deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel method), 441
get_dim_fparam() (deepmd.pt.model.atomic_model.DPAtomicModel method), 456
get_dim_fparam() (deepmd.pt.model.atomic_model.linear_atomic_model.LinearEnergyAtomicModel method), 444
get_dim_fparam() (deepmd.pt.model.atomic_model.LinearEnergyAtomicModel method), 460
get_dim_fparam() (deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 449
get_dim_fparam() (deepmd.pt.model.atomic_model.PairTabAtomicModel method), 464
get_dim_fparam() (deepmd.pt.model.model.frozen.FrozenModel method), 533
get_dim_fparam() (deepmd.pt.model.model.FrozenModel method), 543
get_dim_fparam() (deepmd.pt.model.model.spin_model.SpinModel method), 538
get_dim_fparam() (deepmd.pt.model.model.SpinModel method), 547
get_dim_fparam() (deepmd.pt.model.task.fitting.GeneralFitting method), 590
get_dim_fparam() (deepmd.tf.DeepEval method), 1009
get_dim_fparam() (deepmd.tf.infer.deep_eval.DeepEval method), 802
get_dim_fparam() (deepmd.tf.infer.deep_tensor.DeepTensor method), 812
get_dim_fparam() (deepmd.tf.infer.DeepEval method), 827
get_dim_in() (deepmd.pt.model.descriptor.descriptor.Descriptor method), 469
get_dim_in() (deepmd.pt.model.descriptor.DescriptorBlock method), 506
get_dim_in() (deepmd.pt.model.descriptor.DescrptBlockRepformers method), 521
get_dim_in() (deepmd.pt.model.descriptor.DescrptBlockSeA

method), 523
get_dim_in() (deepmd.pt.model.descriptor.DescrptBlockSeAtten method), 507
get_dim_in() (deepmd.pt.model.descriptor.DescrptSeR method), 526
get_dim_in() (deepmd.pt.model.descriptor.repformers.DescrptBlockRepformers method), 491
get_dim_in() (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA method), 495
get_dim_in() (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtten method), 497
get_dim_in() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 503
get_dim_out() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 301
get_dim_out() (deepmd.dpmodel.descriptor.DescrptHybrid method), 303
get_dim_out() (deepmd.dpmodel.descriptor.DescrptSeA method), 306
get_dim_out() (deepmd.dpmodel.descriptor.DescrptSeR method), 308
get_dim_out() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 method), 285
get_dim_out() (deepmd.dpmodel.descriptor.hybrid.DescrptHybrid method), 288
get_dim_out() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA method), 292
get_dim_out() (deepmd.dpmodel.descriptor.se_r.DescrptSeR method), 295
get_dim_out() (deepmd.pt.model.descriptor.descriptor.DescriptorBlock method), 469
get_dim_out() (deepmd.pt.model.descriptor.DescriptorBlock method), 506
get_dim_out() (deepmd.pt.model.descriptor.DescrptBlockRepformers method), 521
get_dim_out() (deepmd.pt.model.descriptor.DescrptBlockSeA method), 523
get_dim_out() (deepmd.pt.model.descriptor.DescrptBlockSeAtten method), 507
get_dim_out() (deepmd.pt.model.descriptor.DescrptDPA1 method), 512
get_dim_out() (deepmd.pt.model.descriptor.DescrptDPA2 method), 515
get_dim_out() (deepmd.pt.model.descriptor.DescrptHybrid method), 519
get_dim_out() (deepmd.pt.model.descriptor.DescrptSeA method), 524
get_dim_out() (deepmd.pt.model.descriptor.DescrptSeR method), 526
get_dim_out() (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 method), 473
get_dim_out() (deepmd.pt.model.descriptor.dpa2.DescrptDPA2 method), 477
get_dim_out() (deepmd.pt.model.descriptor.hybrid.DescrptHybrid method), 482
get_dim_out() (deepmd.pt.model.descriptor.repformers.DescrptBlockRepformers method), 491
get_dim_out() (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA method), 495
get_dim_out() (deepmd.pt.model.descriptor.se_a.DescrptSeA method), 492
get_dim_out() (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtten method), 497
get_dim_out() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 503
get_dim_out() (deepmd.tf.descriptor.Descriptor method), 699
get_dim_out() (deepmd.tf.descriptor.descriptor.Descriptor method), 641
get_dim_out() (deepmd.tf.descriptor.DescrptHybrid method), 706
get_dim_out() (deepmd.tf.descriptor.DescrptLocFrame method), 711
get_dim_out() (deepmd.tf.descriptor.DescrptSeA method), 715
get_dim_out() (deepmd.tf.descriptor.DescrptSeAEf method), 723
get_dim_out() (deepmd.tf.descriptor.DescrptSeR method), 739
get_dim_out() (deepmd.tf.descriptor.DescrptSeT method), 743
get_dim_out() (deepmd.tf.descriptor.hybrid.DescrptHybrid method), 648
get_dim_out() (deepmd.tf.descriptor.loc_frame.DescrptLocFrame method), 653
get_dim_out() (deepmd.tf.descriptor.se_a.DescrptSeA method), 661
get_dim_out() (deepmd.tf.descriptor.se_a_ef.DescrptSeAEf method), 669
get_dim_out() (deepmd.tf.descriptor.se_r.DescrptSeR method), 691
get_dim_out() (deepmd.tf.descriptor.se_t.DescrptSeT method), 695
get_dim_rot_mat_1() (deepmd.tf.descriptor.Descriptor method), 700
get_dim_rot_mat_1() (deepmd.tf.descriptor.descriptor.Descriptor method), 642
get_dim_rot_mat_1() (deepmd.tf.descriptor.DescrptSeA method), 715
get_dim_rot_mat_1() (deepmd.tf.descriptor.DescrptSeAEf method), 723
get_dim_rot_mat_1() (deepmd.tf.descriptor.se_a.DescrptSeA method), 661

get_dim_rot_mat_1() (deepmd.tf.descriptor.se_a_ef.DescrptSeAEf method), 669
get_dp_init_weights() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 913
get_dscp_jdata() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
get_embedding_net_nodes() (in module deepmd.tf.utils.graph), 954
get_embedding_net_nodes_from_graph_def() (in module deepmd.tf.utils.graph), 954
get_embedding_net_variables() (in module deepmd.tf.utils.graph), 955
get_embedding_net_variables_from_graph_def() (in module deepmd.tf.utils.graph), 954
get_env() (in module deepmd.tf.lmp), 1004
get_evaluation_results() (deepmd.tf.train.trainer.DPTrainer method), 935
get_exclude_types() (deepmd.dpmodel.utils.AtomExcludeMask method), 367
get_exclude_types() (deepmd.dpmodel.utils.exclude_mask.AtomExcludeMask method), 353
get_exclude_types() (deepmd.dpmodel.utils.exclude_mask.PairExcludeMask method), 353
get_exclude_types() (deepmd.dpmodel.utils.PairExcludeMask method), 368
get_exclude_types() (deepmd.pt.utils.AtomExcludeMask method), 638
get_exclude_types() (deepmd.pt.utils.exclude_mask.AtomExcludeMask method), 620
get_exclude_types() (deepmd.pt.utils.exclude_mask.PairExcludeMask method), 620
get_exclude_types() (deepmd.pt.utils.PairExcludeMask method), 638
get_extra_embedding_net_nodes_from_graph_def() (in module deepmd.tf.utils.graph), 955
get_extra_embedding_net_suffix() (in module deepmd.tf.utils.graph), 954
get_extra_embedding_net_variables_from_graph_def() (in module deepmd.tf.utils.graph), 955
get_extra_side_embedding_net_variable() (in module deepmd.tf.utils.compress), 940
get_extra_state() (deepmd.pt.train.wrapper.ModelWrapper method), 611
get_face_distance() (deepmd.pt.utils.preprocess.Region3D method), 630
get_feed_dict() (deepmd.tf.model.model.Model method), 865
get_feed_dict() (deepmd.tf.model.pairwise_dprc.PairwiseDPRc method), 876
get_feed_dict() (deepmd.tf.model.PairwiseDPRc method), 888
get_feed_dict() (deepmd.tf.train.trainer.DPTrainer method), 935
get_file_list() (deepmd.tf.nvnmd.utils.fio.Fio method), 915
get_filter_type_weight() (in module deepmd.tf.nvnmd.utils.weight), 919
get_filter_weight() (in module deepmd.tf.nvnmd.utils), 923
get_filter_weight() (in module deepmd.tf.nvnmd.utils.weight), 919
get_fitn_jdata() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
get_fitnet_weight() (in module deepmd.tf.nvnmd.utils), 923
get_fitnet_weight() (in module deepmd.tf.nvnmd.utils.weight), 919
get_fitting() (deepmd.tf.model.frozen.FrozenModel method), 857
get_fitting() (deepmd.tf.model.FrozenModel method), 882
get_fitting() (deepmd.tf.model.linear.LinearModel method), 858
get_fitting() (deepmd.tf.model.model.Model method), 865
get_fitting() (deepmd.tf.model.model.StandardModel method), 867
get_fitting() (deepmd.tf.model.multi.MultiModel method), 871
get_fitting() (deepmd.tf.model.MultiModel method), 895
get_fitting() (deepmd.tf.model.pairtab.PairTabModel method), 873
get_fitting() (deepmd.tf.model.PairTabModel method), 885
get_fitting() (deepmd.tf.model.pairwise_dprc.PairwiseDPRc method), 875
get_fitting() (deepmd.tf.model.PairwiseDPRc method), 887
get_fitting_net() (deepmd.pt.model.model.dp_model.DPModelCo method), 530
get_fitting_net() (deepmd.pt.model.model.DPModelCommon method), 542
get_fitting_net_nodes() (in module deepmd.tf.utils.graph), 956
get_fitting_net_nodes_from_graph_def() (in

module deepmd.tf.utils.graph), 956
get_fitting_net_variables() (in module deepmd.tf.utils.graph), 956
get_fitting_net_variables_from_graph_def() (in module deepmd.tf.utils.graph), 956
get_global_step() (deepmd.tf.train.trainer.DPTrainer method), 935
get_gpus() (in module deepmd.tf.cluster.local), 639
get_has_efield() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 339
get_has_efield() (deepmd.infer.deep_eval.DeepEvalBackend method), 392
get_has_efield() (deepmd.pt.infer.deep_eval.DeepEval method), 417
get_has_efield() (deepmd.tf.DeepEval method), 1011
get_has_efield() (deepmd.tf.infer.deep_eval.DeepEval method), 804
get_has_efield() (deepmd.tf.infer.DeepEval method), 829
get_has_spin() (deepmd.infer.deep_eval.DeepEvalBackend method), 392
get_has_spin() (deepmd.pt.infer.deep_eval.DeepEval method), 417
get_hash() (deepmd.pt.utils.env_mat_stat.EnvMatStat method), 619
get_hessian_name() (in module deepmd.dpmodel), 382
get_hessian_name() (in module deepmd.dpmodel.output_def), 378
get_host_names() (in module deepmd.utils.hostlist), 1039
get_item_torch() (deepmd.tf.utils.data.DeepmdData method), 945
get_item_torch() (deepmd.tf.utils.DeepmdData method), 986
get_item_torch() (deepmd.utils.data.DeepmdData method), 1027
get_leading_dims() (in module deepmd.pt.model.model.transform_output), 540
get_learning_rate_jdata() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
get_library_path() (in module deepmd.tf.lmp), 1004
get_ll() (in module deepmd.main), 1064
get_ll() (in module deepmd.tf.entrypoints.main), 750
get_loss() (deepmd.tf.fit.dipole.DipoleFittingSeA method), 762
get_loss() (deepmd.tf.fit.DipoleFittingSeA method), 780
get_loss() (deepmd.tf.fit.dos.DOSFitting method), 766
get_loss() (deepmd.tf.fit.DOSFitting method), 783
get_loss() (deepmd.tf.fit.ener.EnerFitting method), 770
get_loss() (deepmd.tf.fit.EnerFitting method), 786
get_loss() (deepmd.tf.fit.Fitting method), 788
get_loss() (deepmd.tf.fit.fitting.Fitting method), 771
get_loss() (deepmd.tf.fit.GlobalPolarFittingSeA method), 791
get_loss() (deepmd.tf.fit.polar.GlobalPolarFittingSeA method), 777
get_loss() (deepmd.tf.fit.polar.PolarFittingSeA method), 775
get_loss() (deepmd.tf.fit.PolarFittingSeA method), 793
get_loss() (deepmd.tf.model.frozen.FrozenModel method), 857
get_loss() (deepmd.tf.model.FrozenModel method), 882
get_loss() (deepmd.tf.model.linear.LinearModel method), 859
get_loss() (deepmd.tf.model.model.Model method), 865
get_loss() (deepmd.tf.model.model.StandardModel method), 867
get_loss() (deepmd.tf.model.multi.MultiModel method), 871
get_loss() (deepmd.tf.model.MultiModel method), 896
get_loss() (deepmd.tf.model.pairtab.PairTabModel method), 873
get_loss() (deepmd.tf.model.PairTabModel method), 885
get_loss() (deepmd.tf.model.pairwise_dprc.PairwiseDPRc method), 875
get_loss() (deepmd.tf.model.PairwiseDPRc method), 887
get_loss_jdata() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
get_min_nbor_dist() (deepmd.utils.update_sel.BaseUpdateSel method), 1057
get_model() (in module deepmd.dpmodel.model.model), 344
get_model() (in module deepmd.pt.model.model), 548
get_model_def_script() (deepmd.dpmodel.model.base_model.BaseModel method), 342
get_model_def_script() (deepmd.dpmodel.model.spin_model.SpinModel method), 345
get_model_def_script()

(deepmd.dpmodel.model.SpinModel method), 349
get_model_def_script() (deepmd.pt.model.model.BaseModel method), 546
get_model_def_script() (deepmd.pt.model.model.frozen.FrozenModel method), 534
get_model_def_script() (deepmd.pt.model.model.FrozenModel method), 544
get_model_def_script() (deepmd.pt.model.model.model.BaseModel method), 536
get_model_def_script() (deepmd.pt.model.model.spin_model.SpinModel method), 539
get_model_def_script() (deepmd.pt.model.model.SpinModel method), 547
get_model_jdata() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
get_model_nsels() (deepmd.dpmodel.atomic_model.linear_atomic_model.DPZBLLinearAtomicModel method), 265
get_model_nsels() (deepmd.dpmodel.atomic_model.linear_atomic_model.LinearAtomicModel method), 276
get_model_nsels() (deepmd.pt.model.atomic_model.linear_atomic_model.DPZBLLinearAtomicModel method), 443
get_model_nsels() (deepmd.pt.model.atomic_model.linear_atomic_model.LinearAtomicModel method), 458
get_model_rcuts() (deepmd.dpmodel.atomic_model.linear_atomic_model.DPZBLLinearAtomicModel method), 265
get_model_rcuts() (deepmd.dpmodel.atomic_model.linear_atomic_model.LinearAtomicModel method), 276
get_model_rcuts() (deepmd.pt.model.atomic_model.linear_atomic_model.DPZBLLinearAtomicModel method), 443
get_model_rcuts() (deepmd.pt.model.atomic_model.linear_atomic_model.LinearAtomicModel method), 458
get_model_sels() (deepmd.dpmodel.atomic_model.linear_atomic_model.DPZBLLinearAtomicModel method), 265
get_model_sels() (deepmd.dpmodel.atomic_model.linear_atomic_model.LinearAtomicModel method), 276
get_model_sels() (deepmd.pt.model.atomic_model.linear_atomic_model.DPZBLLinearAtomicModel method), 443
get_model_sels() (deepmd.pt.model.atomic_model.linear_atomic_model.LinearAtomicModel method), 458
get_multiple_nlist_key() (in module deepmd.dpmodel.utils), 372
get_multiple_nlist_key() (in module deepmd.dpmodel.utils.nlist), 360
get_multiple_nlist_key() (in module deepmd.pt.utils.nlist), 627
get_natoms() (deepmd.tf.utils.data.DeepmdData method), 946
get_natoms() (deepmd.tf.utils.DeepmdData method), 987
get_natoms() (deepmd.utils.data.DeepmdData method), 1027
get_natoms_vec() (deepmd.tf.utils.data.DeepmdData method), 946
get_natoms_vec() (deepmd.tf.utils.DeepmdData method), 987
get_natoms_vec() (deepmd.utils.data.DeepmdData method), 1027
get_nbatches() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 950
get_nbatches() (deepmd.tf.utils.DeepmdDataSystem method), 990
get_nbatches() (deepmd.utils.data_system.DeepmdDataSystem method), 1032
get_nbor_stat() (deepmd.utils.update_sel.BaseUpdateSel method), 1057
get_ngpus() (deepmd.pt.entrypoints.main.SummaryPrinter method), 415
get_ngpus() (deepmd.utils.summary.SummaryPrinter method), 1056
get_nlist() (deepmd.dpmodel.descriptor.DescrptSeA method), 700
get_nlist() (deepmd.dpmodel.descriptor.descriptor.Descriptor method), 642
get_nlist() (deepmd.dpmodel.descriptor.hybrid.DescrptHybrid method), 706
get_nlist() (deepmd.dpmodel.descriptor.loc_frame.DescrptLocFrame method), 711
get_nlist() (deepmd.dpmodel.descriptor.se_a.DescrptSeA method), 715
get_nlist() (deepmd.dpmodel.descriptor.se_a_ef.DescrptSeAEf method), 723
get_nlist() (deepmd.dpmodel.descriptor.se_r.DescrptSeR method), 739
get_nlist() (deepmd.dpmodel.descriptor.se_t.DescrptSeT method), 743
get_nlist() (deepmd.pt.model.descriptor.hybrid.DescrptHybrid method), 648
get_nlist() (deepmd.pt.model.descriptor.loc_frame.DescrptLocFrame method), 653
get_nlist() (deepmd.pt.model.descriptor.se_a.DescrptSeA method), 661
get_nlist() (deepmd.pt.model.descriptor.se_a_ef.DescrptSeAEf method), 669
get_nlist() (deepmd.tf.descriptor.se_r.DescrptSeR method), 691
get_nlist() (deepmd.tf.descriptor.se_t.DescrptSeT method), 695
get_nlist_i() (deepmd.tf.descriptor.DescrptHybrid method), 706
get_nlist_i() (deepmd.tf.descriptor.hybrid.DescrptHybrid method), 648

get_nnei() (deepmd.dpmodel.model.spin_model.SpinModel method), 345
get_nnei() (deepmd.dpmodel.model.SpinModel method), 349
get_nnei() (deepmd.pt.model.model.frozen.FrozenModel method), 534
get_nnei() (deepmd.pt.model.model.FrozenModel method), 544
get_nnei() (deepmd.pt.model.model.spin_model.SpinModel method), 539
get_nnei() (deepmd.pt.model.model.SpinModel method), 548
get_normalize() (in module deepmd.tf.nvnmd.utils.weight), 919
get_np_precision() (in module deepmd.common), 1062
get_np_precision() (in module deepmd.tf.common), 998
get_nsel() (deepmd.dpmodel.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 269
get_nsel() (deepmd.dpmodel.atomic_model.PairTabAtomicModel method), 278
get_nsel() (deepmd.dpmodel.model.spin_model.SpinModel method), 345
get_nsel() (deepmd.dpmodel.model.SpinModel method), 349
get_nsel() (deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 447
get_nsel() (deepmd.pt.model.atomic_model.PairTabAtomicModel method), 461
get_nsel() (deepmd.pt.model.descriptor.descriptor.DescriptorBlock method), 468
get_nsel() (deepmd.pt.model.descriptor.DescriptorBlock method), 506
get_nsel() (deepmd.pt.model.descriptor.DescrptBlockRepformers method), 521
get_nsel() (deepmd.pt.model.descriptor.DescrptBlockSeA method), 523
get_nsel() (deepmd.pt.model.descriptor.DescrptBlockSeAtten method), 507
get_nsel() (deepmd.pt.model.descriptor.DescrptDPA1 method), 512
get_nsel() (deepmd.pt.model.descriptor.DescrptDPA2 method), 515
get_nsel() (deepmd.pt.model.descriptor.DescrptSeA method), 524
get_nsel() (deepmd.pt.model.descriptor.DescrptSeR method), 526
get_nsel() (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 method), 473
get_nsel() (deepmd.pt.model.descriptor.dpa2.DescrptDPA2 method), 477
get_nsel() (deepmd.pt.model.descriptor.repformers.DescrptBlockRepformers method), 491
get_nsel() (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA method), 494
get_nsel() (deepmd.pt.model.descriptor.se_a.DescrptSeA method), 492
get_nsel() (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtten method), 497
get_nsel() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 503
get_nsel() (deepmd.pt.model.model.frozen.FrozenModel method), 534
get_nsel() (deepmd.pt.model.model.FrozenModel method), 544
get_nsel() (deepmd.pt.model.model.spin_model.SpinModel method), 539
get_nsel() (deepmd.pt.model.model.SpinModel method), 548
get_nsystems() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 950
get_nsystems() (deepmd.utils.data_system.DeepmdDataSystem method), 991
get_nsystems() (deepmd.utils.data_system.DeepmdDataSystem method), 1033
get_ntypes() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 301
get_ntypes() (deepmd.dpmodel.descriptor.DescrptHybrid method), 302
get_ntypes() (deepmd.dpmodel.descriptor.DescrptSeA method), 306
get_ntypes() (deepmd.dpmodel.descriptor.DescrptSeR method), 309
get_ntypes() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 method), 285
get_ntypes() (deepmd.dpmodel.descriptor.hybrid.DescrptHybrid method), 288
get_ntypes() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA method), 293
get_ntypes() (deepmd.dpmodel.descriptor.se_r.DescrptSeR method), 296
get_ntypes() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 339
get_ntypes() (deepmd.infer.deep_eval.DeepEval method), 393
get_ntypes() (deepmd.infer.deep_eval.DeepEvalBackend method), 391
get_ntypes() (deepmd.infer.DeepEval method), 407
get_ntypes() (deepmd.pt.infer.deep_eval.DeepEval method), 417
get_ntypes() (deepmd.pt.model.descriptor.descriptor.DescriptorBlock method), 469
get_ntypes() (deepmd.pt.model.descriptor.DescriptorBlock method), 506
get_ntypes() (deepmd.pt.model.descriptor.DescrptBlockRepformers method), 521
get_ntypes() (deepmd.pt.model.descriptor.DescrptBlockSeA

method), 523
get_ntypes() (deepmd.pt.model.descriptor.DescrptBlockSeAtten method), 507
get_ntypes() (deepmd.pt.model.descriptor.DescrptDPA1 method), 512
get_ntypes() (deepmd.pt.model.descriptor.DescrptDPA2 method), 515
get_ntypes() (deepmd.pt.model.descriptor.DescrptHybrid method), 519
get_ntypes() (deepmd.pt.model.descriptor.DescrptSeA method), 524
get_ntypes() (deepmd.pt.model.descriptor.DescrptSeR method), 526
get_ntypes() (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 method), 473
get_ntypes() (deepmd.pt.model.descriptor.dpa2.DescrptDPA2 method), 477
get_ntypes() (deepmd.pt.model.descriptor.hybrid.DescrptHybrid method), 482
get_ntypes() (deepmd.pt.model.descriptor.repformers.DescrptBlockRepformers method), 491
get_ntypes() (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA method), 495
get_ntypes() (deepmd.pt.model.descriptor.se_a.DescrptSeA method), 492
get_ntypes() (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtten method), 497
get_ntypes() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 503
get_ntypes() (deepmd.pt.model.model.BaseModel method), 546
get_ntypes() (deepmd.pt.model.model.model.BaseModel method), 536
get_ntypes() (deepmd.tf.DeepEval method), 1009
get_ntypes() (deepmd.tf.descriptor.Descriptor method), 699
get_ntypes() (deepmd.tf.descriptor.descriptor.Descriptor method), 641
get_ntypes() (deepmd.tf.descriptor.DescrptHybrid method), 706
get_ntypes() (deepmd.tf.descriptor.DescrptLocFrame method), 711
get_ntypes() (deepmd.tf.descriptor.DescrptSeA method), 715
get_ntypes() (deepmd.tf.descriptor.DescrptSeAEf method), 723
get_ntypes() (deepmd.tf.descriptor.DescrptSeR method), 739
get_ntypes() (deepmd.tf.descriptor.DescrptSeT method), 743
get_ntypes() (deepmd.tf.descriptor.hybrid.DescrptHybrid method), 648
get_ntypes() (deepmd.tf.descriptor.loc_frame.DescrptLocFrame method), 653
get_ntypes() (deepmd.tf.descriptor.se_a.DescrptSeA method), 661
get_ntypes() (deepmd.tf.descriptor.se_a_ef.DescrptSeAEf method), 669
get_ntypes() (deepmd.tf.descriptor.se_r.DescrptSeR method), 690
get_ntypes() (deepmd.tf.descriptor.se_t.DescrptSeT method), 695
get_ntypes() (deepmd.tf.infer.deep_eval.DeepEval method), 802
get_ntypes() (deepmd.tf.infer.deep_tensor.DeepTensor method), 812
get_ntypes() (deepmd.tf.infer.DeepEval method), 827
get_ntypes() (deepmd.tf.model.dos.DOSModel method), 851
get_ntypes() (deepmd.tf.model.DOSModel method), 889
get_ntypes() (deepmd.tf.model.ener.EnerModel method), 854
get_ntypes() (deepmd.tf.model.EnerModel method), 891
get_ntypes() (deepmd.tf.model.frozen.FrozenModel method), 857
get_ntypes() (deepmd.tf.model.FrozenModel method), 882
get_ntypes() (deepmd.tf.model.linear.LinearModel method), 859
get_ntypes() (deepmd.tf.model.model.Model method), 865
get_ntypes() (deepmd.tf.model.model.StandardModel method), 867
get_ntypes() (deepmd.tf.model.multi.MultiModel method), 870
get_ntypes() (deepmd.tf.model.MultiModel method), 894
get_ntypes() (deepmd.tf.model.pairtab.PairTabModel method), 873
get_ntypes() (deepmd.tf.model.PairTabModel method), 886
get_ntypes() (deepmd.tf.model.pairwise_dprc.PairwiseDPRc method), 875
get_ntypes() (deepmd.tf.model.PairwiseDPRc method), 887
get_ntypes() (deepmd.tf.model.tensor.TensorModel method), 878
get_ntypes() (deepmd.tf.utils.data.DeepmdData method), 945
get_ntypes() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 950
get_ntypes() (deepmd.tf.utils.DeepmdData method), 987
get_ntypes() (deepmd.tf.utils.DeepmdDataSystem method), 991

get_ntypes() (deepmd.utils.data.DeepmdData method), 1027
get_ntypes() (deepmd.utils.data_system.DeepmdDataSystem method), 1032
get_ntypes_input() (deepmd.utils.spin.Spin method), 1054
get_ntypes_real() (deepmd.utils.spin.Spin method), 1054
get_ntypes_real_and_spin() (deepmd.utils.spin.Spin method), 1054
get_ntypes_spin() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 340
get_ntypes_spin() (deepmd.infer.deep_eval.DeepEval method), 394
get_ntypes_spin() (deepmd.infer.deep_eval.DeepEvalBackend method), 392
get_ntypes_spin() (deepmd.infer.DeepEval method), 408
get_ntypes_spin() (deepmd.pt.infer.deep_eval.DeepEval method), 417
get_ntypes_spin() (deepmd.tf.DeepEval method), 1009
get_ntypes_spin() (deepmd.tf.infer.deep_eval.DeepEval method), 802
get_ntypes_spin() (deepmd.tf.infer.DeepEval method), 827
get_ntypes_spin() (deepmd.tf.utils.spin.Spin method), 979
get_ntypes_spin() (deepmd.utils.spin.Spin method), 1054
get_numb_aparam() (deepmd.tf.fit.dos.DOSFitting method), 764
get_numb_aparam() (deepmd.tf.fit.DOSFitting method), 781
get_numb_aparam() (deepmd.tf.fit.ener.EnerFitting method), 768
get_numb_aparam() (deepmd.tf.fit.EnerFitting method), 785
get_numb_aparam() (deepmd.tf.model.dos.DOSModel method), 851
get_numb_aparam() (deepmd.tf.model.DOSModel method), 889
get_numb_aparam() (deepmd.tf.model.ener.EnerModel method), 854
get_numb_aparam() (deepmd.tf.model.EnerModel method), 892
get_numb_aparam() (deepmd.tf.model.model.Model method), 864
get_numb_aparam() (deepmd.tf.model.multi.MultiModel method), 871
get_numb_aparam() (deepmd.tf.model.MultiModel method), 895
get_numb_batch() (deepmd.tf.utils.data.DeepmdData method), 946
get_numb_batch() (deepmd.tf.utils.DeepmdData method), 987
get_numb_batch() (deepmd.utils.data.DeepmdData method), 1027
get_numb_dos() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 339
get_numb_dos() (deepmd.infer.deep_dos.DeepDOS method), 389
get_numb_dos() (deepmd.infer.deep_eval.DeepEvalBackend method), 392
get_numb_dos() (deepmd.pt.infer.deep_eval.DeepEval method), 417
get_numb_dos() (deepmd.pt.model.model.dos_model.DOSModel method), 530
get_numb_dos() (deepmd.tf.DeepEval method), 1011
get_numb_dos() (deepmd.tf.fit.dos.DOSFitting method), 764
get_numb_dos() (deepmd.tf.fit.DOSFitting method), 781
get_numb_dos() (deepmd.tf.infer.deep_dos.DeepDOS method), 798
get_numb_dos() (deepmd.tf.infer.deep_eval.DeepEval method), 804
get_numb_dos() (deepmd.tf.infer.DeepDOS method), 823
get_numb_dos() (deepmd.tf.infer.DeepEval method), 829
get_numb_dos() (deepmd.tf.model.dos.DOSModel method), 851
get_numb_dos() (deepmd.tf.model.DOSModel method), 889
get_numb_dos() (deepmd.tf.model.model.Model method), 864
get_numb_dos() (deepmd.tf.model.multi.MultiModel method), 871
get_numb_dos() (deepmd.tf.model.MultiModel method), 895
get_numb_fparam() (deepmd.tf.fit.dos.DOSFitting method), 764
get_numb_fparam() (deepmd.tf.fit.DOSFitting method), 781
get_numb_fparam() (deepmd.tf.fit.ener.EnerFitting method), 768
get_numb_fparam() (deepmd.tf.fit.EnerFitting method), 785
get_numb_fparam() (deepmd.tf.model.dos.DOSModel method), 851
get_numb_fparam() (deepmd.tf.model.DOSModel method), 889
get_numb_fparam() (deepmd.tf.model.ener.EnerModel method), 854
get_numb_fparam() (deepmd.tf.model.EnerModel method), 891

get_numb_fparam() (deepmd.tf.model.model.Model method), 864
get_numb_fparam() (deepmd.tf.model.multi.MultiModel method), 871
get_numb_fparam() (deepmd.tf.model.MultiModel method), 895
get_numb_set() (deepmd.tf.utils.data.DeepmdData method), 946
get_numb_set() (deepmd.tf.utils.DeepmdData method), 987
get_numb_set() (deepmd.utils.data.DeepmdData method), 1027
get_nvnmd_jdata() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
get_op_dir() (in module deepmd.tf.lmp), 1005
get_out_bias() (deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel method), 441
get_out_bias() (deepmd.pt.model.atomic_model.DPAtomicModel method), 456
get_out_bias() (deepmd.pt.model.atomic_model.linear_atomic_model.LinearEnergyAtomicModel method), 443
get_out_bias() (deepmd.pt.model.atomic_model.LinearEnergyAtomicModel method), 458
get_out_bias() (deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 447
get_out_bias() (deepmd.pt.model.atomic_model.PairTabAtomicModel method), 461
get_out_size() (deepmd.tf.fit.dipole.DipoleFittingSeA method), 761
get_out_size() (deepmd.tf.fit.DipoleFittingSeA method), 779
get_out_size() (deepmd.tf.fit.GlobalPolarFittingSeA method), 790
get_out_size() (deepmd.tf.fit.polar.GlobalPolarFittingSeA method), 776
get_out_size() (deepmd.tf.fit.polar.PolarFittingSeA method), 774
get_out_size() (deepmd.tf.fit.PolarFittingSeA method), 792
get_out_size() (deepmd.tf.model.tensor.TensorModel method), 878
get_pair_exclude_types() (deepmd.utils.spin.Spin method), 1055
get_pattern_nodes_from_graph_def() (in module deepmd.tf.utils.graph), 953
get_plugin() (deepmd.pt.utils.plugin.Plugin method), 629
get_plugin() (deepmd.tf.utils.Plugin method), 994
get_plugin() (deepmd.tf.utils.plugin.Plugin method), 975
get_plugin() (deepmd.utils.plugin.Plugin method), 1051
get_precision() (in module deepmd.tf.common), 1000
get_rcut() (deepmd.dpmodel.atomic_model.dp_atomic_model.DPA method), 263
get_rcut() (deepmd.dpmodel.atomic_model.DPAtomicModel method), 273
get_rcut() (deepmd.dpmodel.atomic_model.linear_atomic_model.Li method), 265
get_rcut() (deepmd.dpmodel.atomic_model.LinearEnergyAtomicMo method), 276
get_rcut() (deepmd.dpmodel.atomic_model.pairtab_atomic_model. method), 269
get_rcut() (deepmd.dpmodel.atomic_model.PairTabAtomicModel method), 278
get_rcut() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 301
get_rcut() (deepmd.dpmodel.descriptor.DescrptHybrid method), 302
get_rcut() (deepmd.dpmodel.descriptor.DescrptSeA method), 306
get_rcut() (deepmd.dpmodel.descriptor.DescrptSeR method), 308
get_rcut() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 method), 285
get_rcut() (deepmd.dpmodel.descriptor.hybrid.DescrptHybrid method), 286
get_rcut() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA method), 293
get_rcut() (deepmd.dpmodel.descriptor.se_r.DescrptSeR method), 296
get_rcut() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 339
get_rcut() (deepmd.dpmodel.model.spin_model.SpinModel method), 345
get_rcut() (deepmd.dpmodel.model.SpinModel method), 349
get_rcut() (deepmd.infer.deep_eval.DeepEval method), 393
get_rcut() (deepmd.infer.deep_eval.DeepEvalBackend method), 391
get_rcut() (deepmd.infer.DeepEval method), 406
get_rcut() (deepmd.pt.infer.deep_eval.DeepEval method), 417
get_rcut() (deepmd.pt.model.atomic_model.dp_atomic_model.DPA method), 440
get_rcut() (deepmd.pt.model.atomic_model.DPAtomicModel method), 455
get_rcut() (deepmd.pt.model.atomic_model.linear_atomic_model.Li method), 443
get_rcut() (deepmd.pt.model.atomic_model.LinearEnergyAtomicMo method), 458
get_rcut() (deepmd.pt.model.atomic_model.pairtab_atomic_model. method), 447
get_rcut() (deepmd.pt.model.atomic_model.PairTabAtomicModel method), 461
get_rcut() (deepmd.pt.model.descriptor.descriptor.DescriptorBlock

method), 468
get_rcut() (deepmd.pt.model.descriptor.DescriptorBlock method), 506
get_rcut() (deepmd.pt.model.descriptor.DescrptBlockRepformers method), 521
get_rcut() (deepmd.pt.model.descriptor.DescrptBlockSeA method), 523
get_rcut() (deepmd.pt.model.descriptor.DescrptBlockSeAtten method), 507
get_rcut() (deepmd.pt.model.descriptor.DescrptDPA1 method), 512
get_rcut() (deepmd.pt.model.descriptor.DescrptDPA2 method), 515
get_rcut() (deepmd.pt.model.descriptor.DescrptHybrid method), 519
get_rcut() (deepmd.pt.model.descriptor.DescrptSeA method), 524
get_rcut() (deepmd.pt.model.descriptor.DescrptSeR method), 526
get_rcut() (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 method), 473
get_rcut() (deepmd.pt.model.descriptor.dpa2.DescrptDPA2 method), 477
get_rcut() (deepmd.pt.model.descriptor.hybrid.DescrptHybrid method), 481
get_rcut() (deepmd.pt.model.descriptor.repformers.DescrptBlockRepformers method), 491
get_rcut() (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA method), 494
get_rcut() (deepmd.pt.model.descriptor.se_a.DescrptSeA method), 492
get_rcut() (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtten method), 497
get_rcut() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 503
get_rcut() (deepmd.pt.model.model.frozen.FrozenModel method), 533
get_rcut() (deepmd.pt.model.model.FrozenModel method), 543
get_rcut() (deepmd.pt.model.model.spin_model.SpinModel method), 538
get_rcut() (deepmd.pt.model.model.SpinModel method), 547
get_rcut() (deepmd.tf.DeepEval method), 1009
get_rcut() (deepmd.tf.descriptor.Descriptor method), 699
get_rcut() (deepmd.tf.descriptor.descriptor.Descriptor method), 641
get_rcut() (deepmd.tf.descriptor.DescrptHybrid method), 705
get_rcut() (deepmd.tf.descriptor.DescrptLocFrame method), 711
get_rcut() (deepmd.tf.descriptor.DescrptSeA method), 715
get_rcut() (deepmd.tf.descriptor.DescrptSeAEf method), 723
get_rcut() (deepmd.tf.descriptor.DescrptSeAMask method), 727
get_rcut() (deepmd.tf.descriptor.DescrptSeR method), 738
get_rcut() (deepmd.tf.descriptor.DescrptSeT method), 743
get_rcut() (deepmd.tf.descriptor.hybrid.DescrptHybrid method), 648
get_rcut() (deepmd.tf.descriptor.loc_frame.DescrptLocFrame method), 653
get_rcut() (deepmd.tf.descriptor.se_a.DescrptSeA method), 661
get_rcut() (deepmd.tf.descriptor.se_a_ef.DescrptSeAEf method), 669
get_rcut() (deepmd.tf.descriptor.se_a_mask.DescrptSeAMask method), 674
get_rcut() (deepmd.tf.descriptor.se_r.DescrptSeR method), 690
get_rcut() (deepmd.tf.descriptor.se_t.DescrptSeT method), 695
get_rcut() (deepmd.tf.infer.deep_eval.DeepEval method), 802
get_rcut() (deepmd.tf.infer.deep_tensor.DeepTensor method), 818
get_rcut() (deepmd.tf.infer.DeepEval method), 827
get_rcut() (deepmd.tf.model.dos.DOSModel method), 851
get_rcut() (deepmd.tf.model.DOSModel method), 889
get_rcut() (deepmd.tf.model.ener.EnerModel method), 854
get_rcut() (deepmd.tf.model.EnerModel method), 891
get_rcut() (deepmd.tf.model.frozen.FrozenModel method), 857
get_rcut() (deepmd.tf.model.FrozenModel method), 882
get_rcut() (deepmd.tf.model.linear.LinearModel method), 859
get_rcut() (deepmd.tf.model.model.Model method), 865
get_rcut() (deepmd.tf.model.model.StandardModel method), 867
get_rcut() (deepmd.tf.model.multi.MultiModel method), 870
get_rcut() (deepmd.tf.model.MultiModel method), 894
get_rcut() (deepmd.tf.model.pairtab.PairTabModel method), 873
get_rcut() (deepmd.tf.model.PairTabModel method), 885
get_rcut() (deepmd.tf.model.pairwise_dprc.PairwiseDPRc

method), 875
get_rcut() (deepmd.tf.model.PairwiseDPRc method), 887
get_rcut() (deepmd.tf.model.tensor.TensorModel method), 878
get_rcut() (deepmd.utils.update_sel.BaseUpdateSel method), 1057
get_reduce_name() (in module deepmd.dpmodel), 382
get_reduce_name() (in module deepmd.dpmodel.output_def), 378
get_resource() (in module deepmd.tf.cluster), 640
get_resource() (in module deepmd.tf.cluster.local), 639
get_rot_mat() (deepmd.tf.descriptor.DescrptLocFram method), 712
get_rot_mat() (deepmd.tf.descriptor.DescrptSeA method), 718
get_rot_mat() (deepmd.tf.descriptor.DescrptSeAEf method), 723
get_rot_mat() (deepmd.tf.descriptor.loc_frame.DescrptLocFrame method), 655
get_rot_mat() (deepmd.tf.descriptor.se_a.DescrptSeA method), 663
get_rot_mat() (deepmd.tf.descriptor.se_a_ef.DescrptSeAEf method), 669
get_s_range() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
get_sel() (deepmd.dpmodel.atomic_model.dp_atomic_model.DPA method), 263
get_sel() (deepmd.dpmodel.atomic_model.DPAtomicModel method), 273
get_sel() (deepmd.dpmodel.atomic_model.linear_atomic_model.Li method), 265
get_sel() (deepmd.dpmodel.atomic_model.LinearEnergyAtomicMo method), 276
get_sel() (deepmd.dpmodel.atomic_model.pairtab_atomic_model.P method), 269
get_sel() (deepmd.dpmodel.atomic_model.PairTabAtomicModel method), 278
get_sel() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 301
get_sel() (deepmd.dpmodel.descriptor.DescrptHybrid method), 302
get_sel() (deepmd.dpmodel.descriptor.DescrptSeA method), 306
get_sel() (deepmd.dpmodel.descriptor.DescrptSeR method), 308
get_sel() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 method), 285
get_sel() (deepmd.dpmodel.descriptor.hybrid.DescrptHybrid method), 288
get_sel() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA method), 293
get_sel() (deepmd.dpmodel.descriptor.se_r.DescrptSeR method), 296
get_sel() (deepmd.pt.model.atomic_model.dp_atomic_model.DPA method), 440
get_sel() (deepmd.pt.model.atomic_model.DPAtomicModel method), 455
get_sel() (deepmd.pt.model.atomic_model.linear_atomic_model.Li method), 443
get_sel() (deepmd.pt.model.atomic_model.LinearEnergyAtomicMo method), 458
get_sel() (deepmd.pt.model.atomic_model.pairtab_atomic_model.P method), 447
get_sel() (deepmd.pt.model.atomic_model.PairTabAtomicModel method), 461
get_sel() (deepmd.pt.model.descriptor.descriptor.DescriptorBlock method), 469
get_sel() (deepmd.pt.model.descriptor.DescriptorBlock method), 506
get_sel() (deepmd.pt.model.descriptor.DescrptBlockRepformers method), 521
get_sel() (deepmd.pt.model.descriptor.DescrptBlockSeA method), 523
get_sel() (deepmd.pt.model.descriptor.DescrptBlockSeAtten method), 507
get_sel() (deepmd.pt.model.descriptor.DescrptDPA1 method), 512
get_sel() (deepmd.pt.model.descriptor.DescrptDPA2 method), 515
get_sel() (deepmd.pt.model.descriptor.DescrptHybrid method), 519
get_sel() (deepmd.pt.model.descriptor.DescrptSeA method), 524
get_sel() (deepmd.pt.model.descriptor.DescrptSeR method), 526
get_sel() (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 method), 473
get_sel() (deepmd.pt.model.descriptor.dpa2.DescrptDPA2 method), 477
get_sel() (deepmd.pt.model.descriptor.hybrid.DescrptHybrid method), 482
get_sel() (deepmd.pt.model.descriptor.repformers.DescrptBlockRep method), 491
get_sel() (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA method), 495
get_sel() (deepmd.pt.model.descriptor.se_a.DescrptSeA method), 492
get_sel() (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtte method), 497
get_sel() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 503
get_sel() (deepmd.pt.model.model.frozen.FrozenModel method), 533
get_sel() (deepmd.pt.model.model.FrozenModel method), 543

get_sel() (deepmd.utils.update_sel.BaseUpdateSel method), 1057
get_sel_type() (deepmd.dpmodel.atomic_model.dp_atomic_model.DPAtomicModel method), 264
get_sel_type() (deepmd.dpmodel.atomic_model.DPAtomicModel method), 274
get_sel_type() (deepmd.dpmodel.atomic_model.linear_atomic_model.LinearAtomicModel method), 266
get_sel_type() (deepmd.dpmodel.atomic_model.LinearEnergyAtomicModel method), 277
get_sel_type() (deepmd.dpmodel.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 271
get_sel_type() (deepmd.dpmodel.atomic_model.PairTabAtomicModel method), 280
get_sel_type() (deepmd.dpmodel.fitting.general_fitting.GeneralFitting method), 319
get_sel_type() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 339
get_sel_type() (deepmd.dpmodel.model.spin_model.SpinModel method), 345
get_sel_type() (deepmd.dpmodel.model.SpinModel method), 349
get_sel_type() (deepmd.infer.deep_eval.DeepEval method), 394
get_sel_type() (deepmd.infer.deep_eval.DeepEvalBackend method), 392
get_sel_type() (deepmd.infer.DeepEval method), 408
get_sel_type() (deepmd.pt.infer.deep_eval.DeepEval method), 417
get_sel_type() (deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel method), 441
get_sel_type() (deepmd.pt.model.atomic_model.DPAtomicModel method), 457
get_sel_type() (deepmd.pt.model.atomic_model.linear_atomic_model.LinearAtomicModel method), 444
get_sel_type() (deepmd.pt.model.atomic_model.LinearEnergyAtomicModel method), 460
get_sel_type() (deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 450
get_sel_type() (deepmd.pt.model.atomic_model.PairTabAtomicModel method), 464
get_sel_type() (deepmd.pt.model.model.frozen.FrozenModel method), 533
get_sel_type() (deepmd.pt.model.model.FrozenModel method), 543
get_sel_type() (deepmd.pt.model.model.spin_model.SpinModel method), 538
get_sel_type() (deepmd.pt.model.model.SpinModel method), 547
get_sel_type() (deepmd.pt.model.task.fitting.GeneralFitting method), 590
get_sel_type() (deepmd.tf.DeepEval method), 1009
get_sel_type() (deepmd.tf.fit.dipole.DipoleFittingSeA method), 761
get_sel_type() (deepmd.tf.fit.DipoleFittingSeA method), 779
get_sel_type() (deepmd.tf.fit.GlobalPolarFittingSeA method), 790
get_sel_type() (deepmd.tf.fit.PolarFittingSeA method), 776
get_sel_type() (deepmd.tf.fit.polar.PolarFittingSeA method), 774
get_sel_type() (deepmd.tf.fit.polar.GlobalPolarFittingSeA method), 792
get_sel_type() (deepmd.tf.infer.deep_eval.DeepEval method), 802
get_sel_type() (deepmd.tf.infer.deep_tensor.DeepTensor method), 812
get_sel_type() (deepmd.tf.infer.DeepEval method), 827
get_sel_type() (deepmd.tf.model.tensor.TensorModel method), 878
get_sess() (in module deepmd.tf.nvnmd.utils.network), 917
get_spin_mask() (deepmd.utils.spin.Spin method), 1055
get_spin_model() (in module deepmd.dpmodel.model.model), 344
get_spin_norm() (deepmd.tf.utils.spin.Spin method), 979
get_standard_model() (in module deepmd.dpmodel.model.model), 344
get_stat() (deepmd.utils.neighbor_stat.NeighborStat method), 1041
get_stats() (deepmd.pt.model.descriptor.descriptor.DescriptorBlock method), 469
get_stats() (deepmd.pt.model.descriptor.DescriptorBlock method), 506
get_stats() (deepmd.pt.model.descriptor.DescrptBlockRepformers method), 522
get_stats() (deepmd.pt.model.descriptor.DescrptBlockSeA method), 523
get_stats() (deepmd.pt.model.descriptor.DescrptBlockSeAtten method), 508
get_stats() (deepmd.pt.model.descriptor.DescrptSeR method), 527
get_stats() (deepmd.pt.model.descriptor.repformers.DescrptBlockR method), 492
get_stats() (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA method), 495
get_stats() (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeA method), 497
get_stats() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 503
get_std() (deepmd.utils.env_mat_stat.EnvMatStat method), 1038

get_sys() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 950
get_sys() (deepmd.tf.utils.DeepmdDataSystem method), 991
get_sys() (deepmd.utils.data_system.DeepmdDataSystem method), 1033
get_sys_ntest() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 950
get_sys_ntest() (deepmd.tf.utils.DeepmdDataSystem method), 990
get_sys_ntest() (deepmd.utils.data_system.DeepmdDataSystem method), 1032
get_sys_numb_batch() (deepmd.tf.utils.data.DeepmdData method), 946
get_sys_numb_batch() (deepmd.tf.utils.DeepmdData method), 987
get_sys_numb_batch() (deepmd.utils.data.DeepmdData method), 1027
get_tensor_by_name() (in module deepmd.tf.utils.graph), 953
get_tensor_by_name_from_graph() (in module deepmd.tf.utils.graph), 953
get_tensor_names() (deepmd.tf.descriptor.Descriptor method), 703
get_tensor_names() (deepmd.tf.descriptor.descriptor.Descriptor method), 645
get_tensor_names() (deepmd.tf.descriptor.DescrptHybrid method), 709
get_tensor_names() (deepmd.tf.descriptor.hybrid.DescrptHybrid method), 651
get_tensor_names() (deepmd.tf.descriptor.se.DescrptSe method), 657
get_test() (deepmd.tf.utils.data.DeepmdData method), 945
get_test() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 949
get_test() (deepmd.tf.utils.DeepmdData method), 987
get_test() (deepmd.tf.utils.DeepmdDataSystem method), 990
get_test() (deepmd.utils.data.DeepmdData method), 1027
get_test() (deepmd.utils.data_system.DeepmdDataSystem method), 1032
get_trainer() (in module deepmd.pt.entrypoints.main), 415
get_training_jdata() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 913
get_two_side_type_embedding() (in module deepmd.tf.utils.compress), 940
get_type_embedding() (in module deepmd.tf.utils.compress), 940
get_type_embedding_net_nodes_from_graph_def() (in module deepmd.tf.utils.graph), 957
get_type_embedding_net_variables_from_graph_def() (in module deepmd.tf.utils.graph), 957
get_type_embedding_weight() (in module deepmd.tf.nvnmd.utils.weight), 919
get_type_map() (deepmd.dpmodel.atomic_model.base_atomic_model method), 261
get_type_map() (deepmd.dpmodel.atomic_model.BaseAtomicModel method), 272
get_type_map() (deepmd.dpmodel.atomic_model.linear_atomic_model method), 265
get_type_map() (deepmd.dpmodel.atomic_model.LinearEnergyAtom method), 276
get_type_map() (deepmd.dpmodel.atomic_model.pairtab_atomic_model method), 269
get_type_map() (deepmd.dpmodel.atomic_model.PairTabAtomicModel method), 278
get_type_map() (deepmd.dpmodel.infer.deep_eval.DeepEval method), 339
get_type_map() (deepmd.dpmodel.model.spin_model.SpinModel method), 345
get_type_map() (deepmd.dpmodel.model.SpinModel method), 349
get_type_map() (deepmd.infer.deep_eval.DeepEval method), 393
get_type_map() (deepmd.infer.deep_eval.DeepEvalBackend method), 391
get_type_map() (deepmd.infer.DeepEval method), 417
get_type_map() (deepmd.pt.infer.deep_eval.DeepEval method), 417
get_type_map() (deepmd.pt.model.atomic_model.base_atomic_model method), 435
get_type_map() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 452
get_type_map() (deepmd.pt.model.atomic_model.linear_atomic_model method), 443
get_type_map() (deepmd.pt.model.atomic_model.LinearEnergyAtom method), 458
get_type_map() (deepmd.pt.model.atomic_model.pairtab_atomic_model method), 447
get_type_map() (deepmd.pt.model.atomic_model.PairTabAtomicModel method), 461
get_type_map() (deepmd.pt.model.model.frozen.FrozenModel method), 533
get_type_map() (deepmd.pt.model.model.FrozenModel method), 543
get_type_map() (deepmd.pt.model.model.spin_model.SpinModel method), 538
get_type_map() (deepmd.pt.model.model.SpinModel method), 547

get_type_map() (deepmd.tf.DeepEval method), 1009
get_type_map() (deepmd.tf.infer.deep_eval.DeepEval method), 802
get_type_map() (deepmd.tf.infer.deep_tensor.DeepTensor method), 812
get_type_map() (deepmd.tf.infer.DeepEval method), 827
get_type_map() (deepmd.tf.model.dos.DOSModel method), 851
get_type_map() (deepmd.tf.model.DOSModel method), 889
get_type_map() (deepmd.tf.model.ener.EnerModel method), 854
get_type_map() (deepmd.tf.model.EnerModel method), 891
get_type_map() (deepmd.tf.model.frozen.FrozenModel method), 857
get_type_map() (deepmd.tf.model.FrozenModel method), 882
get_type_map() (deepmd.tf.model.linear.LinearModel method), 859
get_type_map() (deepmd.tf.model.model.Model method), 861
get_type_map() (deepmd.tf.model.multi.MultiModel method), 870
get_type_map() (deepmd.tf.model.MultiModel method), 894
get_type_map() (deepmd.tf.model.tensor.TensorModel method), 878
get_type_map() (deepmd.tf.utils.data.DeepmdData method), 945
get_type_map() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 950
get_type_map() (deepmd.tf.utils.DeepmdData method), 987
get_type_map() (deepmd.tf.utils.DeepmdDataSystem method), 990
get_type_map() (deepmd.utils.data.DeepmdData method), 1027
get_type_map() (deepmd.utils.data_system.DeepmdDataSystem method), 1032
get_type_map() (deepmd.utils.update_sel.BaseUpdateSel method), 1057
get_type_mask() (deepmd.dpmodel.utils.AtomExcludeMask method), 368
get_type_mask() (deepmd.dpmodel.utils.exclude_mask.AtomExcludeMask method), 353
get_type_mask() (deepmd.pt.utils.AtomExcludeMask method), 638
get_type_mask() (deepmd.pt.utils.exclude_mask.AtomExcludeMask method), 620
get_type_weight() (in module deepmd.tf.nvnmd.utils.weight), 920
get_use_spin() (deepmd.tf.utils.spin.Spin method), 979
get_use_spin() (deepmd.utils.spin.Spin method), 1054
get_virtual_len() (deepmd.tf.utils.spin.Spin method), 979
get_virtual_scale() (deepmd.utils.spin.Spin method), 1054
get_virtual_scale_mask() (deepmd.utils.spin.Spin method), 1055
get_weight() (in module deepmd.tf.nvnmd.utils.weight), 919
get_weighted_sampler() (in module deepmd.pt.utils.dataloader), 616
glob() (deepmd.tf.utils.path.DPH5Path method), 970
glob() (deepmd.tf.utils.path.DPOSPath method), 971
glob() (deepmd.tf.utils.path.DPPath method), 973
glob() (deepmd.utils.path.DPH5Path method), 1049
glob() (deepmd.utils.path.DPOSPath method), 1047
glob() (deepmd.utils.path.DPPath method), 1046
GLOBAL_CONFIG (in module deepmd.env), 1063
GLOBAL_CONFIG (in module deepmd.tf.env), 1002
global_cvt_2_ener_float() (in module deepmd.tf.env), 1003
global_cvt_2_tf_float() (in module deepmd.tf.env), 1003
GLOBAL_NP_FLOAT_PRECISION (in module deepmd.dpmodel.common), 374
GLOBAL_NP_FLOAT_PRECISION (in module deepmd.env), 1063
GLOBAL_NP_FLOAT_PRECISION (in module deepmd.pt.utils.env), 618
GLOBAL_NP_FLOAT_PRECISION (in module deepmd.tf.env), 1002
GLOBAL_PT_ENER_FLOAT_PRECISION (in module deepmd.pt.utils.env), 618
GLOBAL_PT_FLOAT_PRECISION (in module deepmd.pt.utils.env), 618
GLOBAL_TF_FLOAT_PRECISION (in module deepmd.tf.env), 1003
GLOBAL_TF_FLOAT_PRECISION (in module deepmd.tf.nvnmd.fit.ener), 910
GlobalPolarFittingSeA (class in deepmd.tf.fit), 789
GlobalPolarFittingSeA (class in deepmd.tf.fit.polar), 776
GlobalPolarModel (class in deepmd.tf.model), 896
GlobalPolarModel (class in deepmd.tf.model.tensor), 880
GPU_MAX_NBOR_SIZE (C macro), 1320, 1321
gpuDeviceSynchronize (C macro), 1321
gpuGetLastError (C macro), 1321, 1322
gpuMemcpy (C macro), 1322

gpuMemcpyDeviceToDevice (C macro), 1322
gpuMemcpyDeviceToHost (C macro), 1323
gpuMemcpyHostToDevice (C macro), 1323
gpuMemset (C macro), 1324
gpus (deepmd.tf.train.run_options.RunOptions attribute), 933
gradient_max_norm: training/gradient_max_norm (Argument), 159
GraphTooLargeError, 950
GraphWithoutTensorError, 950

H

has_efield (deepmd.infer.deep_eval.DeepEval property), 393
has_efield (deepmd.infer.DeepEval property), 406
has_spin (deepmd.infer.deep_eval.DeepEval property), 393
has_spin (deepmd.infer.DeepEval property), 406
has_spin() (deepmd.dpmodel.model.spin_model.SpinModel static method), 345
has_spin() (deepmd.dpmodel.model.SpinModel static method), 349
has_spin() (deepmd.pt.model.model.spin_model.SpinModel method), 539
has_spin() (deepmd.pt.model.model.SpinModel method), 548
hex2bin() (deepmd.tf.nvnmd.utils.Encode method), 922
hex2bin() (deepmd.tf.nvnmd.utils.encode.Encode method), 914
hex2bin_str() (deepmd.tf.nvnmd.utils.Encode method), 922
hex2bin_str() (deepmd.tf.nvnmd.utils.encode.Encode method), 914
hook() (deepmd.dpmodel.utils.update_sel.UpdateSel method), 366
hook() (deepmd.pt.utils.update_sel.UpdateSel method), 636
hook() (deepmd.tf.utils.update_sel.UpdateSel method), 984
hook() (deepmd.utils.update_sel.BaseUpdateSel method), 1057

I

Identity (class in deepmd.pt.model.network.network), 554
implemented_properties (deepmd.calculator.DP attribute), 1059
implemented_properties (deepmd.pt.utils.ase_calc.DPCalculator attribute), 612
implemented_properties (deepmd.tf.calculator.DP attribute), 995
import_ops() (in module deepmd.tf.op), 932
init_atom_exclude_types_placeholder() (deepmd.utils.spin.Spin method), 1055
init_atom_exclude_types_placeholder_spin() (deepmd.utils.spin.Spin method), 1054
init_config_by_version() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
init_ctrl() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
init_dpin() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
init_dscp() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
init_fitn() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
init_from_config() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
init_from_deepmd_input() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
init_from_jdata() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 911
init_nbit() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
init_net_size() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
init_out_stat() (deepmd.dpmodel.atomic_model.base_atomic_mod method), 261
init_out_stat() (deepmd.dpmodel.atomic_model.BaseAtomicMode method), 272
init_out_stat() (deepmd.pt.model.atomic_model.base_atomic_mod method), 435
init_out_stat() (deepmd.pt.model.atomic_model.BaseAtomicMode method), 451
init_pair_exclude_types_placeholder() (deepmd.utils.spin.Spin method), 1054
init_size() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
init_train_mode() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 911
init_value() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 911
init_variables() (deepmd.tf.descriptor.Descriptor method), 703
init_variables() (deepmd.tf.descriptor.descriptor.Descriptor method), 645
init_variables() (deepmd.tf.descriptor.DescrptHybrid method), 709
init_variables() (deepmd.tf.descriptor.DescrptLocFrame method), 713
init_variables() (deepmd.tf.descriptor.DescrptSeA method), 719
init_variables() (deepmd.tf.descriptor.DescrptSeAtten method), 733

init_variables() (deepmd.tf.descriptor.hybrid.DescrptHybrid method), 651
init_variables() (deepmd.tf.descriptor.loc_frame.DescrptLocFrame method), 655
init_variables() (deepmd.tf.descriptor.se.DescrptSe method), 657
init_variables() (deepmd.tf.descriptor.se_a.DescrptSeA method), 664
init_variables() (deepmd.tf.descriptor.se_atten.DescrptDPA1Compat method), 687
init_variables() (deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 681
init_variables() (deepmd.tf.fit.dipole.DipoleFittingSeA method), 761
init_variables() (deepmd.tf.fit.DipoleFittingSeA method), 779
init_variables() (deepmd.tf.fit.dos.DOSFitting method), 765
init_variables() (deepmd.tf.fit.DOSFitting method), 782
init_variables() (deepmd.tf.fit.ener.EnerFitting method), 769
init_variables() (deepmd.tf.fit.EnerFitting method), 786
init_variables() (deepmd.tf.fit.Fitting method), 787
init_variables() (deepmd.tf.fit.fitting.Fitting method), 771
init_variables() (deepmd.tf.fit.GlobalPolarFittingSeA method), 790
init_variables() (deepmd.tf.fit.polar.GlobalPolarFittingSeA method), 777
init_variables() (deepmd.tf.fit.polar.PolarFittingSeA method), 775
init_variables() (deepmd.tf.fit.PolarFittingSeA method), 793
init_variables() (deepmd.tf.model.dos.DOSModel method), 852
init_variables() (deepmd.tf.model.DOSModel method), 890
init_variables() (deepmd.tf.model.ener.EnerModel method), 855
init_variables() (deepmd.tf.model.EnerModel method), 892
init_variables() (deepmd.tf.model.frozen.FrozenModel method), 857
init_variables() (deepmd.tf.model.FrozenModel method), 882
init_variables() (deepmd.tf.model.linear.LinearModel method), 859
init_variables() (deepmd.tf.model.model.Model method), 862
init_variables() (deepmd.tf.model.multi.MultiModel method), 870
init_variables() (deepmd.tf.model.MultiModel method), 895
init_variables() (deepmd.tf.model.pairtab.PairTabModel method), 873
init_variables() (deepmd.tf.model.PairTabModel method), 885
init_variables() (deepmd.tf.model.pairwise_dprc.PairwiseDPRc method), 875
init_variables() (deepmd.tf.model.PairwiseDPRc method), 887
init_variables() (deepmd.tf.model.tensor.TensorModel method), 878
init_variables() (deepmd.tf.utils.type_embed.TypeEmbedNet method), 983
int_64 (C++ type), 1326
inter2phys() (deepmd.pt.utils.preprocess.Region3D method), 630
inter2phys() (in module deepmd.dpmodel.utils), 372
inter2phys() (in module deepmd.dpmodel.utils.region), 362
inter2phys() (in module deepmd.pt.utils.region), 632
InvarFitting (class in deepmd.dpmodel.fitting), 333
InvarFitting (class in deepmd.dpmodel.fitting.invar_fitting), 320
InvarFitting (class in deepmd.pt.model.task.invar_fitting), 591
io (deepmd.backend.backend.Backend.Feature attribute), 254
is_aparam_nall() (deepmd.dpmodel.atomic_model.dp_atomic_mod method), 264
is_aparam_nall() (deepmd.dpmodel.atomic_model.DPAtomicModel method), 274
is_aparam_nall() (deepmd.dpmodel.atomic_model.linear_atomic_m method), 266
is_aparam_nall() (deepmd.dpmodel.atomic_model.LinearEnergyAt method), 277
is_aparam_nall() (deepmd.dpmodel.atomic_model.pairtab_atomic_ method), 271
is_aparam_nall() (deepmd.dpmodel.atomic_model.PairTabAtomicM method), 280
is_aparam_nall() (deepmd.dpmodel.model.spin_model.SpinModel method), 345
is_aparam_nall() (deepmd.dpmodel.model.SpinModel method), 349
is_aparam_nall() (deepmd.pt.model.atomic_model.dp_atomic_mod method), 441
is_aparam_nall() (deepmd.pt.model.atomic_model.DPAtomicModel method), 457
is_aparam_nall() (deepmd.pt.model.atomic_model.linear_atomic_m method), 444

is_aparam_nall() (deepmd.pt.model.atomic_model.LinearEnergyAtomicModel method), 460
is_aparam_nall() (deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 450
is_aparam_nall() (deepmd.pt.model.atomic_model.PairTabAtomicModel method), 464
is_aparam_nall() (deepmd.pt.model.model.frozen.FrozenModel method), 533
is_aparam_nall() (deepmd.pt.model.model.FrozenModel method), 543
is_aparam_nall() (deepmd.pt.model.model.spin_model.SpinModel method), 538
is_aparam_nall() (deepmd.pt.model.model.SpinModel method), 547
is_available() (deepmd.backend.backend.Backend method), 255
is_available() (deepmd.backend.dpmodel.DPModelBackend method), 257
is_available() (deepmd.backend.pytorch.PyTorchBackend method), 258
is_available() (deepmd.backend.tensorflow.TensorFlowBackend method), 260
is_built_with_cuda() (deepmd.pt.entrypoints.main.SummaryPrinter method), 415
is_built_with_cuda() (deepmd.utils.summary.SummaryPrinter method), 1056
is_built_with_rocm() (deepmd.pt.entrypoints.main.SummaryPrinter method), 415
is_built_with_rocm() (deepmd.utils.summary.SummaryPrinter method), 1056
is_chief (deepmd.tf.train.run_options.RunOptions property), 933
is_dir() (deepmd.tf.utils.path.DPH5Path method), 970
is_dir() (deepmd.tf.utils.path.DPOSPath method), 972
is_dir() (deepmd.tf.utils.path.DPPath method), 973
is_dir() (deepmd.utils.path.DPH5Path method), 1050
is_dir() (deepmd.utils.path.DPOSPath method), 1048
is_dir() (deepmd.utils.path.DPPath method), 1046
is_file() (deepmd.tf.nvnmd.utils.fio.Fio method), 915
is_file() (deepmd.tf.utils.path.DPH5Path method), 970
is_file() (deepmd.tf.utils.path.DPOSPath method), 972
is_file() (deepmd.tf.utils.path.DPPath method), 973
is_file() (deepmd.utils.path.DPH5Path method), 1050
is_file() (deepmd.utils.path.DPOSPath method), 1048
is_file() (deepmd.utils.path.DPPath method), 1046
is_gpu_available() (deepmd.dpmodel.utils.batch_size.AutoBatchSize method), 351
is_gpu_available() (deepmd.pt.utils.auto_batch_size.AutoBatchSize method), 613
is_gpu_available() (deepmd.tf.utils.batch_size.AutoBatchSize method), 938
is_gpu_available() (deepmd.utils.batch_size.AutoBatchSize method), 1021
is_oom_error() (deepmd.dpmodel.utils.batch_size.AutoBatchSize method), 351
is_oom_error() (deepmd.pt.utils.auto_batch_size.AutoBatchSize method), 613
is_oom_error() (deepmd.tf.utils.batch_size.AutoBatchSize method), 938
is_oom_error() (deepmd.utils.batch_size.AutoBatchSize method), 1021
is_path() (deepmd.tf.nvnmd.utils.fio.Fio method), 915
iter() (deepmd.pt.utils.env_mat_stat.EnvMatStatSe method), 619
iter() (deepmd.utils.env_mat_stat.EnvMatStat method), 1037
iterator() (deepmd.dpmodel.utils.neighbor_stat.NeighborStat method), 355
iterator() (deepmd.pt.utils.neighbor_stat.NeighborStat method), 625
iterator() (deepmd.tf.utils.neighbor_stat.NeighborStat method), 961
iterator() (deepmd.utils.neighbor_stat.NeighborStat method), 1041

J

j_loader() (in module deepmd.common), 1062
j_loader() (in module deepmd.tf.common), 998
j_must_have() (in module deepmd.common), 1061
j_must_have() (in module deepmd.tf.common), 998
jdata_cmd_freeze (in module deepmd.tf.nvnmd.entrypoints.train), 904
jdata_cmd_train (in module deepmd.tf.nvnmd.entrypoints.train), 904
jdata_config_v0 (in module deepmd.tf.nvnmd.data.data), 898
jdata_config_v0_ni128 (in module deepmd.tf.nvnmd.data.data), 898

jdata_config_v0_ni256 (in module deepmd.tf.nvnmd.data.data), 898
jdata_config_v1 (in module deepmd.tf.nvnmd.data.data), 898
jdata_config_v1_ni128 (in module deepmd.tf.nvnmd.data.data), 898
jdata_config_v1_ni256 (in module deepmd.tf.nvnmd.data.data), 898
jdata_deepmd_input_v0 (in module deepmd.tf.nvnmd.data.data), 898
jdata_deepmd_input_v0_ni128 (in module deepmd.tf.nvnmd.data.data), 898
jdata_deepmd_input_v0_ni256 (in module deepmd.tf.nvnmd.data.data), 898
jdata_deepmd_input_v1 (in module deepmd.tf.nvnmd.data.data), 898
jdata_deepmd_input_v1_ni128 (in module deepmd.tf.nvnmd.data.data), 898
jdata_deepmd_input_v1_ni256 (in module deepmd.tf.nvnmd.data.data), 898
jdata_sys (in module deepmd.tf.nvnmd.data.data), 898
JIT (in module deepmd.pt.utils.env), 618

K

keys() (deepmd.dpmodel.FittingOutputDef method), 381
keys() (deepmd.dpmodel.ModelOutputDef method), 381
keys() (deepmd.dpmodel.output_def.FittingOutputDef method), 378
keys() (deepmd.dpmodel.output_def.ModelOutputDef method), 378
keys_derv_c() (deepmd.dpmodel.ModelOutputDef method), 381
keys_derv_c() (deepmd.dpmodel.output_def.ModelOutputDef method), 378
keys_derv_c_redu() (deepmd.dpmodel.ModelOutputDef method), 381
keys_derv_c_redu() (deepmd.dpmodel.output_def.ModelOutputDef method), 378
keys_derv_r() (deepmd.dpmodel.ModelOutputDef method), 381
keys_derv_r() (deepmd.dpmodel.output_def.ModelOutputDef method), 378
keys_hess_r() (deepmd.dpmodel.ModelOutputDef method), 381
keys_hess_r() (deepmd.dpmodel.output_def.ModelOutputDef method), 378
keys_outp() (deepmd.dpmodel.ModelOutputDef method), 381
keys_outp() (deepmd.dpmodel.output_def.ModelOutputDef method), 378
keys_redu() (deepmd.dpmodel.ModelOutputDef method), 381
keys_redu() (deepmd.dpmodel.output_def.ModelOutputDef method), 378
kf_blocksize: training[LKF]/kf_blocksize (Argument), 159
KFOptimizerWrapper (class in deepmd.pt.optimizer), 607
KFOptimizerWrapper (class in deepmd.pt.optimizer.KFWrapper), 606

L

label() (deepmd.driver.DPDriver method), 1063
label_requirement (deepmd.pt.loss.dos.DOSLoss property), 421
label_requirement (deepmd.pt.loss.DOSLoss property), 429
label_requirement (deepmd.pt.loss.ener.EnergyStdLoss property), 422
label_requirement (deepmd.pt.loss.ener_spin.EnergySpinLoss property), 424
label_requirement (deepmd.pt.loss.EnergySpinLoss property), 431
label_requirement (deepmd.pt.loss.EnergyStdLoss property), 430
label_requirement (deepmd.pt.loss.loss.TaskLoss property), 425
label_requirement (deepmd.pt.loss.TaskLoss property), 432
label_requirement (deepmd.pt.loss.tensor.TensorLoss property), 426
label_requirement (deepmd.pt.loss.TensorLoss property), 433
layer_name: model[standard]/fitting_net[ener]/layer_name (Argument), 141
layer_norm_numpy() (deepmd.dpmodel.utils.network.LayerNorm static method), 358
layer_norm_tf() (in module deepmd.tf.utils.network), 963
LayerNorm (class in deepmd.dpmodel.utils.network), 357
LayerNorm (class in deepmd.pt.model.network.layernorm), 549
layernorm() (in module deepmd.tf.utils.network), 963
learning_rate (Argument)
learning_rate:, 147
learning_rate/scale_by_worker (Argument)
scale_by_worker:, 147
learning_rate/type (Argument)
type:, 147
learning_rate: learning_rate (Argument), 147

learning_rate_args() (in module deepmd.utils.argcheck), 1018
learning_rate_dict (Argument)
    learning_rate_dict:, 148
learning_rate_dict:
    learning_rate_dict (Argument), 148
learning_rate_dict_args() (in module deepmd.utils.argcheck), 1018
learning_rate_exp() (in module deepmd.utils.argcheck), 1018
learning_rate_variant_type_args() (in module deepmd.utils.argcheck), 1018
learning_rate[exp]/decay_rate (Argument)
    decay_rate:, 147
learning_rate[exp]/decay_steps (Argument)
    decay_steps:, 147
learning_rate[exp]/start_lr (Argument)
    start_lr:, 147
learning_rate[exp]/stop_lr (Argument)
    stop_lr:, 147
LearningRateExp (class in deepmd.pt.utils.learning_rate), 622
LearningRateExp (class in deepmd.tf.utils), 991
LearningRateExp (class in deepmd.tf.utils.learning_rate), 958
lett_to_ln (in module deepmd.utils.econf_embd), 1035
lib_env (in module deepmd.tf.lmp), 1004
libpython (in module deepmd.tf.lmp), 1005
limit_pref() (in module deepmd.utils.argcheck), 1018
limit_pref_acdf:
    loss[dos]/limit_pref_acdf (Argument), 153
limit_pref_ados:
    loss[dos]/limit_pref_ados (Argument), 153
limit_pref_ae:
    loss[ener_spin]/limit_pref_ae (Argument), 152
    loss[ener]/limit_pref_ae (Argument), 149
limit_pref_cdf:
    loss[dos]/limit_pref_cdf (Argument), 153
limit_pref_dos:
    loss[dos]/limit_pref_dos (Argument), 152
limit_pref_e:
    loss[ener_spin]/limit_pref_e (Argument), 150
    loss[ener]/limit_pref_e (Argument), 148
limit_pref_f:
    loss[ener]/limit_pref_f (Argument), 149
limit_pref_fm:
    loss[ener_spin]/limit_pref_fm (Argument), 151
limit_pref_fr:
    loss[ener_spin]/limit_pref_fr (Argument), 151
limit_pref_gf:
    loss[ener]/limit_pref_gf (Argument), 150
limit_pref_pf:
    loss[ener_spin]/limit_pref_pf (Argument), 152
    loss[ener]/limit_pref_pf (Argument), 149
limit_pref_v:
    loss[ener_spin]/limit_pref_v (Argument), 151
    loss[ener]/limit_pref_v (Argument), 149
Linear (class in deepmd.pt.model.network.network), 557
linear_ener_model_args() (in module deepmd.utils.argcheck), 1018
LinearEnergyAtomicModel (class in deepmd.dpmodel.atomic_model), 275
LinearEnergyAtomicModel (class in deepmd.dpmodel.atomic_model.linear_atomic_model), 264
LinearEnergyAtomicModel (class in deepmd.pt.model.atomic_model), 458
LinearEnergyAtomicModel (class in deepmd.pt.model.atomic_model.linear_atomic_model), 442
LinearEnergyModel (class in deepmd.tf.model), 883
LinearEnergyModel (class in deepmd.tf.model.linear), 859
LinearModel (class in deepmd.tf.model.linear), 858
list:
    model[standard]/descriptor[hybrid]/list (Argument), 125
list_to_doc() (in module deepmd.tf.utils.argcheck), 937
list_to_doc() (in module deepmd.utils.argcheck), 1016
list_update() (deepmd.pt.model.descriptor.repformer_layer.Repfor method), 489
list_update_res_avg() (deepmd.pt.model.descriptor.repformer_layer.RepformerLa method), 489
list_update_res_incr() (deepmd.pt.model.descriptor.repformer_layer.RepformerLa method), 489
LKFOptimizer (class in deepmd.pt.optimizer), 608
LKFOptimizer (class in deepmd.pt.optimizer.LKF), 606
ln_eps:
    model[standard]/descriptor[se_atten]/ln_eps (Argument), 128
ln_to_lett (in module deepmd.utils.econf_embd), 1035
load() (deepmd.tf.nvnmd.utils.fio.FioBin method),

916
load() (deepmd.tf.nvnmd.utils.fio.FioDic method), 915
load() (deepmd.tf.nvnmd.utils.fio.FioJsonDic method), 916
load() (deepmd.tf.nvnmd.utils.fio.FioNpyDic method), 916
load() (deepmd.tf.nvnmd.utils.fio.FioTxt method), 916
load() (deepmd.tf.nvnmd.utils.FioBin method), 922
load() (deepmd.tf.nvnmd.utils.FioDic method), 922
load() (deepmd.tf.nvnmd.utils.FioTxt method), 922
load_dp_model() (in module deepmd.dpmodel.utils), 373
load_dp_model() (in module deepmd.dpmodel.utils.serialization), 364
load_graph_def() (in module deepmd.tf.utils.graph), 952
load_numpy() (deepmd.tf.utils.path.DPH5Path method), 969
load_numpy() (deepmd.tf.utils.path.DPOSPath method), 971
load_numpy() (deepmd.tf.utils.path.DPPath method), 972
load_numpy() (deepmd.utils.path.DPH5Path method), 1049
load_numpy() (deepmd.utils.path.DPOSPath method), 1047
load_numpy() (deepmd.utils.path.DPPath method), 1045
load_or_compute_stats() (deepmd.utils.env_mat_stat.EnvMatStat method), 1037
load_prefix (deepmd.tf.infer.deep_eval.DeepEvalOld attribute), 805
load_stats() (deepmd.utils.env_mat_stat.EnvMatStat method), 1037
load_txt() (deepmd.tf.utils.path.DPH5Path method), 970
load_txt() (deepmd.tf.utils.path.DPOSPath method), 971
load_txt() (deepmd.tf.utils.path.DPPath method), 972
load_txt() (deepmd.utils.path.DPH5Path method), 1049
load_txt() (deepmd.utils.path.DPOSPath method), 1047
load_txt() (deepmd.utils.path.DPPath method), 1045
local_cluster (deepmd.pt.model.descriptor.descriptor.DescriptorBlock attribute), 468
local_cluster (deepmd.pt.model.descriptor.Descriptor attribute), 506
LOCAL_RANK (in module deepmd.pt.utils.env), 618
LocalAtten (class in deepmd.pt.model.descriptor.repformer_layer), 486
LocalSelfMultiheadAttention (class in deepmd.pt.model.network.network), 569
log (in module deepmd.entrypoints.neighbor_stat), 385
log (in module deepmd.pt.entrypoints.main), 415
log (in module deepmd.pt.infer.inference), 419
log (in module deepmd.pt.model.atomic_model.base_atomic_model), 434
log (in module deepmd.pt.model.atomic_model.dp_atomic_model), 440
log (in module deepmd.pt.model.descriptor.descriptor), 468
log (in module deepmd.pt.model.task.dipole), 580
log (in module deepmd.pt.model.task.dos), 583
log (in module deepmd.pt.model.task.ener), 585
log (in module deepmd.pt.model.task.fitting), 588
log (in module deepmd.pt.model.task.invar_fitting), 591
log (in module deepmd.pt.model.task.polarizability), 593
log (in module deepmd.pt.train.training), 609
log (in module deepmd.pt.train.wrapper), 610
log (in module deepmd.pt.utils.dataloader), 615
log (in module deepmd.pt.utils.finetune), 621
log (in module deepmd.pt.utils.preprocess), 630
log (in module deepmd.pt.utils.stat), 634
log (in module deepmd.tf.descriptor.se_a_ebd_v2), 668
log (in module deepmd.tf.descriptor.se_atten), 676
log (in module deepmd.tf.descriptor.se_atten_v2), 688
log (in module deepmd.tf.fit.dos), 763
log (in module deepmd.tf.fit.ener), 767
log (in module deepmd.tf.nvnmd.descriptor.se_a), 899
log (in module deepmd.tf.nvnmd.descriptor.se_atten), 900
log (in module deepmd.tf.nvnmd.entrypoints.mapt), 902
log (in module deepmd.tf.nvnmd.entrypoints.train), 904
log (in module deepmd.tf.nvnmd.entrypoints.wrap), 905
log (in module deepmd.tf.nvnmd.utils.config), 911
log (in module deepmd.tf.nvnmd.utils.encode), 913
log (in module deepmd.tf.nvnmd.utils.fio), 915
log (in module deepmd.tf.nvnmd.utils.network), 917
log (in module deepmd.tf.nvnmd.utils.weight), 919
log (in module deepmd.tf.op), 932
log (in module deepmd.tf.train.trainer), 934
log (in module deepmd.tf.utils.compress), 940

log (in module deepmd.tf.utils.convert), 941
log (in module deepmd.tf.utils.finetune), 951
log (in module deepmd.tf.utils.multi_init), 960
log (in module deepmd.tf.utils.neighbor_stat), 960
log (in module deepmd.tf.utils.tabulate), 979
log (in module deepmd.utils.argcheck), 1016
log (in module deepmd.utils.batch_size), 1020
log (in module deepmd.utils.data), 1025
log (in module deepmd.utils.data_system), 1030
log (in module deepmd.utils.env_mat_stat), 1036
log (in module deepmd.utils.finetune), 1039
log (in module deepmd.utils.neighbor_stat), 1041
log (in module deepmd.utils.pair_tab), 1043
log (in module deepmd.utils.summary), 1055
log (in module deepmd.utils.update_sel), 1056
loss (Argument)
    loss:, 148
Loss (class in deepmd.tf.loss.loss), 842
loss/type (Argument)
    type:, 148
loss:
    loss (Argument), 148
loss_args() (in module deepmd.utils.argcheck), 1018
loss_args_plugin (in module deepmd.utils.argcheck), 1018
loss_dict (Argument)
    loss_dict:, 154
loss_dict:
    loss_dict (Argument), 154
loss_dict_args() (in module deepmd.utils.argcheck), 1018
loss_dos() (in module deepmd.utils.argcheck), 1018
loss_ener() (in module deepmd.utils.argcheck), 1018
loss_ener_spin() (in module deepmd.utils.argcheck), 1018
loss_tensor() (in module deepmd.utils.argcheck), 1018
loss_variant_type_args() (in module deepmd.utils.argcheck), 1018
loss[dos]/limit_pref_acdf (Argument)
    limit_pref_acdf:, 153
loss[dos]/limit_pref_ados (Argument)
    limit_pref_ados:, 153
loss[dos]/limit_pref_cdf (Argument)
    limit_pref_cdf:, 153
loss[dos]/limit_pref_dos (Argument)
    limit_pref_dos:, 152
loss[dos]/start_pref_acdf (Argument)
    start_pref_acdf:, 153
loss[dos]/start_pref_ados (Argument)
    start_pref_ados:, 153
loss[dos]/start_pref_cdf (Argument)
    start_pref_cdf:, 153
loss[dos]/start_pref_dos (Argument)
    start_pref_dos:, 152
loss[ener_spin]/enable_atom_ener_coeff (Argument)
    enable_atom_ener_coeff:, 152
loss[ener_spin]/limit_pref_ae (Argument)
    limit_pref_ae:, 152
loss[ener_spin]/limit_pref_e (Argument)
    limit_pref_e:, 150
loss[ener_spin]/limit_pref_fm (Argument)
    limit_pref_fm:, 151
loss[ener_spin]/limit_pref_fr (Argument)
    limit_pref_fr:, 151
loss[ener_spin]/limit_pref_pf (Argument)
    limit_pref_pf:, 152
loss[ener_spin]/limit_pref_v (Argument)
    limit_pref_v:, 151
loss[ener_spin]/relative_f (Argument)
    relative_f:, 152
loss[ener_spin]/start_pref_ae (Argument)
    start_pref_ae:, 151
loss[ener_spin]/start_pref_e (Argument)
    start_pref_e:, 150
loss[ener_spin]/start_pref_fm (Argument)
    start_pref_fm:, 151
loss[ener_spin]/start_pref_fr (Argument)
    start_pref_fr:, 150
loss[ener_spin]/start_pref_pf (Argument)
    start_pref_pf:, 152
loss[ener_spin]/start_pref_v (Argument)
    start_pref_v:, 151
loss[ener]/enable_atom_ener_coeff (Argument)
    enable_atom_ener_coeff:, 150
loss[ener]/limit_pref_ae (Argument)
    limit_pref_ae:, 149
loss[ener]/limit_pref_e (Argument)
    limit_pref_e:, 148
loss[ener]/limit_pref_f (Argument)
    limit_pref_f:, 149
loss[ener]/limit_pref_gf (Argument)
    limit_pref_gf:, 150
loss[ener]/limit_pref_pf (Argument)
    limit_pref_pf:, 149
loss[ener]/limit_pref_v (Argument)
    limit_pref_v:, 149
loss[ener]/numb_generalized_coord (Argument)
    numb_generalized_coord:, 150
loss[ener]/relative_f (Argument)
    relative_f:, 149
loss[ener]/start_pref_ae (Argument)
    start_pref_ae:, 149
loss[ener]/start_pref_e (Argument)
    start_pref_e:, 148

loss[ener]/start_pref_f (Argument)
start_pref_f:, 148
loss[ener]/start_pref_gf (Argument)
start_pref_gf:, 150
loss[ener]/start_pref_pf (Argument)
start_pref_pf:, 149
loss[ener]/start_pref_v (Argument)
start_pref_v:, 149
loss[tensor]/pref (Argument)
pref:, 154
loss[tensor]/pref_atomic (Argument)
pref_atomic:, 154
lru_cache() (in module deepmd.pt.utils.cache), 614

M

MAG (deepmd.dpmodel.output_def.OutputVariableOperation attribute), 376
main() (in module deepmd.entrypoints.main), 384
main() (in module deepmd.main), 1066
main() (in module deepmd.pt.entrypoints.main), 415
main() (in module deepmd.tf.entrypoints.main), 750
main_parser() (in module deepmd.main), 1066
main_parser() (in module deepmd.tf.entrypoints.main), 750
make_atom_mask() (deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel method), 435
make_atom_mask() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 452
make_base_atomic_model() (in module deepmd.dpmodel.atomic_model), 277
make_base_atomic_model() (in module deepmd.dpmodel.atomic_model.make_base_atomic_model), 268
make_base_descriptor() (in module deepmd.dpmodel.descriptor), 304
make_base_descriptor() (in module deepmd.dpmodel.descriptor.make_base_descriptor), 290
make_base_fitting() (in module deepmd.dpmodel.fitting), 336
make_base_fitting() (in module deepmd.dpmodel.fitting.make_base_fitting), 323
make_base_model() (in module deepmd.dpmodel.model.base_model), 342
make_data() (in module deepmd.tf.utils.compress), 940
make_default_mesh() (in module deepmd.common), 1061
make_default_mesh() (in module deepmd.tf.common), 998
make_default_type_embedding() (in module deepmd.pt.model.descriptor), 507
make_default_type_embedding() (in module deepmd.pt.model.descriptor.descriptor), 469
make_econf_embedding() (in module deepmd.utils.econf_embd), 1035
make_element_embedding_list_vec() (in module deepmd.utils.econf_embd), 1035
make_embedding_network() (in module deepmd.dpmodel.utils), 370
make_embedding_network() (in module deepmd.dpmodel.utils.network), 358
make_empty_list_vec() (in module deepmd.utils.econf_embd), 1035
make_env_mat() (in module deepmd.pt.utils.preprocess), 631
make_fitting_network() (in module deepmd.dpmodel.utils), 370
make_fitting_network() (in module deepmd.dpmodel.utils.network), 358
make_hessian_model() (in module deepmd.pt.model.model), 544
make_hessian_model() (in module deepmd.pt.model.model.make_hessian_model), 534
make_index() (in module deepmd.utils.argcheck), 1019
make_link() (in module deepmd.utils.argcheck), 1016
make_model() (in module deepmd.dpmodel.model), 348
make_model() (in module deepmd.dpmodel.model.make_model), 343
make_model() (in module deepmd.pt.model.model), 545
make_model() (in module deepmd.pt.model.model.make_model), 535
make_model_devi() (in module deepmd.infer.model_devi), 405
make_model_devi() (in module deepmd.tf.entrypoints), 755
make_model_devi() (in module deepmd.tf.infer.model_devi), 818
make_multilayer_network() (in module deepmd.dpmodel.utils), 370
make_multilayer_network() (in module deepmd.dpmodel.utils.network), 358
make_natoms_vec() (deepmd.tf.DeepEval method), 1007
make_natoms_vec() (deepmd.tf.infer.deep_eval.DeepEval method), 800
make_natoms_vec() (deepmd.tf.infer.deep_eval.DeepEvalOld method), 806

make_natoms_vec() (deepmd.tf.infer.DeepEval method), 826
make_plugin_registry() (in module deepmd.utils.plugin), 1052
make_stat_input() (in module deepmd.pt.utils.stat), 634
make_stat_input() (in module deepmd.tf.model.model_stat), 868
make_stat_input() (in module deepmd.utils.model_stat), 1040
map_aparam() (in module deepmd.tf.env.op_module), 1103
map_file:
    nvnmd/map_file (Argument), 160
map_flt_nvnmd() (in module deepmd.tf.env.op_module), 1103
map_nvnmd() (in module deepmd.tf.nvnmd.utils), 923
map_nvnmd() (in module deepmd.tf.nvnmd.utils.op), 918
MapAparam() (in module deepmd.tf.env.op_module), 1077
MapFltNvnmd() (in module deepmd.tf.env.op_module), 1078
mapping() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 903
mapping() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
mapping2() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 903
mapping2() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
mapt() (in module deepmd.tf.nvnmd.entrypoints.mapt), 904
MapTable (class in deepmd.tf.nvnmd.entrypoints), 907
MapTable (class in deepmd.tf.nvnmd.entrypoints.mapt), 902
MaskLMHead (class in deepmd.pt.model.network.network), 562
matmul2_qq() (in module deepmd.tf.nvnmd.utils.network), 917
matmul3_qq() (in module deepmd.tf.nvnmd.utils.network), 917
matmul_fitnet_nvnmd() (in module deepmd.tf.env.op_module), 1104
matmul_flt2fix_nvnmd() (in module deepmd.tf.env.op_module), 1104
matmul_flt_nvnmd() (in module deepmd.tf.env.op_module), 1104
MatmulFitnetNvnmd() (in module deepmd.tf.env.op_module), 1078
MatmulFlt2fixNvnmd() (in module deepmd.tf.env.op_module), 1078
MatmulFltNvnmd() (in module deepmd.tf.env.op_module), 1078
max_ckpt_keep:
    training/max_ckpt_keep (Argument), 157
max_nnei:
    nvnmd/max_nnei (Argument), 159
maxl (in module deepmd.utils.econf_embd), 1035
maxm (in module deepmd.utils.econf_embd), 1035
maxn (in module deepmd.utils.econf_embd), 1035
merge_bin() (deepmd.tf.nvnmd.utils.Encode method), 922
merge_bin() (deepmd.tf.nvnmd.utils.encode.Encode method), 914
merge_input_stats() (deepmd.tf.descriptor.DescrptHybrid method), 707
merge_input_stats() (deepmd.tf.descriptor.DescrptSeA method), 716
merge_input_stats() (deepmd.tf.descriptor.DescrptSeR method), 739
merge_input_stats() (deepmd.tf.descriptor.DescrptSeT method), 743
merge_input_stats() (deepmd.tf.descriptor.hybrid.DescrptHybrid method), 649
merge_input_stats() (deepmd.tf.descriptor.se_a.DescrptSeA method), 662
merge_input_stats() (deepmd.tf.descriptor.se_r.DescrptSeR method), 691
merge_input_stats() (deepmd.tf.descriptor.se_t.DescrptSeT method), 696
merge_sys_stat() (in module deepmd.tf.model.model_stat), 868
merge_sys_stat() (in module deepmd.utils.model_stat), 1040
mixed_precision:
    training/mixed_precision (Argument), 156
mixed_precision_args() (in module deepmd.utils.argcheck), 1018
mixed_types() (deepmd.dpmodel.atomic_model.dp_atomic_model.D method), 263
mixed_types() (deepmd.dpmodel.atomic_model.DPAtomicModel method), 273
mixed_types() (deepmd.dpmodel.atomic_model.linear_atomic_mode method), 265
mixed_types() (deepmd.dpmodel.atomic_model.LinearEnergyAtomi method), 276
mixed_types() (deepmd.dpmodel.atomic_model.pairtab_atomic_mo

method), 269
mixed_types() (deepmd.dpmodel.atomic_model.PairTabAtomicModel method), 278
mixed_types() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 301
mixed_types() (deepmd.dpmodel.descriptor.DescrptHybrid method), 303
mixed_types() (deepmd.dpmodel.descriptor.DescrptSeA method), 306
mixed_types() (deepmd.dpmodel.descriptor.DescrptSeR method), 308
mixed_types() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 method), 285
mixed_types() (deepmd.dpmodel.descriptor.hybrid.DescrptHybrid method), 288
mixed_types() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA method), 293
mixed_types() (deepmd.dpmodel.descriptor.se_r.DescrptSeR method), 296
mixed_types() (deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel method), 440
mixed_types() (deepmd.pt.model.atomic_model.DPAtomicModel method), 455
mixed_types() (deepmd.pt.model.atomic_model.linear_atomic_model.LinearEnergyAtomicModel method), 443
mixed_types() (deepmd.pt.model.atomic_model.LinearEnergyAtomicModel method), 458
mixed_types() (deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 447
mixed_types() (deepmd.pt.model.atomic_model.PairTabAtomicModel method), 461
mixed_types() (deepmd.pt.model.descriptor.DescrptBlockRepformers method), 521
mixed_types() (deepmd.pt.model.descriptor.DescrptBlockSeA method), 523
mixed_types() (deepmd.pt.model.descriptor.DescrptBlockSeAtten method), 508
mixed_types() (deepmd.pt.model.descriptor.DescrptDPA1 method), 512
mixed_types() (deepmd.pt.model.descriptor.DescrptDPA2 method), 515
mixed_types() (deepmd.pt.model.descriptor.DescrptHybrid method), 519
mixed_types() (deepmd.pt.model.descriptor.DescrptSeA method), 524
mixed_types() (deepmd.pt.model.descriptor.DescrptSeR method), 526
mixed_types() (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 method), 473
mixed_types() (deepmd.pt.model.descriptor.dpa2.DescrptDPA2 method), 477
mixed_types() (deepmd.pt.model.descriptor.hybrid.DescrptHybrid method), 482
mixed_types() (deepmd.pt.model.descriptor.repformers.DescrptBlockRepformers method), 491
mixed_types() (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA method), 495
mixed_types() (deepmd.pt.model.descriptor.se_a.DescrptSeA method), 492
mixed_types() (deepmd.pt.model.descriptor.se_atten.DescrptBlockSeAtten method), 497
mixed_types() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 503
mixed_types() (deepmd.pt.model.model.frozen.FrozenModel method), 533
mixed_types() (deepmd.pt.model.model.FrozenModel method), 544
mkdir() (deepmd.tf.nvnmd.utils.fio.Fio method), 915
mkdir() (deepmd.tf.utils.path.DPH5Path method), 970
mkdir() (deepmd.tf.utils.path.DPOSPath method), 972
mkdir() (deepmd.tf.utils.path.DPPath method), 974
mkdir() (deepmd.utils.path.DPH5Path method), 1050
mkdir() (deepmd.utils.path.DPOSPath method), 1048
mkdir() (deepmd.utils.path.DPPath method), 1047
MLP (class in deepmd.pt.model.network.mlp), 552
MLP_ (in module deepmd.pt.model.network.mlp), 552
MLPLayer (class in deepmd.pt.model.network.mlp), 551
MODIFIER_ARGS_PLUGIN (C macro), 1324
model (Argument)
Model (class in deepmd.tf.model.model), 860
model/atom_exclude_types (Argument)
atom_exclude_types:, 112
model/compress (Argument)
compress:, 114
model/data_bias_nsample (Argument)
data_bias_nsample:, 111
model/data_stat_nbatch (Argument)
data_stat_nbatch:, 111
model/data_stat_protect (Argument)
data_stat_protect:, 111
model/modifier (Argument)
modifier:, 113
model/modifier/type (Argument)
type:, 113
model/modifier[dipole_charge]/ewald_beta (Argument)
ewald_beta:, 114
model/modifier[dipole_charge]/ewald_h (Argument)
ewald_h:, 114

model/modifier[dipole_charge]/model_charge_map (Argument)
    model_charge_map:, 114
model/modifier[dipole_charge]/model_name (Argument)
    model_name:, 113
model/modifier[dipole_charge]/sys_charge_map (Argument)
    sys_charge_map:, 114
model/pair_exclude_types (Argument)
    pair_exclude_types:, 112
model/smin_alpha (Argument)
    smin_alpha:, 111
model/spin (Argument)
    spin:, 114
model/spin/spin_norm (Argument)
    spin_norm:, 114
model/spin/use_spin (Argument)
    use_spin:, 114
model/spin/virtual_len (Argument)
    virtual_len:, 114
model/spin/virtual_scale (Argument)
    virtual_scale:, 115
model/srtab_add_bias (Argument)
    srtab_add_bias:, 112
model/sw_rmax (Argument)
    sw_rmax:, 112
model/sw_rmin (Argument)
    sw_rmin:, 112
model/type (Argument)
    type:, 115
model/type_embedding (Argument)
    type_embedding:, 112
model/type_embedding/activation_function (Argument)
    activation_function:, 112
model/type_embedding/neuron (Argument)
    neuron:, 112
model/type_embedding/precision (Argument)
    precision:, 113
model/type_embedding/resnet_dt (Argument)
    resnet_dt:, 113
model/type_embedding/seed (Argument)
    seed:, 113
model/type_embedding/trainable (Argument)
    trainable:, 113
model/type_map (Argument)
    type_map:, 111
model/use_srtab (Argument)
    use_srtab:, 111
model:
    model (Argument), 111
model_args() (in module deepmd.utils.argcheck), 1018
model_charge_map:
    model/modifier[dipole_charge]/model_charge_map (Argument), 114
model_check_output() (in module deepmd.dpmodel), 382
model_check_output() (in module deepmd.dpmodel.output_def), 375
model_compression() (in module deepmd.utils.argcheck), 1018
model_compression_type_args() (in module deepmd.utils.argcheck), 1018
model_file:
    model[frozen]/model_file (Argument), 145
model_name:
    model/modifier[dipole_charge]/model_name (Argument), 113
model_output_type() (deepmd.dpmodel.model.spin_model.SpinModel method), 345
model_output_type() (deepmd.dpmodel.model.SpinModel method), 349
model_output_type() (deepmd.pt.model.model.frozen.FrozenModel method), 534
model_output_type() (deepmd.pt.model.model.FrozenModel method), 544
model_output_type() (deepmd.pt.model.model.spin_model.SpinModel method), 539
model_output_type() (deepmd.pt.model.model.SpinModel method), 547
model_type (deepmd.dpmodel.infer.deep_eval.DeepEval property), 339
model_type (deepmd.infer.deep_eval.DeepEvalBackend property), 390
model_type (deepmd.pt.infer.deep_eval.DeepEval property), 416
model_type (deepmd.pt.model.model.dipole_model.DipoleModel attribute), 529
model_type (deepmd.pt.model.model.dos_model.DOSModel attribute), 530
model_type (deepmd.pt.model.model.dp_zbl_model.DPZBLModel attribute), 531
model_type (deepmd.pt.model.model.DPZBLModel attribute), 542
model_type (deepmd.pt.model.model.ener_model.EnergyModel attribute), 532
model_type (deepmd.pt.model.model.EnergyModel attribute), 543
model_type (deepmd.pt.model.model.polar_model.PolarModel attribute), 537

model_type (deepmd.pt.model.model.spin_model.SpinEnergyModel attribute), 539
model_type (deepmd.pt.model.model.SpinEnergyModel attribute), 546
model_type (deepmd.tf.DeepEval property), 1006
model_type (deepmd.tf.infer.deep_eval.DeepEval property), 799
model_type (deepmd.tf.infer.deep_eval.DeepEvalOld property), 804
model_type (deepmd.tf.infer.DeepEval property), 824
model_type (deepmd.tf.model.dos.DOSModel attribute), 851
model_type (deepmd.tf.model.DOSModel attribute), 889
model_type (deepmd.tf.model.ener.EnerModel attribute), 854
model_type (deepmd.tf.model.EnerModel attribute), 891
model_type (deepmd.tf.model.linear.LinearEnergyModel attribute), 859
model_type (deepmd.tf.model.LinearEnergyModel attribute), 883
model_type (deepmd.tf.model.multi.MultiModel attribute), 870
model_type (deepmd.tf.model.MultiModel attribute), 894
model_type (deepmd.tf.model.pairtab.PairTabModel attribute), 872
model_type (deepmd.tf.model.PairTabModel attribute), 884
model_type (deepmd.tf.model.pairwise_dprc.PairwiseDPRc attribute), 874
model_type (deepmd.tf.model.PairwiseDPRc attribute), 886
model_version (deepmd.tf.DeepEval property), 1006
model_version (deepmd.tf.infer.deep_eval.DeepEval property), 799
model_version (deepmd.tf.infer.deep_eval.DeepEvalOld property), 804
model_version (deepmd.tf.infer.DeepEval property), 824
MODEL_VERSION (in module deepmd.tf.env), 1003
ModelOutputDef (class in deepmd.dpmodel), 381
ModelOutputDef (class in deepmd.dpmodel.output_def), 378
models:
    model[linear_ener]/models (Argument), 146
ModelWrapper (class in deepmd.pt.train.wrapper), 610
model[frozen]/model_file (Argument)
    model_file:, 145
model[linear_ener]/models (Argument)
    models:, 146
model[linear_ener]/weights (Argument)
    weights:, 147
model[multi]/descriptor (Argument)
    descriptor:, 145
model[multi]/fitting_net_dict (Argument)
    fitting_net_dict:, 145
model[pairtab]/rcut (Argument)
    rcut:, 146
model[pairtab]/sel (Argument)
    sel:, 146
model[pairtab]/tab_file (Argument)
    tab_file:, 146
model[pairwise_dprc]/qm_model (Argument)
    qm_model:, 146
model[pairwise_dprc]/qmmm_model (Argument)
    qmmm_model:, 146
model[standard]/descriptor (Argument)
    descriptor:, 115
model[standard]/descriptor/type (Argument)
    type:, 115
model[standard]/descriptor[dpa2]/concat_output_tebd (Argument)
    concat_output_tebd:, 132
model[standard]/descriptor[dpa2]/repformer_activation (Argument)
    repformer_activation:, 135
model[standard]/descriptor[dpa2]/repformer_add_type_ebd_to (Argument)
    repformer_add_type_ebd_to_seq:, 135
model[standard]/descriptor[dpa2]/repformer_attn1_hidden (Argument)
    repformer_attn1_hidden:, 135
model[standard]/descriptor[dpa2]/repformer_attn1_nhead (Argument)
    repformer_attn1_nhead:, 135
model[standard]/descriptor[dpa2]/repformer_attn2_has_gate (Argument)
    repformer_attn2_has_gate:, 135
model[standard]/descriptor[dpa2]/repformer_attn2_hidden (Argument)
    repformer_attn2_hidden:, 135
model[standard]/descriptor[dpa2]/repformer_attn2_nhead (Argument)
    repformer_attn2_nhead:, 135
model[standard]/descriptor[dpa2]/repformer_axis_dim (Argument)
    repformer_axis_dim:, 133
model[standard]/descriptor[dpa2]/repformer_bn_momentum (Argument)
    repformer_bn_momentum:, 134
model[standard]/descriptor[dpa2]/repformer_do_bn_mode (Argument)
    repformer_do_bn_mode:, 133

model[standard]/descriptor[dpa2]/repformer_g1_dim
(Argument)
repformer_g1_dim:, 133
model[standard]/descriptor[dpa2]/repformer_g2_dim
(Argument)
repformer_g2_dim:, 133
model[standard]/descriptor[dpa2]/repformer_nlayers
(Argument)
repformer_nlayers:, 133
model[standard]/descriptor[dpa2]/repformer_nsel
(Argument)
repformer_nsel:, 132
model[standard]/descriptor[dpa2]/repformer_rcut
(Argument)
repformer_rcut:, 132
model[standard]/descriptor[dpa2]/repformer_rcut_smth
(Argument)
repformer_rcut_smth:, 132
model[standard]/descriptor[dpa2]/repformer_set_davg_zero
(Argument)
repformer_set_davg_zero:, 135
model[standard]/descriptor[dpa2]/repformer_update_g1_has_attn
(Argument)
repformer_update_g1_has_attn:, 134
model[standard]/descriptor[dpa2]/repformer_update_g1_has_conv
(Argument)
repformer_update_g1_has_conv:, 134
model[standard]/descriptor[dpa2]/repformer_update_g1_has_drrd
(Argument)
repformer_update_g1_has_drrd:, 134
model[standard]/descriptor[dpa2]/repformer_update_g1_has_grrg
(Argument)
repformer_update_g1_has_grrg:, 134
model[standard]/descriptor[dpa2]/repformer_update_g2_has_attn
(Argument)
repformer_update_g2_has_attn:, 134
model[standard]/descriptor[dpa2]/repformer_update_g2_has_g1g1
(Argument)
repformer_update_g2_has_g1g1:, 134
model[standard]/descriptor[dpa2]/repformer_update_h2
(Argument)
repformer_update_h2:, 134
model[standard]/descriptor[dpa2]/repformer_update_style
(Argument)
repformer_update_style:, 135
model[standard]/descriptor[dpa2]/repinit_activation
(Argument)
repinit_activation:, 133
model[standard]/descriptor[dpa2]/repinit_axis_neuron
(Argument)
repinit_axis_neuron:, 133
model[standard]/descriptor[dpa2]/repinit_neuron
(Argument)
repinit_neuron:, 132
model[standard]/descriptor[dpa2]/repinit_nsel
(Argument)
repinit_nsel:, 132
model[standard]/descriptor[dpa2]/repinit_rcut
(Argument)
repinit_rcut:, 132
model[standard]/descriptor[dpa2]/repinit_rcut_smth
(Argument)
repinit_rcut_smth:, 132
model[standard]/descriptor[dpa2]/repinit_set_davg_zero
(Argument)
repinit_set_davg_zero:, 133
model[standard]/descriptor[dpa2]/tebd_dim
(Argument)
tebd_dim:, 132
model[standard]/descriptor[hybrid]/list
(Argument)
list:, 125
model[standard]/descriptor[loc_frame]/axis_rule
(Argument)
axis_rule:, 116
model[standard]/descriptor[loc_frame]/rcut
(Argument)
rcut:, 116
model[standard]/descriptor[loc_frame]/sel_a
(Argument)
sel_a:, 116
model[standard]/descriptor[loc_frame]/sel_r
(Argument)
sel_r:, 116
model[standard]/descriptor[se_a_ebd_v2]/activation_functio
(Argument)
activation_function:, 136
model[standard]/descriptor[se_a_ebd_v2]/axis_neuron
(Argument)
axis_neuron:, 136
model[standard]/descriptor[se_a_ebd_v2]/env_protection
(Argument)
env_protection:, 138
model[standard]/descriptor[se_a_ebd_v2]/exclude_types
(Argument)
exclude_types:, 137
model[standard]/descriptor[se_a_ebd_v2]/neuron
(Argument)
neuron:, 136
model[standard]/descriptor[se_a_ebd_v2]/precision
(Argument)
precision:, 137
model[standard]/descriptor[se_a_ebd_v2]/rcut
(Argument)
rcut:, 136
model[standard]/descriptor[se_a_ebd_v2]/rcut_smth
(Argument)
rcut_smth:, 136

model[standard]/descriptor[se_a_ebd_v2]/resnet_dt
(Argument)
resnet_dt:, 137
model[standard]/descriptor[se_a_ebd_v2]/seed
(Argument)
seed:, 137
model[standard]/descriptor[se_a_ebd_v2]/sel
(Argument)
sel:, 136
model[standard]/descriptor[se_a_ebd_v2]/set_davg_zero
(Argument)
set_davg_zero:, 138
model[standard]/descriptor[se_a_ebd_v2]/trainable
(Argument)
trainable:, 137
model[standard]/descriptor[se_a_ebd_v2]/type_one_side
(Argument)
type_one_side:, 137
model[standard]/descriptor[se_a_mask]/activation_function
(Argument)
activation_function:, 138
model[standard]/descriptor[se_a_mask]/axis_neuron
(Argument)
axis_neuron:, 138
model[standard]/descriptor[se_a_mask]/exclude_types
(Argument)
exclude_types:, 139
model[standard]/descriptor[se_a_mask]/neuron
(Argument)
neuron:, 138
model[standard]/descriptor[se_a_mask]/precision
(Argument)
precision:, 139
model[standard]/descriptor[se_a_mask]/resnet_dt
(Argument)
resnet_dt:, 139
model[standard]/descriptor[se_a_mask]/seed
(Argument)
seed:, 139
model[standard]/descriptor[se_a_mask]/sel
(Argument)
sel:, 138
model[standard]/descriptor[se_a_mask]/trainable
(Argument)
trainable:, 139
model[standard]/descriptor[se_a_mask]/type_one_side
(Argument)
type_one_side:, 139
model[standard]/descriptor[se_a_tpe]/activation_function
(Argument)
activation_function:, 121
model[standard]/descriptor[se_a_tpe]/axis_neuron
(Argument)
axis_neuron:, 121
model[standard]/descriptor[se_a_tpe]/env_protection
(Argument)
env_protection:, 122
model[standard]/descriptor[se_a_tpe]/exclude_types
(Argument)
exclude_types:, 122
model[standard]/descriptor[se_a_tpe]/neuron
(Argument)
neuron:, 121
model[standard]/descriptor[se_a_tpe]/numb_aparam
(Argument)
numb_aparam:, 123
model[standard]/descriptor[se_a_tpe]/precision
(Argument)
precision:, 122
model[standard]/descriptor[se_a_tpe]/rcut
(Argument)
rcut:, 121
model[standard]/descriptor[se_a_tpe]/rcut_smth
(Argument)
rcut_smth:, 121
model[standard]/descriptor[se_a_tpe]/resnet_dt
(Argument)
resnet_dt:, 121
model[standard]/descriptor[se_a_tpe]/seed
(Argument)
seed:, 122
model[standard]/descriptor[se_a_tpe]/sel
(Argument)
sel:, 120
model[standard]/descriptor[se_a_tpe]/set_davg_zero
(Argument)
set_davg_zero:, 122
model[standard]/descriptor[se_a_tpe]/trainable
(Argument)
trainable:, 122
model[standard]/descriptor[se_a_tpe]/type_nchanl
(Argument)
type_nchanl:, 122
model[standard]/descriptor[se_a_tpe]/type_nlayer
(Argument)
type_nlayer:, 123
model[standard]/descriptor[se_a_tpe]/type_one_side
(Argument)
type_one_side:, 121
model[standard]/descriptor[se_atten_v2]/activation_function
(Argument)
activation_function:, 130
model[standard]/descriptor[se_atten_v2]/attn
(Argument)
attn:, 131
model[standard]/descriptor[se_atten_v2]/attn_dotr
(Argument)
attn_dotr:, 131

model[standard]/descriptor[se_atten_v2]/attn_layer
(Argument)
attn_layer:, 131
model[standard]/descriptor[se_atten_v2]/attn_mask
(Argument)
attn_mask:, 131
model[standard]/descriptor[se_atten_v2]/axis_neuron
(Argument)
axis_neuron:, 130
model[standard]/descriptor[se_atten_v2]/exclude_types
(Argument)
exclude_types:, 131
model[standard]/descriptor[se_atten_v2]/neuron
(Argument)
neuron:, 130
model[standard]/descriptor[se_atten_v2]/precision
(Argument)
precision:, 130
model[standard]/descriptor[se_atten_v2]/rcut
(Argument)
rcut:, 129
model[standard]/descriptor[se_atten_v2]/rcut_smth
(Argument)
rcut_smth:, 129
model[standard]/descriptor[se_atten_v2]/resnet_dt
(Argument)
resnet_dt:, 130
model[standard]/descriptor[se_atten_v2]/seed
(Argument)
seed:, 131
model[standard]/descriptor[se_atten_v2]/sel
(Argument)
sel:, 129
model[standard]/descriptor[se_atten_v2]/set_davg_zero
(Argument)
set_davg_zero:, 131
model[standard]/descriptor[se_atten_v2]/trainable
(Argument)
trainable:, 130
model[standard]/descriptor[se_atten_v2]/type_one_side
(Argument)
type_one_side:, 130
model[standard]/descriptor[se_atten]/activation_function
(Argument)
activation_function:, 126
model[standard]/descriptor[se_atten]/attn
(Argument)
attn:, 127
model[standard]/descriptor[se_atten]/attn_dotr
(Argument)
attn_dotr:, 127
model[standard]/descriptor[se_atten]/attn_layer
(Argument)
attn_layer:, 127
model[standard]/descriptor[se_atten]/attn_mask
(Argument)
attn_mask:, 127
model[standard]/descriptor[se_atten]/axis_neuron
(Argument)
axis_neuron:, 126
model[standard]/descriptor[se_atten]/concat_output_tebd
(Argument)
concat_output_tebd:, 129
model[standard]/descriptor[se_atten]/exclude_types
(Argument)
exclude_types:, 126
model[standard]/descriptor[se_atten]/ln_eps
(Argument)
ln_eps:, 128
model[standard]/descriptor[se_atten]/neuron
(Argument)
neuron:, 125
model[standard]/descriptor[se_atten]/normalize
(Argument)
normalize:, 128
model[standard]/descriptor[se_atten]/precision
(Argument)
precision:, 126
model[standard]/descriptor[se_atten]/rcut
(Argument)
rcut:, 125
model[standard]/descriptor[se_atten]/rcut_smth
(Argument)
rcut_smth:, 125
model[standard]/descriptor[se_atten]/resnet_dt
(Argument)
resnet_dt:, 126
model[standard]/descriptor[se_atten]/scaling_factor
(Argument)
scaling_factor:, 128
model[standard]/descriptor[se_atten]/seed
(Argument)
seed:, 126
model[standard]/descriptor[se_atten]/sel
(Argument)
sel:, 125
model[standard]/descriptor[se_atten]/set_davg_zero
(Argument)
set_davg_zero:, 127
model[standard]/descriptor[se_atten]/smooth_type_embedding
(Argument)
smooth_type_embedding:, 127
model[standard]/descriptor[se_atten]/stripped_type_embeddi
(Argument)
stripped_type_embedding:, 127
model[standard]/descriptor[se_atten]/tebd_dim
(Argument)
tebd_dim:, 128

model[standard]/descriptor[se_atten]/tebd_input_mode
(Argument)
tebd_input_mode:, 128
model[standard]/descriptor[se_atten]/temperature
(Argument)
temperature:, 129
model[standard]/descriptor[se_atten]/trainable
(Argument)
trainable:, 126
model[standard]/descriptor[se_atten]/trainable_ln
(Argument)
trainable_ln:, 128
model[standard]/descriptor[se_atten]/type_one_side
(Argument)
type_one_side:, 126
model[standard]/descriptor[se_e2_a]/activation_function
(Argument)
activation_function:, 117
model[standard]/descriptor[se_e2_a]/axis_neuron
(Argument)
axis_neuron:, 117
model[standard]/descriptor[se_e2_a]/env_protection
(Argument)
env_protection:, 118
model[standard]/descriptor[se_e2_a]/exclude_types
(Argument)
exclude_types:, 118
model[standard]/descriptor[se_e2_a]/neuron
(Argument)
neuron:, 117
model[standard]/descriptor[se_e2_a]/precision
(Argument)
precision:, 118
model[standard]/descriptor[se_e2_a]/rcut
(Argument)
rcut:, 117
model[standard]/descriptor[se_e2_a]/rcut_smth
(Argument)
rcut_smth:, 117
model[standard]/descriptor[se_e2_a]/resnet_dt
(Argument)
resnet_dt:, 118
model[standard]/descriptor[se_e2_a]/seed
(Argument)
seed:, 118
model[standard]/descriptor[se_e2_a]/sel
(Argument)
sel:, 117
model[standard]/descriptor[se_e2_a]/set_davg_zero
(Argument)
set_davg_zero:, 119
model[standard]/descriptor[se_e2_a]/trainable
(Argument)
trainable:, 118
model[standard]/descriptor[se_e2_a]/type_one_side
(Argument)
type_one_side:, 118
model[standard]/descriptor[se_e2_r]/activation_function
(Argument)
activation_function:, 124
model[standard]/descriptor[se_e2_r]/exclude_types
(Argument)
exclude_types:, 124
model[standard]/descriptor[se_e2_r]/neuron
(Argument)
neuron:, 123
model[standard]/descriptor[se_e2_r]/precision
(Argument)
precision:, 124
model[standard]/descriptor[se_e2_r]/rcut
(Argument)
rcut:, 123
model[standard]/descriptor[se_e2_r]/rcut_smth
(Argument)
rcut_smth:, 123
model[standard]/descriptor[se_e2_r]/resnet_dt
(Argument)
resnet_dt:, 124
model[standard]/descriptor[se_e2_r]/seed
(Argument)
seed:, 124
model[standard]/descriptor[se_e2_r]/sel
(Argument)
sel:, 123
model[standard]/descriptor[se_e2_r]/set_davg_zero
(Argument)
set_davg_zero:, 125
model[standard]/descriptor[se_e2_r]/trainable
(Argument)
trainable:, 124
model[standard]/descriptor[se_e2_r]/type_one_side
(Argument)
type_one_side:, 124
model[standard]/descriptor[se_e3]/activation_function
(Argument)
activation_function:, 120
model[standard]/descriptor[se_e3]/neuron
(Argument)
neuron:, 119
model[standard]/descriptor[se_e3]/precision
(Argument)
precision:, 120
model[standard]/descriptor[se_e3]/rcut
(Argument)
rcut:, 119
model[standard]/descriptor[se_e3]/rcut_smth
(Argument)
rcut_smth:, 119

model[standard]/descriptor[se_e3]/resnet_dt (Argument)
    resnet_dt:, 120
model[standard]/descriptor[se_e3]/seed (Argument)
    seed:, 120
model[standard]/descriptor[se_e3]/sel (Argument)
    sel:, 119
model[standard]/descriptor[se_e3]/set_davg_zero (Argument)
    set_davg_zero:, 120
model[standard]/descriptor[se_e3]/trainable (Argument)
    trainable:, 120
model[standard]/fitting_net (Argument)
    fitting_net:, 139
model[standard]/fitting_net/type (Argument)
    type:, 140
model[standard]/fitting_net[dipole]/activation_function (Argument)
    activation_function:, 144
model[standard]/fitting_net[dipole]/neuron (Argument)
    neuron:, 144
model[standard]/fitting_net[dipole]/precision (Argument)
    precision:, 145
model[standard]/fitting_net[dipole]/resnet_dt (Argument)
    resnet_dt:, 145
model[standard]/fitting_net[dipole]/seed (Argument)
    seed:, 145
model[standard]/fitting_net[dipole]/sel_type (Argument)
    sel_type:, 145
model[standard]/fitting_net[dos]/activation_function (Argument)
    activation_function:, 142
model[standard]/fitting_net[dos]/neuron (Argument)
    neuron:, 142
model[standard]/fitting_net[dos]/numb_aparam (Argument)
    numb_aparam:, 142
model[standard]/fitting_net[dos]/numb_dos (Argument)
    numb_dos:, 143
model[standard]/fitting_net[dos]/numb_fparam (Argument)
    numb_fparam:, 142
model[standard]/fitting_net[dos]/precision (Argument)
    precision:, 142
model[standard]/fitting_net[dos]/rcond (Argument)
    rcond:, 143
model[standard]/fitting_net[dos]/resnet_dt (Argument)
    resnet_dt:, 142
model[standard]/fitting_net[dos]/seed (Argument)
    seed:, 143
model[standard]/fitting_net[dos]/trainable (Argument)
    trainable:, 143
model[standard]/fitting_net[ener]/activation_function (Argument)
    activation_function:, 140
model[standard]/fitting_net[ener]/atom_ener (Argument)
    atom_ener:, 141
model[standard]/fitting_net[ener]/layer_name (Argument)
    layer_name:, 141
model[standard]/fitting_net[ener]/neuron (Argument)
    neuron:, 140
model[standard]/fitting_net[ener]/numb_aparam (Argument)
    numb_aparam:, 140
model[standard]/fitting_net[ener]/numb_fparam (Argument)
    numb_fparam:, 140
model[standard]/fitting_net[ener]/precision (Argument)
    precision:, 141
model[standard]/fitting_net[ener]/rcond (Argument)
    rcond:, 141
model[standard]/fitting_net[ener]/resnet_dt (Argument)
    resnet_dt:, 141
model[standard]/fitting_net[ener]/seed (Argument)
    seed:, 141
model[standard]/fitting_net[ener]/trainable (Argument)
    trainable:, 141
model[standard]/fitting_net[ener]/use_aparam_as_mask (Argument)
    use_aparam_as_mask:, 142
model[standard]/fitting_net[polar]/activation_function (Argument)
    activation_function:, 143
model[standard]/fitting_net[polar]/fit_diag (Argument)

fit_diag:, 144
model[standard]/fitting_net[polar]/neuron (Argument)
neuron:, 143
model[standard]/fitting_net[polar]/precision (Argument)
precision:, 144
model[standard]/fitting_net[polar]/resnet_dt (Argument)
resnet_dt:, 143
model[standard]/fitting_net[polar]/scale (Argument)
scale:, 144
model[standard]/fitting_net[polar]/seed (Argument)
seed:, 144
model[standard]/fitting_net[polar]/sel_type (Argument)
sel_type:, 144
model[standard]/fitting_net[polar]/shift_diag (Argument)
shift_diag:, 144
modifier:
model/modifier (Argument), 113
modifier_dipole_charge() (in module deepmd.utils.argcheck), 1018
modifier_variant_type_args() (in module deepmd.utils.argcheck), 1018
modify_data() (deepmd.tf.DipoleChargeModifier method), 1012
modify_data() (deepmd.tf.infer.data_modifier.DipoleChargeModifier method), 795
modify_data() (deepmd.tf.infer.DipoleChargeModifier method), 821
module
deepmd, 253
deepmd.__main__, 1058
deepmd.backend, 253
deepmd.backend.backend, 253
deepmd.backend.dpmodel, 256
deepmd.backend.pytorch, 257
deepmd.backend.suffix, 258
deepmd.backend.tensorflow, 259
deepmd.calculator, 1058
deepmd.common, 1060
deepmd.dpmodel, 260
deepmd.dpmodel.atomic_model, 260
deepmd.dpmodel.atomic_model.base_atomic_model, 261
deepmd.dpmodel.atomic_model.dp_atomic_model, 263
deepmd.dpmodel.atomic_model.linear_atomic_model, 264
deepmd.dpmodel.atomic_model.make_base_atomic_model, 268
deepmd.dpmodel.atomic_model.pairtab_atomic_model, 268
deepmd.dpmodel.common, 374
deepmd.dpmodel.descriptor, 281
deepmd.dpmodel.descriptor.base_descriptor, 281
deepmd.dpmodel.descriptor.dpa1, 281
deepmd.dpmodel.descriptor.hybrid, 288
deepmd.dpmodel.descriptor.make_base_descriptor, 290
deepmd.dpmodel.descriptor.se_e2_a, 290
deepmd.dpmodel.descriptor.se_r, 294
deepmd.dpmodel.fitting, 310
deepmd.dpmodel.fitting.base_fitting, 310
deepmd.dpmodel.fitting.dipole_fitting, 310
deepmd.dpmodel.fitting.dos_fitting, 313
deepmd.dpmodel.fitting.ener_fitting, 315
deepmd.dpmodel.fitting.general_fitting, 317
deepmd.dpmodel.fitting.invar_fitting, 320
deepmd.dpmodel.fitting.make_base_fitting, 323
deepmd.dpmodel.fitting.polarizability_fitting, 324
deepmd.dpmodel.infer, 338
deepmd.dpmodel.infer.deep_eval, 338
deepmd.dpmodel.model, 341
deepmd.dpmodel.model.base_model, 341
deepmd.dpmodel.model.dp_model, 342
deepmd.dpmodel.model.ener_model, 343
deepmd.dpmodel.model.make_model, 343
deepmd.dpmodel.model.model, 344
deepmd.dpmodel.model.spin_model, 344
deepmd.dpmodel.model.transform_output, 347
deepmd.dpmodel.output_def, 374
deepmd.dpmodel.utils, 351
deepmd.dpmodel.utils.batch_size, 351
deepmd.dpmodel.utils.env_mat, 352
deepmd.dpmodel.utils.exclude_mask, 353
deepmd.dpmodel.utils.neighbor_stat, 354
deepmd.dpmodel.utils.network, 355
deepmd.dpmodel.utils.nlist, 359
deepmd.dpmodel.utils.region, 361
deepmd.dpmodel.utils.serialization, 363
deepmd.dpmodel.utils.type_embed, 364
deepmd.dpmodel.utils.update_sel, 365
deepmd.driver, 1063
deepmd.entrypoints, 382
deepmd.entrypoints.convert_backend, 382
deepmd.entrypoints.doc, 383
deepmd.entrypoints.gui, 383

deepmd.entrypoints.main, 384
deepmd.entrypoints.neighbor_stat, 384
deepmd.entrypoints.test, 386
deepmd.env, 1063
deepmd.infer, 387
deepmd.infer.deep_dipole, 387
deepmd.infer.deep_dos, 388
deepmd.infer.deep_eval, 389
deepmd.infer.deep_polar, 395
deepmd.infer.deep_pot, 397
deepmd.infer.deep_tensor, 399
deepmd.infer.deep_wfc, 401
deepmd.infer.model_devi, 402
deepmd.loggers, 411
deepmd.loggers.loggers, 412
deepmd.loggers.training, 413
deepmd.main, 1064
deepmd.pt, 414
deepmd.pt.cxx_op, 639
deepmd.pt.entrypoints, 414
deepmd.pt.entrypoints.main, 414
deepmd.pt.infer, 416
deepmd.pt.infer.deep_eval, 416
deepmd.pt.infer.inference, 419
deepmd.pt.loss, 419
deepmd.pt.loss.denoise, 419
deepmd.pt.loss.dos, 420
deepmd.pt.loss.ener, 422
deepmd.pt.loss.ener_spin, 423
deepmd.pt.loss.loss, 425
deepmd.pt.loss.tensor, 426
deepmd.pt.model, 434
deepmd.pt.model.atomic_model, 434
deepmd.pt.model.atomic_model.base_atomic_model, 434
deepmd.pt.model.atomic_model.dipole_atomic_model, 438
deepmd.pt.model.atomic_model.dos_atomic_model, 439
deepmd.pt.model.atomic_model.dp_atomic_model, 439
deepmd.pt.model.atomic_model.energy_atomic_model, 442
deepmd.pt.model.atomic_model.linear_atomic_model, 442
deepmd.pt.model.atomic_model.pairtab_atomic_model, 446
deepmd.pt.model.atomic_model.polar_atomic_model, 450
deepmd.pt.model.backbone, 464
deepmd.pt.model.backbone.backbone, 464
deepmd.pt.model.backbone.evoformer2b, 465
deepmd.pt.model.descriptor, 468
deepmd.pt.model.descriptor.base_descriptor, 468
deepmd.pt.model.descriptor.descriptor, 468
deepmd.pt.model.descriptor.dpa1, 470
deepmd.pt.model.descriptor.dpa2, 475
deepmd.pt.model.descriptor.env_mat, 479
deepmd.pt.model.descriptor.gaussian_lcc, 479
deepmd.pt.model.descriptor.hybrid, 481
deepmd.pt.model.descriptor.repformer_layer, 483
deepmd.pt.model.descriptor.repformers, 489
deepmd.pt.model.descriptor.se_a, 492
deepmd.pt.model.descriptor.se_atten, 496
deepmd.pt.model.descriptor.se_r, 502
deepmd.pt.model.model, 528
deepmd.pt.model.model.dipole_model, 529
deepmd.pt.model.model.dos_model, 529
deepmd.pt.model.model.dp_model, 530
deepmd.pt.model.model.dp_zbl_model, 531
deepmd.pt.model.model.ener_model, 532
deepmd.pt.model.model.frozen, 533
deepmd.pt.model.model.make_hessian_model, 534
deepmd.pt.model.model.make_model, 535
deepmd.pt.model.model.model, 535
deepmd.pt.model.model.polar_model, 537
deepmd.pt.model.model.spin_model, 537
deepmd.pt.model.model.transform_output, 540
deepmd.pt.model.network, 548
deepmd.pt.model.network.layernorm, 548
deepmd.pt.model.network.mlp, 550
deepmd.pt.model.network.network, 552
deepmd.pt.model.task, 578
deepmd.pt.model.task.atten_lcc, 578
deepmd.pt.model.task.base_fitting, 579
deepmd.pt.model.task.denoise, 579
deepmd.pt.model.task.dipole, 580
deepmd.pt.model.task.dos, 582
deepmd.pt.model.task.ener, 584
deepmd.pt.model.task.fitting, 588
deepmd.pt.model.task.invar_fitting, 591
deepmd.pt.model.task.polarizability, 593
deepmd.pt.model.task.task, 595
deepmd.pt.model.task.type_predict, 595
deepmd.pt.optimizer, 606
deepmd.pt.optimizer.KFWrapper, 606
deepmd.pt.optimizer.LKF, 606
deepmd.pt.train, 609
deepmd.pt.train.training, 609
deepmd.pt.train.wrapper, 610

deepmd.pt.utils, 611
deepmd.pt.utils.ase_calc, 611
deepmd.pt.utils.auto_batch_size, 613
deepmd.pt.utils.cache, 614
deepmd.pt.utils.dataloader, 614
deepmd.pt.utils.dataset, 616
deepmd.pt.utils.dp_random, 617
deepmd.pt.utils.env, 618
deepmd.pt.utils.env_mat_stat, 618
deepmd.pt.utils.exclude_mask, 620
deepmd.pt.utils.finetune, 621
deepmd.pt.utils.learning_rate, 622
deepmd.pt.utils.multi_task, 623
deepmd.pt.utils.neighbor_stat, 624
deepmd.pt.utils.nlist, 626
deepmd.pt.utils.plugin, 628
deepmd.pt.utils.preprocess, 630
deepmd.pt.utils.region, 631
deepmd.pt.utils.serialization, 633
deepmd.pt.utils.stat, 634
deepmd.pt.utils.update_sel, 636
deepmd.pt.utils.utils, 636
deepmd.tf, 639
deepmd.tf.__about__, 994
deepmd.tf.__main__, 994
deepmd.tf.calculator, 994
deepmd.tf.cluster, 639
deepmd.tf.cluster.local, 639
deepmd.tf.common, 996
deepmd.tf.descriptor, 640
deepmd.tf.descriptor.descriptor, 640
deepmd.tf.descriptor.hybrid, 647
deepmd.tf.descriptor.loc_frame, 653
deepmd.tf.descriptor.se, 656
deepmd.tf.descriptor.se_a, 659
deepmd.tf.descriptor.se_a_ebd, 665
deepmd.tf.descriptor.se_a_ebd_v2, 667
deepmd.tf.descriptor.se_a_ef, 668
deepmd.tf.descriptor.se_a_mask, 673
deepmd.tf.descriptor.se_atten, 676
deepmd.tf.descriptor.se_atten_v2, 688
deepmd.tf.descriptor.se_r, 689
deepmd.tf.descriptor.se_t, 694
deepmd.tf.entrypoints, 746
deepmd.tf.entrypoints.compress, 746
deepmd.tf.entrypoints.convert, 747
deepmd.tf.entrypoints.doc, 747
deepmd.tf.entrypoints.freeze, 747
deepmd.tf.entrypoints.gui, 748
deepmd.tf.entrypoints.ipi, 749
deepmd.tf.entrypoints.main, 750
deepmd.tf.entrypoints.neighbor_stat, 751
deepmd.tf.entrypoints.test, 752
deepmd.tf.entrypoints.train, 753
deepmd.tf.entrypoints.transfer, 754
deepmd.tf.env, 1001
deepmd.tf.env.op_grads_module, 1122
deepmd.tf.env.op_module, 1069
deepmd.tf.fit, 760
deepmd.tf.fit.dipole, 760
deepmd.tf.fit.dos, 763
deepmd.tf.fit.ener, 766
deepmd.tf.fit.fitting, 771
deepmd.tf.fit.polar, 773
deepmd.tf.infer, 794
deepmd.tf.infer.data_modifier, 794
deepmd.tf.infer.deep_dipole, 795
deepmd.tf.infer.deep_dos, 796
deepmd.tf.infer.deep_eval, 798
deepmd.tf.infer.deep_polar, 808
deepmd.tf.infer.deep_pot, 810
deepmd.tf.infer.deep_tensor, 812
deepmd.tf.infer.deep_wfc, 814
deepmd.tf.infer.ewald_recp, 815
deepmd.tf.infer.model_devi, 816
deepmd.tf.lmp, 1004
deepmd.tf.loggers, 835
deepmd.tf.loggers.loggers, 835
deepmd.tf.loss, 837
deepmd.tf.loss.dos, 837
deepmd.tf.loss.ener, 838
deepmd.tf.loss.loss, 842
deepmd.tf.loss.tensor, 844
deepmd.tf.model, 850
deepmd.tf.model.dos, 850
deepmd.tf.model.ener, 853
deepmd.tf.model.frozen, 856
deepmd.tf.model.linear, 858
deepmd.tf.model.model, 860
deepmd.tf.model.model_stat, 868
deepmd.tf.model.multi, 869
deepmd.tf.model.pairtab, 872
deepmd.tf.model.pairwise_dprc, 874
deepmd.tf.model.tensor, 877
deepmd.tf.nvnmd, 898
deepmd.tf.nvnmd.data, 898
deepmd.tf.nvnmd.data.data, 898
deepmd.tf.nvnmd.descriptor, 899
deepmd.tf.nvnmd.descriptor.se_a, 899
deepmd.tf.nvnmd.descriptor.se_atten, 900
deepmd.tf.nvnmd.entrypoints, 901
deepmd.tf.nvnmd.entrypoints.freeze, 901
deepmd.tf.nvnmd.entrypoints.mapt, 902
deepmd.tf.nvnmd.entrypoints.train, 904
deepmd.tf.nvnmd.entrypoints.wrap, 905
deepmd.tf.nvnmd.fit, 910
deepmd.tf.nvnmd.fit.ener, 910
deepmd.tf.nvnmd.utils, 910

deepmd.tf.nvnmd.utils.argcheck, 910
deepmd.tf.nvnmd.utils.config, 911
deepmd.tf.nvnmd.utils.encode, 913
deepmd.tf.nvnmd.utils.fio, 915
deepmd.tf.nvnmd.utils.network, 916
deepmd.tf.nvnmd.utils.op, 918
deepmd.tf.nvnmd.utils.weight, 918
deepmd.tf.op, 923
deepmd.tf.op._add_flt_nvnmd_grad, 924
deepmd.tf.op._copy_flt_nvnmd_grad, 924
deepmd.tf.op._dotmul_flt_nvnmd_grad, 924
deepmd.tf.op._flt_nvnmd_grad, 925
deepmd.tf.op._gelu, 925
deepmd.tf.op._map_flt_nvnmd_grad, 926
deepmd.tf.op._matmul_fitnet_nvnmd_grad, 926
deepmd.tf.op._matmul_flt2fix_nvnmd, 926
deepmd.tf.op._matmul_flt_nvnmd_grad, 926
deepmd.tf.op._mul_flt_nvnmd_grad, 927
deepmd.tf.op._prod_force_grad, 927
deepmd.tf.op._prod_force_se_a_grad, 927
deepmd.tf.op._prod_force_se_a_mask_grad, 928
deepmd.tf.op._prod_force_se_r_grad, 928
deepmd.tf.op._prod_virial_grad, 928
deepmd.tf.op._prod_virial_se_a_grad, 929
deepmd.tf.op._prod_virial_se_r_grad, 929
deepmd.tf.op._quantize_nvnmd_grad, 929
deepmd.tf.op._soft_min_force_grad, 930
deepmd.tf.op._soft_min_virial_grad, 930
deepmd.tf.op._tabulate_grad, 930
deepmd.tf.op._tanh4_flt_nvnmd_grad, 931
deepmd.tf.train, 932
deepmd.tf.train.run_options, 932
deepmd.tf.train.trainer, 934
deepmd.tf.utils, 936
deepmd.tf.utils.argcheck, 936
deepmd.tf.utils.batch_size, 937
deepmd.tf.utils.compat, 938
deepmd.tf.utils.compress, 939
deepmd.tf.utils.convert, 940
deepmd.tf.utils.data, 943
deepmd.tf.utils.data_system, 947
deepmd.tf.utils.errors, 950
deepmd.tf.utils.finetune, 951
deepmd.tf.utils.graph, 952
deepmd.tf.utils.learning_rate, 958
deepmd.tf.utils.multi_init, 959
deepmd.tf.utils.neighbor_stat, 960
deepmd.tf.utils.network, 962
deepmd.tf.utils.nlist, 965
deepmd.tf.utils.pair_tab, 965
deepmd.tf.utils.parallel_op, 967
deepmd.tf.utils.path, 968
deepmd.tf.utils.plugin, 974
deepmd.tf.utils.random, 975
deepmd.tf.utils.region, 977
deepmd.tf.utils.serialization, 977
deepmd.tf.utils.sess, 978
deepmd.tf.utils.spin, 978
deepmd.tf.utils.tabulate, 979
deepmd.tf.utils.type_embed, 981
deepmd.tf.utils.update_sel, 984
deepmd.tf.utils.weight_avg, 984
deepmd.utils, 1013
deepmd.utils.argcheck, 1013
deepmd.utils.argcheck_nvnmd, 1019
deepmd.utils.batch_size, 1019
deepmd.utils.compat, 1022
deepmd.utils.data, 1025
deepmd.utils.data_system, 1029
deepmd.utils.econf_embd, 1034
deepmd.utils.env_mat_stat, 1036
deepmd.utils.errors, 1038
deepmd.utils.finetune, 1038
deepmd.utils.hostlist, 1039
deepmd.utils.model_stat, 1040
deepmd.utils.neighbor_stat, 1041
deepmd.utils.out_stat, 1042
deepmd.utils.pair_tab, 1043
deepmd.utils.path, 1045
deepmd.utils.plugin, 1050
deepmd.utils.random, 1052
deepmd.utils.spin, 1054
deepmd.utils.summary, 1055
deepmd.utils.update_sel, 1056
deepmd.utils.version, 1057
deepmd.utils.weight_avg, 1057
module_name (in module deepmd.backend), 260
mul_flt_nvnmd (C++ function), 1318
mul_flt_nvnmd() (in module deepmd.tf.env.op_module), 1104
MulFltNvnmd() (in module deepmd.tf.env.op_module), 1079
multi_model_args() (in module deepmd.utils.argcheck), 1018
MultiModel (class in deepmd.tf.model), 893
MultiModel (class in deepmd.tf.model.multi), 869
my_device (deepmd.tf.train.run_options.RunOptions attribute), 933
my_rank (deepmd.tf.train.run_options.RunOptions attribute), 933
mydev (in module deepmd.pt.model.descriptor.repformers), 489
mydtype (in module deepmd.pt.model.descriptor.repformers), 489
mylinear (in module deepmd.pt.model.descriptor.repformers),

490

N

name (deepmd.backend.backend.Backend attribute), 255
name (deepmd.backend.dpmodel.DPModelBackend attribute), 256
name (deepmd.backend.pytorch.PyTorchBackend attribute), 258
name (deepmd.backend.tensorflow.TensorFlowBackend attribute), 260
name (deepmd.calculator.DP attribute), 1059
name (deepmd.pt.utils.ase_calc.DPCalculator attribute), 612
name (deepmd.tf.calculator.DP attribute), 995
name (deepmd.tf.utils.path.DPH5Path property), 969
name (deepmd.tf.utils.path.DPOSPath property), 971
name (deepmd.tf.utils.path.DPPath property), 972
name (deepmd.utils.path.DPH5Path property), 1049
name (deepmd.utils.path.DPOSPath property), 1047
name (deepmd.utils.path.DPPath property), 1045
NativeLayer (class in deepmd.dpmodel.utils), 368
NativeLayer (class in deepmd.dpmodel.utils.network), 356
NativeNet (in module deepmd.dpmodel.utils), 369
NativeNet (in module deepmd.dpmodel.utils.network), 358
NativeOP (class in deepmd.dpmodel), 380
NativeOP (class in deepmd.dpmodel.common), 374
natoms_match() (deepmd.tf.model.ener.EnerModel method), 855
natoms_match() (deepmd.tf.model.EnerModel method), 893
natoms_not_match() (deepmd.tf.model.ener.EnerModel method), 855
natoms_not_match() (deepmd.tf.model.EnerModel method), 893
NBIT_CUTF (C macro), 1324
NBIT_FLTF (C macro), 1324
nborAssert (C++ function), 1318, 1319
nborErrcheck (C macro), 1325
ndescrpt (deepmd.pt.model.descriptor.DescrptBlockSeA attribute), 522
ndescrpt (deepmd.pt.model.descriptor.se_a.DescrptBlockSeA attribute), 494
neighbor_stat (deepmd.backend.backend.Backend property), 254
NEIGHBOR_STAT (deepmd.backend.backend.Backend.Feature attribute), 254
neighbor_stat (deepmd.backend.dpmodel.DPModelBackend property), 256
neighbor_stat (deepmd.backend.pytorch.PyTorchBackend property), 257
neighbor_stat (deepmd.backend.tensorflow.TensorFlowBackend property), 259
neighbor_stat (deepmd.dpmodel.utils.update_sel.UpdateSel property), 366
neighbor_stat (deepmd.pt.utils.update_sel.UpdateSel property), 636
neighbor_stat (deepmd.tf.utils.update_sel.UpdateSel property), 984
neighbor_stat (deepmd.utils.update_sel.BaseUpdateSel property), 1056
neighbor_stat() (in module deepmd.entrypoints.neighbor_stat), 385
neighbor_stat() (in module deepmd.tf.entrypoints), 757
neighbor_stat() (in module deepmd.tf.entrypoints.neighbor_stat), 751
neighbor_stat() (in module deepmd.tf.env.op_module), 1105
NeighborGatedAttention (class in deepmd.dpmodel.descriptor.dpa1), 286
NeighborGatedAttention (class in deepmd.pt.model.descriptor.se_atten), 498
NeighborGatedAttentionLayer (class in deepmd.dpmodel.descriptor.dpa1), 287
NeighborGatedAttentionLayer (class in deepmd.pt.model.descriptor.se_atten), 500
NeighborStat (class in deepmd.dpmodel.utils.neighbor_stat), 355
NeighborStat (class in deepmd.pt.utils.neighbor_stat), 625
NeighborStat (class in deepmd.tf.utils.neighbor_stat), 961
NeighborStat (class in deepmd.utils.neighbor_stat), 1041
NeighborStat() (in module deepmd.tf.env.op_module), 1079
NeighborStatOP (class in deepmd.dpmodel.utils.neighbor_stat), 354
NeighborStatOP (class in deepmd.pt.utils.neighbor_stat), 624
NeighborStatOP (class in deepmd.tf.utils.neighbor_stat), 960
NeighborWiseAttention (class in deepmd.pt.model.network.network), 566
NeighborWiseAttentionLayer (class in deepmd.pt.model.network.network), 567
net_size:
    nvnmd/net_size (Argument), 160
NETWORK_TYPE_MAP (deepmd.dpmodel.utils.network.NetworkCollectio

attribute), 359
NETWORK_TYPE_MAP (deepmd.dpmodel.utils.NetworkCollection attribute), 370
NETWORK_TYPE_MAP (deepmd.pt.model.network.mlp.NetworkCollection attribute), 552
NetworkCollection (class in deepmd.dpmodel.utils), 369
NetworkCollection (class in deepmd.dpmodel.utils.network), 358
NetworkCollection (class in deepmd.pt.model.network.mlp), 552
neuron:
    model/type_embedding/neuron (Argument), 112
    model[standard]/descriptor[se_a_ebd_v2]/neuron (Argument), 136
    model[standard]/descriptor[se_a_mask]/neuron (Argument), 138
    model[standard]/descriptor[se_a_tpe]/neuron (Argument), 121
    model[standard]/descriptor[se_atten_v2]/neuron (Argument), 130
    model[standard]/descriptor[se_atten]/neuron (Argument), 125
    model[standard]/descriptor[se_e2_a]/neuron (Argument), 117
    model[standard]/descriptor[se_e2_r]/neuron (Argument), 123
    model[standard]/descriptor[se_e3]/neuron (Argument), 119
    model[standard]/fitting_net[dipole]/neuron (Argument), 144
    model[standard]/fitting_net[dos]/neuron (Argument), 142
    model[standard]/fitting_net[ener]/neuron (Argument), 140
    model[standard]/fitting_net[polar]/neuron (Argument), 143
nlist_distinguish_types() (in module deepmd.dpmodel.utils), 372
nlist_distinguish_types() (in module deepmd.dpmodel.utils.nlist), 360
nlist_distinguish_types() (in module deepmd.pt.utils.nlist), 627
nodelist (deepmd.tf.train.run_options.RunOptions attribute), 933
nodename (deepmd.tf.train.run_options.RunOptions attribute), 933
NodeTaskHead (class in deepmd.pt.model.network.network), 570
NonLinear (class in deepmd.pt.model.network.network), 561
NonLinearHead (class in deepmd.pt.model.network.network), 561
norm_expo() (deepmd.tf.nvnmd.utils.Encode method), 921
norm_expo() (deepmd.tf.nvnmd.utils.encode.Encode method), 913
normalize() (in module deepmd.tf.utils.argcheck), 937
normalize() (in module deepmd.utils.argcheck), 1019
normalize:
    model[standard]/descriptor[se_atten]/normalize (Argument), 128
normalize_coord() (in module deepmd.dpmodel.utils), 372
normalize_coord() (in module deepmd.dpmodel.utils.region), 362
normalize_coord() (in module deepmd.pt.utils.preprocess), 630
normalize_coord() (in module deepmd.pt.utils.region), 632
normalize_data_dict() (in module deepmd.utils.argcheck), 1019
normalize_fitting_net_dict() (in module deepmd.utils.argcheck), 1019
normalize_fitting_weight() (in module deepmd.utils.argcheck), 1019
normalize_learning_rate_dict() (in module deepmd.utils.argcheck), 1019
normalize_learning_rate_dict_with_single_learning_rate() (in module deepmd.utils.argcheck), 1019
normalize_loss_dict() (in module deepmd.utils.argcheck), 1019
normalize_multi_task() (in module deepmd.utils.argcheck), 1019
normalized_input() (in module deepmd.tf.nvnmd.entrypoints.train), 904
normalized_input_qnn() (in module deepmd.tf.nvnmd.entrypoints.train), 904
NOT_LOADABLE (in module deepmd.backend), 260
NOT_LOADABLE (in module deepmd.tf.op), 932
np_normalize() (in module deepmd.dpmodel.descriptor.dpa1), 281
np_softmax() (in module deepmd.dpmodel.descriptor.dpa1), 281
NUM_WORKERS (in module deepmd.pt.utils.env), 618
numb_aparam:
    model[standard]/descriptor[se_a_tpe]/numb_aparam (Argument), 123
    model[standard]/fitting_net[dos]/numb_aparam (Argument), 142
    model[standard]/fitting_net[ener]/numb_aparam (Argument), 140
numb_btch:

training/validation_data/numb_btch (Argument), 156
numb_dos (deepmd.infer.deep_dos.DeepDOS property), 388
numb_dos (deepmd.tf.infer.deep_dos.DeepDOS property), 797
numb_dos (deepmd.tf.infer.DeepDOS property), 823
numb_dos:
model[standard]/fitting_net[dos]/numb_dos (Argument), 143
numb_fparam:
model[standard]/fitting_net[dos]/numb_fparam (Argument), 142
model[standard]/fitting_net[ener]/numb_fparam (Argument), 140
numb_generalized_coord:
loss[ener]/numb_generalized_coord (Argument), 150
numb_steps:
training/numb_steps (Argument), 157
nvnmd (Argument)
nvnmd:, 159
nvnmd/config_file (Argument)
config_file:, 160
nvnmd/enable (Argument)
enable:, 160
nvnmd/map_file (Argument)
map_file:, 160
nvnmd/max_nnei (Argument)
max_nnei:, 159
nvnmd/net_size (Argument)
net_size:, 160
nvnmd/quantize_descriptor (Argument)
quantize_descriptor:, 160
nvnmd/quantize_fitting_net (Argument)
quantize_fitting_net:, 160
nvnmd/restore_descriptor (Argument)
restore_descriptor:, 160
nvnmd/restore_fitting_net (Argument)
restore_fitting_net:, 160
nvnmd/version (Argument)
version:, 159
nvnmd/weight_file (Argument)
weight_file:, 160
nvnmd:
nvnmd (Argument), 159
nvnmd_args() (in module deepmd.tf.nvnmd.utils), 920
nvnmd_args() (in module deepmd.tf.nvnmd.utils.argcheck), 911
nvnmd_args() (in module deepmd.utils.argcheck_nvnmd), 1019
nvnmd_cfg (in module deepmd.tf.nvnmd.fit.ener), 910
nvnmd_cfg (in module deepmd.tf.nvnmd.utils), 920
nvnmd_cfg (in module deepmd.tf.nvnmd.utils.config), 913
NVNMD_CITATION (in module deepmd.tf.nvnmd.data.data), 898
NVNMD_WELCOME (in module deepmd.tf.nvnmd.data.data), 898
NvnmdConfig (class in deepmd.tf.nvnmd.utils.config), 911

O

omp_get_num_threads (C++ function), 1319
omp_get_thread_num (C++ function), 1319
one_layer() (in module deepmd.tf.nvnmd.utils), 923
one_layer() (in module deepmd.tf.nvnmd.utils.network), 917
one_layer() (in module deepmd.tf.utils.network), 962
one_layer_nvnmd() (in module deepmd.tf.nvnmd.fit.ener), 910
one_layer_rand_seed_shift() (in module deepmd.tf.utils.network), 962
one_layer_t() (in module deepmd.tf.nvnmd.utils.network), 917
one_layer_wb() (in module deepmd.tf.nvnmd.utils.network), 917
op_dir (in module deepmd.tf.lmp), 1005
op_grads_module (in module deepmd.tf.env), 1003
op_module (in module deepmd.tf.env), 1003
opt_type:
training/opt_type (Argument), 159
OUT (deepmd.dpmodel.output_def.OutputVariableCategory attribute), 376
OuterProduct (class in deepmd.pt.model.network.network), 571
OutOfMemoryError, 950, 1038
output_def (deepmd.infer.deep_dos.DeepDOS property), 388
output_def (deepmd.infer.deep_eval.DeepEval property), 393
output_def (deepmd.infer.deep_pot.DeepPot property), 397
output_def (deepmd.infer.deep_tensor.DeepTensor property), 399
output_def (deepmd.infer.DeepEval property), 406
output_def (deepmd.infer.DeepPot property), 409
output_def (deepmd.tf.infer.deep_dos.DeepDOS property), 797
output_def (deepmd.tf.infer.deep_pot.DeepPot property), 810
output_def (deepmd.tf.infer.DeepDOS property), 822
output_def (deepmd.tf.infer.DeepPot property), 832

output_def() (deepmd.dpmodel.fitting.dipole_fitting.DipoleFitting method), 312
output_def() (deepmd.dpmodel.fitting.DipoleFitting method), 329
output_def() (deepmd.dpmodel.fitting.invar_fitting.InvarFitting method), 323
output_def() (deepmd.dpmodel.fitting.InvarFitting method), 335
output_def() (deepmd.dpmodel.fitting.PolarFitting method), 338
output_def() (deepmd.dpmodel.fitting.polarizability_fitting.PolarFitting method), 326
output_def() (deepmd.pt.model.task.denoise.DenoiseNet method), 580
output_def() (deepmd.pt.model.task.DenoiseNet method), 598
output_def() (deepmd.pt.model.task.dipole.DipoleFittingNet method), 582
output_def() (deepmd.pt.model.task.DipoleFittingNet method), 599
output_def() (deepmd.pt.model.task.dos.DOSFittingNet method), 584
output_def() (deepmd.pt.model.task.ener.EnergyFittingNetDirect method), 587
output_def() (deepmd.pt.model.task.EnergyFittingNetDirect method), 602
output_def() (deepmd.pt.model.task.invar_fitting.InvarFitting method), 593
output_def() (deepmd.pt.model.task.PolarFittingNet method), 605
output_def() (deepmd.pt.model.task.polarizability.PolarFittingNet method), 595
output_def_mag (deepmd.infer.deep_pot.DeepPot property), 397
output_def_mag (deepmd.infer.DeepPot property), 409
output_def_mag (deepmd.tf.infer.deep_pot.DeepPot property), 810
output_def_mag (deepmd.tf.infer.DeepPot property), 832
output_prec:
    training/mixed_precision/output_prec (Argument), 156
output_tensor_name (deepmd.infer.deep_dipole.DeepDipole property), 387
output_tensor_name (deepmd.infer.deep_polar.DeepGlobalPolar property), 396
output_tensor_name (deepmd.infer.deep_polar.DeepPolar property), 395
output_tensor_name (deepmd.infer.deep_tensor.DeepTensor property), 399
output_tensor_name (deepmd.infer.deep_wfc.DeepWFC property), 401
output_tensor_name (deepmd.tf.infer.deep_dipole.DeepDipole property), 796
output_tensor_name (deepmd.tf.infer.deep_polar.DeepGlobalPolar property), 808
output_tensor_name (deepmd.tf.infer.deep_polar.DeepPolar property), 809
output_tensor_name (deepmd.tf.infer.deep_wfc.DeepWFC property), 815
output_tensor_name (deepmd.tf.infer.DeepDipole property), 822
output_tensor_name (deepmd.tf.infer.DeepGlobalPolar property), 830
output_tensor_name (deepmd.tf.infer.DeepPolar property), 831
output_tensor_name (deepmd.tf.infer.DeepWFC property), 833
OutputVariableCategory (class in deepmd.dpmodel.output_def), 376
OutputVariableDef (class in deepmd.dpmodel), 381
OutputVariableDef (class in deepmd.dpmodel.output_def), 376
OutputVariableOperation (class in deepmd.dpmodel.output_def), 376

P

PACKAGE_BASE (in module deepmd.backend), 260
PACKAGE_BASE (in module deepmd.tf.op), 932
pair_exclude_types:
    model/pair_exclude_types (Argument), 112
pair_tab() (in module deepmd.tf.env.op_module), 1105
PairExcludeMask (class in deepmd.dpmodel.utils), 368
PairExcludeMask (class in deepmd.dpmodel.utils.exclude_mask), 353
PairExcludeMask (class in deepmd.pt.utils), 638
PairExcludeMask (class in deepmd.pt.utils.exclude_mask), 620
PairTab (class in deepmd.tf.utils), 992
PairTab (class in deepmd.tf.utils.pair_tab), 966
PairTab (class in deepmd.utils.pair_tab), 1043
PairTab() (in module deepmd.tf.env.op_module), 1079
pairtab_model_args() (in module deepmd.utils.argcheck), 1018
PairTabAtomicModel (class in deepmd.dpmodel.atomic_model), 277
PairTabAtomicModel (class in deepmd.dpmodel.atomic_model.pairtab_atomic_model), 268
PairTabAtomicModel (class in deepmd.pt.model.atomic_model), 461
PairTabAtomicModel (class in deepmd.pt.model.atomic_model.pairtab_atomic_model),

446
PairTabModel (class in deepmd.tf.model), 884
PairTabModel (class in deepmd.tf.model.pairtab), 872
pairwise_dprc() (in module deepmd.utils.argcheck), 1018
PairwiseDPRc (class in deepmd.tf.model), 886
PairwiseDPRc (class in deepmd.tf.model.pairwise_dprc), 874
parallel_prod_force_se_a() (in module deepmd.tf.env.op_module), 1105
ParallelOp (class in deepmd.tf.utils.parallel_op), 967
ParallelProdForceSeA() (in module deepmd.tf.env.op_module), 1080
parse_args() (in module deepmd.main), 1066
parse_args() (in module deepmd.tf.entrypoints.main), 750
parse_auto_sel() (deepmd.utils.update_sel.BaseUpdateSel method), 1056
parse_auto_sel_ratio() (deepmd.utils.update_sel.BaseUpdateSel method), 1056
pass_tensors_from_frz_model() (deepmd.tf.descriptor.Descriptor method), 703
pass_tensors_from_frz_model() (deepmd.tf.descriptor.descriptor.Descriptor method), 645
pass_tensors_from_frz_model() (deepmd.tf.descriptor.DescrptHybrid method), 709
pass_tensors_from_frz_model() (deepmd.tf.descriptor.hybrid.DescrptHybrid method), 652
pass_tensors_from_frz_model() (deepmd.tf.descriptor.se.DescrptSe method), 657
phys2inter() (deepmd.pt.utils.preprocess.Region3D method), 630
phys2inter() (in module deepmd.dpmodel.utils), 372
phys2inter() (in module deepmd.dpmodel.utils.region), 361
phys2inter() (in module deepmd.pt.utils.region), 631
plot_lines() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 903
plot_lines() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
Plugin (class in deepmd.pt.utils.plugin), 628
Plugin (class in deepmd.tf.utils), 993
Plugin (class in deepmd.tf.utils.plugin), 974
Plugin (class in deepmd.utils.plugin), 1051
plugin (in module deepmd.backend), 260
PluginVariant (class in deepmd.pt.utils.plugin), 629
PluginVariant (class in deepmd.tf.utils), 994
PluginVariant (class in deepmd.tf.utils.plugin), 975
PluginVariant (class in deepmd.utils.plugin), 1052
PolarFitting (class in deepmd.dpmodel.fitting), 336
PolarFitting (class in deepmd.dpmodel.fitting.polarizability_fitting), 324
PolarFittingNet (class in deepmd.pt.model.task), 603
PolarFittingNet (class in deepmd.pt.model.task.polarizability), 593
PolarFittingSeA (class in deepmd.tf.fit), 791
PolarFittingSeA (class in deepmd.tf.fit.polar), 773
PolarModel (class in deepmd.pt.model.model.polar_model), 537
PolarModel (class in deepmd.tf.model), 897
PolarModel (class in deepmd.tf.model.tensor), 880
precision (deepmd.tf.descriptor.se.DescrptSe property), 656
precision (deepmd.tf.fit.Fitting property), 787
precision (deepmd.tf.fit.fitting.Fitting property), 771
precision:
  model/type_embedding/precision (Argument), 113
  model[standard]/descriptor[se_a_ebd_v2]/precision (Argument), 137
  model[standard]/descriptor[se_a_mask]/precision (Argument), 139
  model[standard]/descriptor[se_a_tpe]/precision (Argument), 122
  model[standard]/descriptor[se_atten_v2]/precision (Argument), 130
  model[standard]/descriptor[se_atten]/precision (Argument), 126
  model[standard]/descriptor[se_e2_a]/precision (Argument), 118
  model[standard]/descriptor[se_e2_r]/precision (Argument), 124
  model[standard]/descriptor[se_e3]/precision (Argument), 120
  model[standard]/fitting_net[dipole]/precision (Argument), 145
  model[standard]/fitting_net[dos]/precision (Argument), 142
  model[standard]/fitting_net[ener]/precision (Argument), 141
  model[standard]/fitting_net[polar]/precision (Argument), 144
PRECISION_DICT (in module deepmd.dpmodel), 380

PRECISION_DICT (in module deepmd.dpmodel.common), 374
PRECISION_DICT (in module deepmd.pt.utils.env), 618
PRECISION_DICT (in module deepmd.tf.common), 999
PRECISION_DICT (in module deepmd.utils.argcheck), 1016
pref:
    loss[tensor]/pref (Argument), 154
pref_atomic:
    loss[tensor]/pref_atomic (Argument), 154
preload_env (in module deepmd.tf.lmp), 1004
preprocess_shared_params() (in module deepmd.pt.utils.multi_task), 623
print_econf_embedding() (in module deepmd.utils.econf_embd), 1035
print_header() (deepmd.pt.train.training.Trainer method), 609
print_header() (deepmd.tf.loss.ener.EnerSpinLoss method), 841
print_header() (deepmd.tf.loss.EnerSpinLoss method), 847
print_header() (deepmd.tf.train.trainer.DPTrainer static method), 935
print_on_training() (deepmd.pt.train.training.Trainer method), 609
print_on_training() (deepmd.tf.loss.ener.EnerSpinLoss method), 841
print_on_training() (deepmd.tf.loss.EnerSpinLoss method), 847
print_on_training() (deepmd.tf.train.trainer.DPTrainer static method), 935
print_resource_summary() (deepmd.tf.train.run_options.RunOptions method), 933
print_summary() (deepmd.pt.utils.dataloader.DpLoaderSet method), 615
print_summary() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 950
print_summary() (deepmd.tf.utils.DeepmdDataSystem method), 991
print_summary() (deepmd.utils.data_system.DeepmdDataSystem method), 1033
print_summary() (in module deepmd.utils.data_system), 1033
prob_sys_size_ext() (in module deepmd.tf.utils.data_system), 950
prob_sys_size_ext() (in module deepmd.utils.data_system), 1033
process_spin_input() (deepmd.dpmodel.model.spin_model.SpinModel method), 344
process_spin_input() (deepmd.dpmodel.model.SpinModel method), 348
process_spin_input() (deepmd.pt.model.model.spin_model.SpinModel method), 537
process_spin_input() (deepmd.pt.model.model.SpinModel method), 546
process_spin_input_lower() (deepmd.dpmodel.model.spin_model.SpinModel method), 344
process_spin_input_lower() (deepmd.dpmodel.model.SpinModel method), 348
process_spin_input_lower() (deepmd.pt.model.model.spin_model.SpinModel method), 538
process_spin_input_lower() (deepmd.pt.model.model.SpinModel method), 546
process_spin_output() (deepmd.dpmodel.model.spin_model.SpinModel method), 345
process_spin_output() (deepmd.dpmodel.model.SpinModel method), 349
process_spin_output() (deepmd.pt.model.model.spin_model.SpinModel method), 538
process_spin_output() (deepmd.pt.model.model.SpinModel method), 547
process_spin_output_lower() (deepmd.dpmodel.model.spin_model.SpinModel method), 345
process_spin_output_lower() (deepmd.dpmodel.model.SpinModel method), 349
process_spin_output_lower() (deepmd.pt.model.model.spin_model.SpinModel method), 538
process_spin_output_lower() (deepmd.pt.model.model.SpinModel method), 547
process_sys_probs() (in module deepmd.tf.utils.data_system), 950
process_sys_probs() (in module deepmd.utils.data_system), 1033
process_systems() (in module deepmd.utils.data_system), 1033
prod_env_mat() (in module

deepmd.pt.model.descriptor), 517
prod_env_mat() (in module deepmd.pt.model.descriptor.env_mat), 479
prod_env_mat_a() (in module deepmd.tf.env.op_module), 1106
prod_env_mat_a_mix() (in module deepmd.tf.env.op_module), 1107
prod_env_mat_a_mix_nvnmd_quantize() (in module deepmd.tf.env.op_module), 1108
prod_env_mat_a_nvnmd_quantize() (in module deepmd.tf.env.op_module), 1108
prod_env_mat_r() (in module deepmd.tf.env.op_module), 1109
prod_force() (in module deepmd.tf.env.op_module), 1110
prod_force_grad() (in module deepmd.tf.env.op_grads_module), 1126
prod_force_norot() (in module deepmd.tf.env.op_module), 1110
prod_force_se_a() (in module deepmd.tf.env.op_module), 1110
prod_force_se_a_grad() (in module deepmd.tf.env.op_grads_module), 1127
prod_force_se_a_mask() (in module deepmd.tf.env.op_module), 1111
prod_force_se_a_mask_grad() (in module deepmd.tf.env.op_grads_module), 1127
prod_force_se_r() (in module deepmd.tf.env.op_module), 1111
prod_force_se_r_grad() (in module deepmd.tf.env.op_grads_module), 1127
prod_force_virial() (deepmd.tf.descriptor.Descriptor method), 702
prod_force_virial() (deepmd.tf.descriptor.descriptor.Descriptor method), 644
prod_force_virial() (deepmd.tf.descriptor.DescrptHybrid method), 708
prod_force_virial() (deepmd.tf.descriptor.DescrptLocFrame method), 712
prod_force_virial() (deepmd.tf.descriptor.DescrptSeA method), 718
prod_force_virial() (deepmd.tf.descriptor.DescrptSeAEf method), 724
prod_force_virial() (deepmd.tf.descriptor.DescrptSeAMask method), 728
prod_force_virial() (deepmd.tf.descriptor.DescrptSeR method), 741
prod_force_virial() (deepmd.tf.descriptor.DescrptSeT method), 745
prod_force_virial() (deepmd.tf.descriptor.hybrid.DescrptHybrid method), 650
prod_force_virial() (deepmd.tf.descriptor.loc_frame.DescrptLocFrame method), 655
prod_force_virial() (deepmd.tf.descriptor.se_a.DescrptSeA method), 663
prod_force_virial() (deepmd.tf.descriptor.se_a_ef.DescrptSeAEf method), 671
prod_force_virial() (deepmd.tf.descriptor.se_a_mask.DescrptSeAMask method), 675
prod_force_virial() (deepmd.tf.descriptor.se_r.DescrptSeR method), 693
prod_force_virial() (deepmd.tf.descriptor.se_t.DescrptSeT method), 697
prod_virial() (in module deepmd.tf.env.op_module), 1111
prod_virial_grad() (in module deepmd.tf.env.op_grads_module), 1128
prod_virial_norot() (in module deepmd.tf.env.op_module), 1112
prod_virial_se_a() (in module deepmd.tf.env.op_module), 1112
prod_virial_se_a_grad() (in module deepmd.tf.env.op_grads_module), 1128
prod_virial_se_r() (in module deepmd.tf.env.op_module), 1113
prod_virial_se_r_grad() (in module deepmd.tf.env.op_grads_module), 1129
ProdEnvMatA() (in module deepmd.tf.env.op_module), 1080
ProdEnvMatAMix() (in module deepmd.tf.env.op_module), 1081
ProdEnvMatAMixNvnmdQuantize() (in module deepmd.tf.env.op_module), 1082
ProdEnvMatANvnmdQuantize() (in module deepmd.tf.env.op_module), 1083
ProdEnvMatR() (in module deepmd.tf.env.op_module), 1083
ProdForce() (in module deepmd.tf.env.op_module), 1084
ProdForceGrad() (in module deepmd.tf.env.op_grads_module), 1123

ProdForceNorot() (in module deepmd.tf.env.op_module), 1084
ProdForceSeA() (in module deepmd.tf.env.op_module), 1084
ProdForceSeAGPUExecuteFunctor (C++ struct), 1274
ProdForceSeAGPUExecuteFunctor::operator() (C++ function), 1275
ProdForceSeAGrad() (in module deepmd.tf.env.op_grads_module), 1123
ProdForceSeAMask() (in module deepmd.tf.env.op_module), 1085
ProdForceSeAMaskGrad() (in module deepmd.tf.env.op_grads_module), 1124
ProdForceSeR() (in module deepmd.tf.env.op_module), 1085
ProdForceSeRGPUExecuteFunctor (C++ struct), 1275
ProdForceSeRGPUExecuteFunctor::operator() (C++ function), 1275
ProdForceSeRGrad() (in module deepmd.tf.env.op_grads_module), 1124
ProdVirial() (in module deepmd.tf.env.op_module), 1085
ProdVirialGrad() (in module deepmd.tf.env.op_grads_module), 1124
ProdVirialNorot() (in module deepmd.tf.env.op_module), 1086
ProdVirialSeA() (in module deepmd.tf.env.op_module), 1086
ProdVirialSeAGPUExecuteFunctor (C++ struct), 1275
ProdVirialSeAGPUExecuteFunctor::operator() (C++ function), 1275
ProdVirialSeAGrad() (in module deepmd.tf.env.op_grads_module), 1125
ProdVirialSeR() (in module deepmd.tf.env.op_module), 1087
ProdVirialSeRGPUExecuteFunctor (C++ struct), 1276
ProdVirialSeRGPUExecuteFunctor::operator() (C++ function), 1276
ProdVirialSeRGrad() (in module deepmd.tf.env.op_grads_module), 1125
profiling:
    training/profiling (Argument), 157
profiling_file:
    training/profiling_file (Argument), 158
PyTorchBackend (class in deepmd.backend.pytorch), 257

Q

qc() (deepmd.tf.nvnmd.utils.Encode method), 921
qc() (deepmd.tf.nvnmd.utils.encode.Encode method), 913
qf() (deepmd.tf.nvnmd.utils.Encode method), 921
qf() (deepmd.tf.nvnmd.utils.encode.Encode method), 913
qf() (in module deepmd.tf.nvnmd.utils.network), 917
qm_model:
    model[pairwise_dprc]/qm_model (Argument), 146
qmmm_model:
    model[pairwise_dprc]/qmmm_model (Argument), 146
qr() (deepmd.tf.nvnmd.utils.Encode method), 921
qr() (deepmd.tf.nvnmd.utils.encode.Encode method), 913
qr() (in module deepmd.tf.nvnmd.utils.network), 917
quantize_descriptor:
    nvnmd/quantize_descriptor (Argument), 160
quantize_fitting_net:
    nvnmd/quantize_fitting_net (Argument), 160
quantize_nvnmd() (in module deepmd.tf.env.op_module), 1113
QuantizeNvnmd() (in module deepmd.tf.env.op_module), 1087
QUEUESIZE (in module deepmd.pt.utils.dataloader), 615

R

r2s() (in module deepmd.tf.nvnmd.utils.op), 918
random() (in module deepmd.pt.utils.dp_random), 617
random() (in module deepmd.tf.utils.random), 976
random() (in module deepmd.utils.random), 1053
RawTextArgumentDefaultsHelpFormatter (class in deepmd.main), 1065
rcond:
    model[standard]/fitting_net[dos]/rcond (Argument), 143
    model[standard]/fitting_net[ener]/rcond (Argument), 141
rcut:
    model[pairtab]/rcut (Argument), 146
    model[standard]/descriptor[loc_frame]/rcut (Argument), 116
    model[standard]/descriptor[se_a_ebd_v2]/rcut (Argument), 136
    model[standard]/descriptor[se_a_tpe]/rcut (Argument), 121
    model[standard]/descriptor[se_atten_v2]/rcut (Argument), 129
    model[standard]/descriptor[se_atten]/rcut (Argument), 125

model[standard]/descriptor[se_e2_a]/rcut (Argument), 117
model[standard]/descriptor[se_e2_r]/rcut (Argument), 123
model[standard]/descriptor[se_e3]/rcut (Argument), 119
rcut_smth:
model[standard]/descriptor[se_a_ebd_v2]/rcut_smth (Argument), 136
model[standard]/descriptor[se_a_tpe]/rcut_smth (Argument), 121
model[standard]/descriptor[se_atten_v2]/rcut_smth (Argument), 129
model[standard]/descriptor[se_atten]/rcut_smth (Argument), 125
model[standard]/descriptor[se_e2_a]/rcut_smth (Argument), 117
model[standard]/descriptor[se_e2_r]/rcut_smth (Argument), 123
model[standard]/descriptor[se_e3]/rcut_smth (Argument), 119
REDU (deepmd.dpmodel.output_def.OutputVariableCategory attribute), 376
REDU (deepmd.dpmodel.output_def.OutputVariableOperation attribute), 376
reduce() (deepmd.tf.utils.data.DeepmdData method), 945
reduce() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 948
reduce() (deepmd.tf.utils.DeepmdData method), 986
reduce() (deepmd.tf.utils.DeepmdDataSystem method), 989
reduce() (deepmd.utils.data.DeepmdData method), 1026
reduce() (deepmd.utils.data_system.DeepmdDataSystem method), 1031
reformat_data_torch() (deepmd.tf.utils.data.DeepmdData method), 946
reformat_data_torch() (deepmd.tf.utils.DeepmdData method), 988
reformat_data_torch() (deepmd.utils.data.DeepmdData method), 1028
Region3D (class in deepmd.pt.utils.preprocess), 630
register() (deepmd.pt.utils.plugin.Plugin method), 629
register() (deepmd.tf.utils.Plugin method), 994
register() (deepmd.tf.utils.plugin.Plugin method), 974
register() (deepmd.utils.argcheck.ArgsPlugin method), 1016
register() (deepmd.utils.plugin.Plugin method), 1051
reinit() (deepmd.pt.utils.AtomExcludeMask method), 638
reinit() (deepmd.pt.utils.exclude_mask.AtomExcludeMask method), 620
reinit() (deepmd.pt.utils.exclude_mask.PairExcludeMask method), 620
reinit() (deepmd.pt.utils.PairExcludeMask method), 638
reinit() (deepmd.tf.utils.pair_tab.PairTab method), 966
reinit() (deepmd.tf.utils.PairTab method), 992
reinit() (deepmd.utils.pair_tab.PairTab method), 1043
reinit_atom_exclude() (deepmd.dpmodel.atomic_model.base_atomic_model.BaseA method), 261
reinit_atom_exclude() (deepmd.dpmodel.atomic_model.BaseAtomicModel method), 272
reinit_atom_exclude() (deepmd.pt.model.atomic_model.base_atomic_model.BaseA method), 435
reinit_atom_exclude() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 452
reinit_exclude() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 301
reinit_exclude() (deepmd.dpmodel.descriptor.DescrptSeA method), 306
reinit_exclude() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 method), 285
reinit_exclude() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA method), 293
reinit_exclude() (deepmd.dpmodel.fitting.general_fitting.GeneralF method), 319
reinit_exclude() (deepmd.pt.model.descriptor.DescrptBlockRepfor method), 522
reinit_exclude() (deepmd.pt.model.descriptor.DescrptBlockSeA method), 523
reinit_exclude() (deepmd.pt.model.descriptor.DescrptBlockSeAtte method), 508
reinit_exclude() (deepmd.pt.model.descriptor.DescrptSeA method), 525
reinit_exclude() (deepmd.pt.model.descriptor.DescrptSeR method), 527
reinit_exclude() (deepmd.pt.model.descriptor.repformers.DescrptF method), 491
reinit_exclude() (deepmd.pt.model.descriptor.se_a.DescrptBlockSe method), 495
reinit_exclude() (deepmd.pt.model.descriptor.se_a.DescrptSeA method), 493
reinit_exclude() (deepmd.pt.model.descriptor.se_atten.DescrptBlo

method), 498
reinit_exclude() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 504
reinit_exclude() (deepmd.pt.model.task.fitting.GeneralFitting method), 590
reinit_pair_exclude() (deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicModel method), 261
reinit_pair_exclude() (deepmd.dpmodel.atomic_model.BaseAtomicModel method), 272
reinit_pair_exclude() (deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel method), 435
reinit_pair_exclude() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 452
relative_f:
    loss[ener_spin]/relative_f (Argument), 152
    loss[ener]/relative_f (Argument), 149
remap_atype() (deepmd.dpmodel.atomic_model.linear_atomic_model.LinearEnergyAtomicModel static method), 266
remap_atype() (deepmd.dpmodel.atomic_model.LinearEnergyAtomicModel static method), 276
remap_atype() (deepmd.pt.model.atomic_model.linear_atomic_model.LinearEnergyAtomicModel static method), 444
remap_atype() (deepmd.pt.model.atomic_model.LinearEnergyAtomicModel static method), 459
remove_decay_rate() (in module deepmd.utils.compat), 1024
REMOVE_SUFFIX_DICT (in module deepmd.tf.env), 1003
repformer_activation:
    model[standard]/descriptor[dpa2]/repformer_activation (Argument), 135
repformer_add_type_ebd_to_seq:
    model[standard]/descriptor[dpa2]/repformer_add_type_ebd_to_seq (Argument), 135
repformer_attn1_hidden:
    model[standard]/descriptor[dpa2]/repformer_attn1_hidden (Argument), 135
repformer_attn1_nhead:
    model[standard]/descriptor[dpa2]/repformer_attn1_nhead (Argument), 135
repformer_attn2_has_gate:
    model[standard]/descriptor[dpa2]/repformer_attn2_has_gate (Argument), 135
repformer_attn2_hidden:
    model[standard]/descriptor[dpa2]/repformer_attn2_hidden (Argument), 135
repformer_attn2_nhead:
    model[standard]/descriptor[dpa2]/repformer_attn2_nhead (Argument), 135
repformer_axis_dim:
    model[standard]/descriptor[dpa2]/repformer_axis_dim (Argument), 133
repformer_bn_momentum:
    model[standard]/descriptor[dpa2]/repformer_bn_momentum (Argument), 134
repformer_do_bn_mode:
    model[standard]/descriptor[dpa2]/repformer_do_bn_mode (Argument), 133
repformer_g1_dim:
    model[standard]/descriptor[dpa2]/repformer_g1_dim (Argument), 133
repformer_g2_dim:
    model[standard]/descriptor[dpa2]/repformer_g2_dim (Argument), 133
repformer_nlayers:
    model[standard]/descriptor[dpa2]/repformer_nlayers (Argument), 133
repformer_nsel:
    model[standard]/descriptor[dpa2]/repformer_nsel (Argument), 132
repformer_rcut:
    model[standard]/descriptor[dpa2]/repformer_rcut (Argument), 132
repformer_rcut_smth:
    model[standard]/descriptor[dpa2]/repformer_rcut_smth (Argument), 132
repformer_set_davg_zero:
    model[standard]/descriptor[dpa2]/repformer_set_davg_zero (Argument), 135
repformer_update_g1_has_attn:
    model[standard]/descriptor[dpa2]/repformer_update_g1_has_attn (Argument), 134
repformer_update_g1_has_conv:
    model[standard]/descriptor[dpa2]/repformer_update_g1_has_conv (Argument), 134
repformer_update_g1_has_drrd:
    model[standard]/descriptor[dpa2]/repformer_update_g1_has_drrd (Argument), 134
repformer_update_g1_has_grrg:
    model[standard]/descriptor[dpa2]/repformer_update_g1_has_grrg (Argument), 134
repformer_update_g2_has_attn:
    model[standard]/descriptor[dpa2]/repformer_update_g2_has_attn (Argument), 134
repformer_update_g2_has_g1g1:
    model[standard]/descriptor[dpa2]/repformer_update_g2_has_g1g1 (Argument), 134
repformer_update_h2:
    model[standard]/descriptor[dpa2]/repformer_update_h2 (Argument), 134
repformer_update_style:
    model[standard]/descriptor[dpa2]/repformer_update_style (Argument), 135
RepformerLayer (class in

deepmd.pt.model.descriptor.repformer_layer), 487
repinit_activation:
model[standard]/descriptor[dpa2]/repinit_activation (Argument), 133
repinit_axis_neuron:
model[standard]/descriptor[dpa2]/repinit_axis_neuron (Argument), 133
repinit_neuron:
model[standard]/descriptor[dpa2]/repinit_neuron (Argument), 132
repinit_nsel:
model[standard]/descriptor[dpa2]/repinit_nsel (Argument), 132
repinit_rcut:
model[standard]/descriptor[dpa2]/repinit_rcut (Argument), 132
repinit_rcut_smth:
model[standard]/descriptor[dpa2]/repinit_rcut_smth (Argument), 132
repinit_set_davg_zero:
model[standard]/descriptor[dpa2]/repinit_set_davg_zero (Argument), 133
replace_model_params_with_frz_multi_model() (in module deepmd.tf.utils.multi_init), 960
replace_model_params_with_pretrained_model() (in module deepmd.tf.utils.finetune), 951
RESERVED_PRECISON_DICT (in module deepmd.dpmodel.common), 374
RESERVED_PRECISON_DICT (in module deepmd.pt.utils.env), 618
reset_default_tf_session_config() (in module deepmd.tf.env), 1003
reset_get_batch() (deepmd.tf.utils.data.DeepmdData method), 946
reset_get_batch() (deepmd.tf.utils.DeepmdData method), 987
reset_get_batch() (deepmd.utils.data.DeepmdData method), 1028
ResidualDeep (class in deepmd.pt.model.network.network), 562
ResidualLinear (class in deepmd.pt.model.network.network), 555
resnet (deepmd.pt.model.network.network.ResidualLinear attribute), 555
resnet_dt:
model/type_embedding/resnet_dt (Argument), 113
model[standard]/descriptor[se_a_ebd_v2]/resnet_dt (Argument), 137
model[standard]/descriptor[se_a_mask]/resnet_dt (Argument), 139
model[standard]/descriptor[se_a_tpe]/resnet_dt (Argument), 121
model[standard]/descriptor[se_atten_v2]/resnet_dt (Argument), 130
model[standard]/descriptor[se_atten]/resnet_dt (Argument), 126
model[standard]/descriptor[se_e2_a]/resnet_dt (Argument), 118
model[standard]/descriptor[se_e2_r]/resnet_dt (Argument), 124
model[standard]/descriptor[se_e3]/resnet_dt (Argument), 120
model[standard]/fitting_net[dipole]/resnet_dt (Argument), 145
model[standard]/fitting_net[dos]/resnet_dt (Argument), 142
model[standard]/fitting_net[ener]/resnet_dt (Argument), 141
model[standard]/fitting_net[polar]/resnet_dt (Argument), 143
restore_descriptor:
nvnmd/restore_descriptor (Argument), 160
restore_fitting_net:
nvnmd/restore_fitting_net (Argument), 160
reverse_bin() (deepmd.tf.nvnmd.utils.Encode method), 922
reverse_bin() (deepmd.tf.nvnmd.utils.encode.Encode method), 914
reverse_map() (deepmd.tf.DeepEval static method), 1007
reverse_map() (deepmd.tf.infer.deep_eval.DeepEval static method), 800
reverse_map() (deepmd.tf.infer.deep_eval.DeepEvalOld static method), 806
reverse_map() (deepmd.tf.infer.DeepEval static method), 825
rglob() (deepmd.tf.utils.path.DPH5Path method), 970
rglob() (deepmd.tf.utils.path.DPOSPath method), 971
rglob() (deepmd.tf.utils.path.DPPath method), 973
rglob() (deepmd.utils.path.DPH5Path method), 1050
rglob() (deepmd.utils.path.DPOSPath method), 1048
rglob() (deepmd.utils.path.DPPath method), 1046
ROOT_DIR (in module deepmd.tf.entrypoints.ipi), 749
run() (deepmd.pt.train.training.Trainer method), 609
run() (deepmd.pt.utils.dataloader.BackgroundConsumer method), 615
run_s2g() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 903
run_s2g() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
run_sess() (in module deepmd.tf.utils.sess), 978

run_t2g() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 904
run_t2g() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
run_u2s() (deepmd.tf.nvnmd.entrypoints.mapt.MapTable method), 903
run_u2s() (deepmd.tf.nvnmd.entrypoints.MapTable method), 908
RunOptions (class in deepmd.tf.train.run_options), 932

S

safe_cast_tensor() (in module deepmd.tf.common), 1000
SAMPLER_RECORD (in module deepmd.pt.utils.env), 618
save() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
save() (deepmd.tf.nvnmd.utils.fio.FioBin method), 916
save() (deepmd.tf.nvnmd.utils.fio.FioDic method), 915
save() (deepmd.tf.nvnmd.utils.fio.FioJsonDic method), 916
save() (deepmd.tf.nvnmd.utils.fio.FioNpyDic method), 916
save() (deepmd.tf.nvnmd.utils.fio.FioTxt method), 916
save() (deepmd.tf.nvnmd.utils.FioBin method), 922
save() (deepmd.tf.nvnmd.utils.FioDic method), 922
save() (deepmd.tf.nvnmd.utils.FioTxt method), 923
save_checkpoint() (deepmd.tf.train.trainer.DPTrainer method), 935
save_ckpt:
    training/save_ckpt (Argument), 157
save_compressed() (deepmd.tf.train.trainer.DPTrainer method), 935
save_dp_model() (in module deepmd.dpmodel.utils), 373
save_dp_model() (in module deepmd.dpmodel.utils.serialization), 364
save_freq:
    training/save_freq (Argument), 157
save_model() (deepmd.pt.train.training.Trainer method), 609
save_numpy() (deepmd.tf.utils.path.DPH5Path method), 970
save_numpy() (deepmd.tf.utils.path.DPOSPath method), 971
save_numpy() (deepmd.tf.utils.path.DPPath method), 973
save_numpy() (deepmd.utils.path.DPH5Path method), 1049
save_numpy() (deepmd.utils.path.DPOSPath method), 1047
save_numpy() (deepmd.utils.path.DPPath method), 1046
save_stats() (deepmd.utils.env_mat_stat.EnvMatStat method), 1037
save_weight() (in module deepmd.tf.nvnmd.entrypoints), 907
save_weight() (in module deepmd.tf.nvnmd.entrypoints.freeze), 901
scale:
    model[standard]/fitting_net[polar]/scale (Argument), 144
scale_by_worker:
    learning_rate/scale_by_worker (Argument), 147
scaling_factor:
    model[standard]/descriptor[se_atten]/scaling_factor (Argument), 128
seed() (in module deepmd.pt.utils.dp_random), 617
seed() (in module deepmd.tf.utils.random), 976
seed() (in module deepmd.utils.random), 1053
seed:
    model/type_embedding/seed (Argument), 113
    model[standard]/descriptor[se_a_ebd_v2]/seed (Argument), 137
    model[standard]/descriptor[se_a_mask]/seed (Argument), 139
    model[standard]/descriptor[se_a_tpe]/seed (Argument), 122
    model[standard]/descriptor[se_atten_v2]/seed (Argument), 131
    model[standard]/descriptor[se_atten]/seed (Argument), 126
    model[standard]/descriptor[se_e2_a]/seed (Argument), 118
    model[standard]/descriptor[se_e2_r]/seed (Argument), 124
    model[standard]/descriptor[se_e3]/seed (Argument), 120
    model[standard]/fitting_net[dipole]/seed (Argument), 145
    model[standard]/fitting_net[dos]/seed (Argument), 143
    model[standard]/fitting_net[ener]/seed (Argument), 141
    model[standard]/fitting_net[polar]/seed (Argument), 144
    training/seed (Argument), 157
sel:
    model[pairtab]/sel (Argument), 146
    model[standard]/descriptor[se_a_ebd_v2]/sel (Argument), 136

model[standard]/descriptor[se_a_mask]/sel (Argument), 138
model[standard]/descriptor[se_a_tpe]/sel (Argument), 120
model[standard]/descriptor[se_atten_v2]/sel (Argument), 129
model[standard]/descriptor[se_atten]/sel (Argument), 125
model[standard]/descriptor[se_e2_a]/sel (Argument), 117
model[standard]/descriptor[se_e2_r]/sel (Argument), 123
model[standard]/descriptor[se_e3]/sel (Argument), 119
sel_a:
model[standard]/descriptor[loc_frame]/sel_a (Argument), 116
sel_r:
model[standard]/descriptor[loc_frame]/sel_r (Argument), 116
sel_type:
model[standard]/fitting_net[dipole]/sel_type (Argument), 145
model[standard]/fitting_net[polar]/sel_type (Argument), 144
select_idx_map() (in module deepmd.common), 1061
select_idx_map() (in module deepmd.tf.common), 999
serialize() (deepmd.dpmodel.atomic_model.base_atomic_model.BaseAtomicModel method), 262
serialize() (deepmd.dpmodel.atomic_model.BaseAtomicModel method), 273
serialize() (deepmd.dpmodel.atomic_model.dp_atomic_model.DPAtomicModel method), 264
serialize() (deepmd.dpmodel.atomic_model.DPAtomicModel method), 274
serialize() (deepmd.dpmodel.atomic_model.DPZBLLinearEnergyAtomicModel method), 275
serialize() (deepmd.dpmodel.atomic_model.linear_atomic_model.DPZBLLinearEnergyAtomicModel method), 267
serialize() (deepmd.dpmodel.atomic_model.linear_atomic_model.LinearEnergyAtomicModel method), 266
serialize() (deepmd.dpmodel.atomic_model.LinearEnergyAtomicModel method), 277
serialize() (deepmd.dpmodel.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 269
serialize() (deepmd.dpmodel.atomic_model.PairTabAtomicModel method), 278
serialize() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 302
serialize() (deepmd.dpmodel.descriptor.DescrptHybrid method), 304
serialize() (deepmd.dpmodel.descriptor.DescrptSeA method), 307
serialize() (deepmd.dpmodel.descriptor.DescrptSeR method), 309
serialize() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 method), 286
serialize() (deepmd.dpmodel.descriptor.dpa1.GatedAttentionLaye method), 288
serialize() (deepmd.dpmodel.descriptor.dpa1.NeighborGatedAtten method), 287
serialize() (deepmd.dpmodel.descriptor.dpa1.NeighborGatedAtten method), 287
serialize() (deepmd.dpmodel.descriptor.hybrid.DescrptHybrid method), 289
serialize() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA method), 294
serialize() (deepmd.dpmodel.descriptor.se_r.DescrptSeR method), 297
serialize() (deepmd.dpmodel.fitting.dipole_fitting.DipoleFitting method), 312
serialize() (deepmd.dpmodel.fitting.DipoleFitting method), 328
serialize() (deepmd.dpmodel.fitting.dos_fitting.DOSFittingNet method), 315
serialize() (deepmd.dpmodel.fitting.DOSFittingNet method), 331
serialize() (deepmd.dpmodel.fitting.ener_fitting.EnergyFittingNet method), 317
serialize() (deepmd.dpmodel.fitting.EnergyFittingNet method), 333
serialize() (deepmd.dpmodel.fitting.general_fitting.GeneralFitting method), 319
serialize() (deepmd.dpmodel.fitting.invar_fitting.InvarFitting method), 322
serialize() (deepmd.dpmodel.fitting.InvarFitting method), 335
serialize() (deepmd.dpmodel.fitting.PolarFitting method), 337
serialize() (deepmd.dpmodel.fitting.polarizability_fitting.PolarFitt method), 326
serialize() (deepmd.dpmodel.model.spin_model.SpinModel method), 345
serialize() (deepmd.dpmodel.model.SpinModel method), 349
serialize() (deepmd.dpmodel.utils.env_mat.EnvMat method), 353
serialize() (deepmd.dpmodel.utils.EnvMat method), 367
serialize() (deepmd.dpmodel.utils.NativeLayer method), 369
serialize() (deepmd.dpmodel.utils.network.LayerNorm method), 357
serialize() (deepmd.dpmodel.utils.network.NativeLayer method), 356
serialize() (deepmd.dpmodel.utils.network.NetworkCollection

method), 359
serialize() (deepmd.dpmodel.utils.NetworkCollection method), 370
serialize() (deepmd.dpmodel.utils.type_embed.TypeEmbedNet method), 365
serialize() (deepmd.pt.model.atomic_model.base_atomic_model.BaseAtomicModel method), 436
serialize() (deepmd.pt.model.atomic_model.BaseAtomicModel method), 453
serialize() (deepmd.pt.model.atomic_model.dp_atomic_model.DPAtomicModel method), 440
serialize() (deepmd.pt.model.atomic_model.DPAtomicModel method), 456
serialize() (deepmd.pt.model.atomic_model.DPZBLLinearEnergyAtomicModel method), 458
serialize() (deepmd.pt.model.atomic_model.linear_atomic_model.DPZBLLinearEnergyAtomicModel method), 446
serialize() (deepmd.pt.model.atomic_model.linear_atomic_model.LinearEnergyAtomicModel method), 444
serialize() (deepmd.pt.model.atomic_model.LinearEnergyAtomicModel method), 460
serialize() (deepmd.pt.model.atomic_model.pairtab_atomic_model.PairTabAtomicModel method), 447
serialize() (deepmd.pt.model.atomic_model.PairTabAtomicModel method), 462
serialize() (deepmd.pt.model.descriptor.DescrptDPA1 method), 513
serialize() (deepmd.pt.model.descriptor.DescrptDPA2 method), 516
serialize() (deepmd.pt.model.descriptor.DescrptHybrid method), 520
serialize() (deepmd.pt.model.descriptor.DescrptSeA method), 525
serialize() (deepmd.pt.model.descriptor.DescrptSeR method), 528
serialize() (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 method), 474
serialize() (deepmd.pt.model.descriptor.dpa2.DescrptDPA2 method), 478
serialize() (deepmd.pt.model.descriptor.hybrid.DescrptHybrid method), 483
serialize() (deepmd.pt.model.descriptor.se_a.DescrptSeA method), 494
serialize() (deepmd.pt.model.descriptor.se_atten.GatedAttentionLayer method), 502
serialize() (deepmd.pt.model.descriptor.se_atten.NeighborGatedAttention method), 499
serialize() (deepmd.pt.model.descriptor.se_atten.NeighborGatedAttentionLayer method), 500
serialize() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 504
serialize() (deepmd.pt.model.model.frozen.FrozenModel method), 534
serialize() (deepmd.pt.model.model.FrozenModel method), 544
serialize() (deepmd.pt.model.model.spin_model.SpinModel method), 539
serialize() (deepmd.pt.model.model.SpinModel method), 548
serialize() (deepmd.pt.model.network.layernorm.LayerNorm method), 550
serialize() (deepmd.pt.model.network.mlp.MLPLayer method), 552
serialize() (deepmd.pt.model.network.network.TypeEmbedNetCon method), 565
serialize() (deepmd.pt.model.task.dipole.DipoleFittingNet method), 582
serialize() (deepmd.pt.model.task.DipoleFittingNet method), 599
serialize() (deepmd.pt.model.task.dos.DOSFittingNet method), 584
serialize() (deepmd.pt.model.task.EnergyFittingNet method), 586
serialize() (deepmd.pt.model.task.ener.EnergyFittingNetDirect method), 587
serialize() (deepmd.pt.model.task.EnergyFittingNet method), 601
serialize() (deepmd.pt.model.task.EnergyFittingNetDirect method), 602
serialize() (deepmd.pt.model.task.fitting.GeneralFitting method), 590
serialize() (deepmd.pt.model.task.invar_fitting.InvarFitting method), 592
serialize() (deepmd.pt.model.task.PolarFittingNet method), 604
serialize() (deepmd.pt.model.task.polarizability.PolarFittingNet method), 595
serialize() (deepmd.tf.descriptor.Descriptor method), 705
serialize() (deepmd.tf.descriptor.descriptor.Descriptor method), 647
serialize() (deepmd.tf.descriptor.DescrptHybrid method), 710
serialize() (deepmd.tf.descriptor.DescrptSeA method), 719
serialize() (deepmd.tf.descriptor.DescrptSeAtten method), 736
serialize() (deepmd.tf.descriptor.DescrptSeR method), 742
serialize() (deepmd.tf.descriptor.hybrid.DescrptHybrid method), 652
serialize() (deepmd.tf.descriptor.se_a.DescrptSeA method), 665
serialize() (deepmd.tf.descriptor.se_atten.DescrptDPA1Compat method), 687
serialize() (deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 684
serialize() (deepmd.tf.descriptor.se_r.DescrptSeR

method), 693
serialize() (deepmd.tf.fit.dipole.DipoleFittingSeA method), 762
serialize() (deepmd.tf.fit.DipoleFittingSeA method), 780
serialize() (deepmd.tf.fit.dos.DOSFitting method), 766
serialize() (deepmd.tf.fit.DOSFitting method), 783
serialize() (deepmd.tf.fit.ener.EnerFitting method), 770
serialize() (deepmd.tf.fit.EnerFitting method), 787
serialize() (deepmd.tf.fit.Fitting method), 788
serialize() (deepmd.tf.fit.fitting.Fitting method), 772
serialize() (deepmd.tf.fit.polar.PolarFittingSeA method), 775
serialize() (deepmd.tf.fit.PolarFittingSeA method), 793
serialize() (deepmd.tf.model.frozen.FrozenModel method), 857
serialize() (deepmd.tf.model.FrozenModel method), 883
serialize() (deepmd.tf.model.model.Model method), 866
serialize() (deepmd.tf.model.model.StandardModel method), 867
serialize() (deepmd.tf.utils.pair_tab.PairTab method), 966
serialize() (deepmd.tf.utils.PairTab method), 992
serialize() (deepmd.tf.utils.type_embed.TypeEmbedNet method), 983
serialize() (deepmd.utils.pair_tab.PairTab method), 1044
serialize() (deepmd.utils.spin.Spin method), 1055
serialize_attention_layers() (deepmd.tf.descriptor.DescrptSeAtten method), 735
serialize_attention_layers() (deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 682
serialize_from_file() (in module deepmd.pt.utils.serialization), 633
serialize_from_file() (in module deepmd.tf.utils.serialization), 977
serialize_hook (deepmd.backend.backend.Backend property), 254
serialize_hook (deepmd.backend.dpmodel.DPModelBackend property), 256
serialize_hook (deepmd.backend.pytorch.PyTorchBackend property), 257
serialize_hook (deepmd.backend.tensorflow.TensorFlowBackend property), 259
serialize_network() (deepmd.tf.descriptor.se.DescrptSe method), 658
serialize_network() (deepmd.tf.fit.Fitting method), 788
serialize_network() (deepmd.tf.fit.fitting.Fitting method), 772
serialize_network_strip() (deepmd.tf.descriptor.DescrptSeAtten method), 735
serialize_network_strip() (deepmd.tf.descriptor.se_atten.DescrptSeAtten method), 682
sess (deepmd.tf.DeepEval property), 1006
sess (deepmd.tf.infer.deep_eval.DeepEval property), 799
sess (deepmd.tf.infer.deep_eval.DeepEvalOld property), 805
sess (deepmd.tf.infer.DeepEval property), 824
set_davg_zero:
    model[standard]/descriptor[se_a_ebd_v2]/set_davg_zero (Argument), 138
    model[standard]/descriptor[se_a_tpe]/set_davg_zero (Argument), 122
    model[standard]/descriptor[se_atten_v2]/set_davg_zero (Argument), 131
    model[standard]/descriptor[se_atten]/set_davg_zero (Argument), 127
    model[standard]/descriptor[se_e2_a]/set_davg_zero (Argument), 119
    model[standard]/descriptor[se_e2_r]/set_davg_zero (Argument), 125
    model[standard]/descriptor[se_e3]/set_davg_zero (Argument), 120
set_extra_state() (deepmd.pt.train.wrapper.ModelWrapper method), 611
set_grad_prefactor() (deepmd.pt.optimizer.LKF.LKFOptimizer method), 607
set_grad_prefactor() (deepmd.pt.optimizer.LKFOptimizer method), 608
set_log_handles() (in module deepmd.loggers), 413
set_log_handles() (in module deepmd.loggers.loggers), 412
set_log_handles() (in module deepmd.tf.loggers), 836
set_log_handles() (in module deepmd.tf.loggers.loggers), 835
set_noise() (deepmd.pt.utils.dataloader.DpLoaderSet method), 615
set_ntype() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 912
set_stat_mean_and_stddev() (deepmd.pt.model.descriptor.DescrptDPA1

method), 513
set_stat_mean_and_stddev() (deepmd.pt.model.descriptor.DescrptSeA method), 525
set_stat_mean_and_stddev() (deepmd.pt.model.descriptor.DescrptSeR method), 528
set_stat_mean_and_stddev() (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 method), 474
set_stat_mean_and_stddev() (deepmd.pt.model.descriptor.se_a.DescrptSeA method), 494
set_stat_mean_and_stddev() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 504
set_sys_probs() (deepmd.tf.utils.data_system.DeepmdDataSystem method), 949
set_sys_probs() (deepmd.tf.utils.DeepmdDataSystem method), 989
set_sys_probs() (deepmd.utils.data_system.DeepmdDataSystem method), 1031
setup_seed() (in module deepmd.pt.utils.dataloader), 615
share_params() (deepmd.dpmodel.descriptor.DescrptDPA1 method), 301
share_params() (deepmd.dpmodel.descriptor.DescrptHybrid method), 303
share_params() (deepmd.dpmodel.descriptor.DescrptSeA method), 306
share_params() (deepmd.dpmodel.descriptor.DescrptSeR method), 309
share_params() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 method), 285
share_params() (deepmd.dpmodel.descriptor.hybrid.DescrptHybrid method), 288
share_params() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA method), 293
share_params() (deepmd.dpmodel.descriptor.se_r.DescrptSeR method), 296
share_params() (deepmd.pt.model.descriptor.descriptor.DescriptorBlock method), 469
share_params() (deepmd.pt.model.descriptor.DescrptDPA1 method), 506
share_params() (deepmd.pt.model.descriptor.DescrptHybrid method), 512
share_params() (deepmd.pt.model.descriptor.DescrptSeA method), 515
share_params() (deepmd.pt.model.descriptor.DescrptSeR method), 519
share_params() (deepmd.pt.model.descriptor.DescrptSeA method), 524
share_params() (deepmd.pt.model.descriptor.DescrptSeR method), 527
share_params() (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 method), 473
share_params() (deepmd.pt.model.descriptor.dpa2.DescrptDPA2 method), 477
share_params() (deepmd.pt.model.descriptor.hybrid.DescrptHybrid method), 482
share_params() (deepmd.pt.model.descriptor.se_a.DescrptSeA method), 492
share_params() (deepmd.pt.model.descriptor.se_r.DescrptSeR method), 503
share_params() (deepmd.pt.model.network.network.TypeEmbedNet method), 564
share_params() (deepmd.pt.model.task.Fitting method), 603
share_params() (deepmd.pt.model.task.fitting.Fitting method), 589
share_params() (deepmd.pt.train.wrapper.ModelWrapper method), 610
shared_dropout() (deepmd.pt.model.network.network.Evoformer3b method), 577
SHARED_LIB_DIR (in module deepmd.env), 1063
SHARED_LIB_DIR (in module deepmd.tf.env), 1002
SHARED_LIB_MODULE (in module deepmd.env), 1063
SHARED_LIB_MODULE (in module deepmd.tf.env), 1002
shift_diag: model[standard]/fitting_net[polar]/shift_diag (Argument), 144
shuffle() (in module deepmd.pt.utils.dp_random), 618
shuffle() (in module deepmd.tf.utils.random), 976
shuffle() (in module deepmd.utils.random), 1053
simple_linear (in module deepmd.pt.model.descriptor.repformers), 490
SimpleLinear (class in deepmd.pt.model.network.network), 556
SimulationRegion (C++ class), 1277
SimulationRegion::~SimulationRegion (C++ function), 1277
SimulationRegion::affineTransform (C++ function), 1277
SimulationRegion::backup (C++ function), 1277
SimulationRegion::compactIndex (C++ function), 1278
SimulationRegion::computeShiftVec (C++ function), 1279
SimulationRegion::DBOX_XX (C++ member), 1279
SimulationRegion::DBOX_YY (C++ member), 1279
SimulationRegion::DBOX_ZZ (C++ member), 1279
SimulationRegion::diffNearestNeighbor (C++ function), 1278
SimulationRegion::getBoxOrigin (C++ function), 1277, 1278
SimulationRegion::getBoxTensor (C++ function),

1277
SimulationRegion::getInterShiftVec (C++ function), 1279
SimulationRegion::getNullShiftIndex (C++ function), 1278
SimulationRegion::getNumbShiftVec (C++ function), 1278
SimulationRegion::getRecBoxTensor (C++ function), 1277
SimulationRegion::getShiftIndex (C++ function), 1278
SimulationRegion::getShiftVec (C++ function), 1278
SimulationRegion::getShiftVecTotalSize (C++ function), 1278
SimulationRegion::getVolume (C++ function), 1278
SimulationRegion::index3to1 (C++ function), 1279
SimulationRegion::inter2Phys (C++ function), 1278
SimulationRegion::inter_shift_vec (C++ member), 1279
SimulationRegion::isPeriodic (C++ function), 1278
SimulationRegion::NBOX_XX (C++ member), 1279
SimulationRegion::NBOX_YY (C++ member), 1279
SimulationRegion::NBOX_ZZ (C++ member), 1279
SimulationRegion::phys2Inter (C++ function), 1278
SimulationRegion::recover (C++ function), 1277
SimulationRegion::reinitBox (C++ function), 1277
SimulationRegion::reinitOrigin (C++ function), 1277
SimulationRegion::shift_info_size (C++ member), 1279
SimulationRegion::shift_vec (C++ member), 1279
SimulationRegion::shift_vec_size (C++ member), 1279
SimulationRegion::shiftCoord (C++ function), 1278
SimulationRegion::SimulationRegion (C++ function), 1277
SimulationRegion::SPACENDIM (C++ member), 1279
SimulationRegion::toFaceDistance (C++ function), 1278
size (deepmd.dpmodel.output_def.OutputVariableDef property), 377
size (deepmd.dpmodel.OutputVariableDef property), 382
smin_alpha:
model/smin_alpha (Argument), 111
smooth_type_embedding:
model[standard]/descriptor[se_atten]/smooth_type_embed (Argument), 127
soft_min_force() (in module deepmd.tf.env.op_module), 1113
soft_min_force_grad() (in module deepmd.tf.env.op_grads_module), 1129
soft_min_switch() (in module deepmd.tf.env.op_module), 1114
soft_min_virial() (in module deepmd.tf.env.op_module), 1114
soft_min_virial_grad() (in module deepmd.tf.env.op_grads_module), 1129
softmax_dropout() (in module deepmd.pt.model.network.network), 555
SoftMinForce() (in module deepmd.tf.env.op_module), 1087
SoftMinForceGrad() (in module deepmd.tf.env.op_grads_module), 1126
SoftMinSwitch() (in module deepmd.tf.env.op_module), 1088
SoftMinVirial() (in module deepmd.tf.env.op_module), 1088
SoftMinVirialGrad() (in module deepmd.tf.env.op_grads_module), 1126
sort_input() (deepmd.tf.DeepEval static method), 1006
sort_input() (deepmd.tf.infer.deep_eval.DeepEval static method), 799
sort_input() (deepmd.tf.infer.deep_eval.DeepEvalOld static method), 805
sort_input() (deepmd.tf.infer.DeepEval static method), 825
Spin (class in deepmd.tf.utils.spin), 978
Spin (class in deepmd.utils.spin), 1054
spin:
model/spin (Argument), 114
spin_args() (in module deepmd.utils.argcheck), 1016
spin_norm:
model/spin/spin_norm (Argument), 114
SpinEnergyModel (class in deepmd.pt.model.model), 546
SpinEnergyModel (class in deepmd.pt.model.model.spin_model), 539
SpinModel (class in deepmd.dpmodel.model), 348
SpinModel (class in deepmd.dpmodel.model.spin_model), 344
SpinModel (class in deepmd.pt.model.model), 546
SpinModel (class in deepmd.pt.model.model.spin_model), 537
split_bin() (deepmd.tf.nvnmd.utils.Encode

method), 922
split_bin() (deepmd.tf.nvnmd.utils.encode.Encode method), 914
split_expo_mant() (deepmd.tf.nvnmd.utils.Encode method), 921
split_expo_mant() (deepmd.tf.nvnmd.utils.encode.Encode method), 913
split_flt (C++ function), 1319
SQRT_2_PI (C macro), 1325
srtab_add_bias:
    model/srtab_add_bias (Argument), 112
standard_model_args() (in module deepmd.utils.argcheck), 1018
StandardModel (class in deepmd.tf.model.model), 866
start_dpgui() (in module deepmd.entrypoints.gui), 383
start_dpgui() (in module deepmd.tf.entrypoints), 757
start_dpgui() (in module deepmd.tf.entrypoints.gui), 748
start_lr() (deepmd.tf.utils.learning_rate.LearningRateExp method), 959
start_lr() (deepmd.tf.utils.LearningRateExp method), 992
start_lr:
    learning_rate[exp]/start_lr (Argument), 147
start_pref() (in module deepmd.utils.argcheck), 1018
start_pref_acdf:
    loss[dos]/start_pref_acdf (Argument), 153
start_pref_ados:
    loss[dos]/start_pref_ados (Argument), 153
start_pref_ae:
    loss[ener_spin]/start_pref_ae (Argument), 151
    loss[ener]/start_pref_ae (Argument), 149
start_pref_cdf:
    loss[dos]/start_pref_cdf (Argument), 153
start_pref_dos:
    loss[dos]/start_pref_dos (Argument), 152
start_pref_e:
    loss[ener_spin]/start_pref_e (Argument), 150
    loss[ener]/start_pref_e (Argument), 148
start_pref_f:
    loss[ener]/start_pref_f (Argument), 148
start_pref_fm:
    loss[ener_spin]/start_pref_fm (Argument), 151
start_pref_fr:
    loss[ener_spin]/start_pref_fr (Argument), 150
start_pref_gf:
    loss[ener]/start_pref_gf (Argument), 150
start_pref_pf:
    loss[ener_spin]/start_pref_pf (Argument), 152
    loss[ener]/start_pref_pf (Argument), 149
start_pref_v:
    loss[ener_spin]/start_pref_v (Argument), 151
    loss[ener]/start_pref_v (Argument), 149
stat_file:
    training/stat_file (Argument), 159
StatItem (class in deepmd.utils.env_mat_stat), 1036
step() (deepmd.pt.optimizer.LKF.LKFOptimizer method), 607
step() (deepmd.pt.optimizer.LKFOptimizer method), 608
stop_lr:
    learning_rate[exp]/stop_lr (Argument), 147
stripped_type_embedding:
    model[standard]/descriptor[se_atten]/stripped_type_emb (Argument), 127
suffixes (deepmd.backend.backend.Backend attribute), 255
suffixes (deepmd.backend.dpmodel.DPModelBackend attribute), 257
suffixes (deepmd.backend.pytorch.PyTorchBackend attribute), 258
suffixes (deepmd.backend.tensorflow.TensorFlowBackend attribute), 260
SummaryPrinter (class in deepmd.pt.entrypoints.main), 415
SummaryPrinter (class in deepmd.utils.summary), 1055
sw_rmax:
    model/sw_rmax (Argument), 112
sw_rmin:
    model/sw_rmin (Argument), 112
sys_charge_map:
    model/modifier[dipole_charge]/sys_charge_map (Argument), 114
sys_probs:
    training/training_data/sys_probs (Argument), 155
    training/validation_data/sys_probs (Argument), 156
systems:
    training/training_data/systems (Argument), 154
    training/validation_data/systems (Argument), 155

T

tab_file:

model[pairtab]/tab_file (Argument), 146
tabulate_fusion() (in module deepmd.tf.env.op_module), 1115
tabulate_fusion_grad() (in module deepmd.tf.env.op_module), 1115
tabulate_fusion_grad_grad() (in module deepmd.tf.env.op_module), 1115
tabulate_fusion_se_a() (in module deepmd.tf.env.op_module), 1116
tabulate_fusion_se_a_grad() (in module deepmd.tf.env.op_module), 1116
tabulate_fusion_se_a_grad_grad() (in module deepmd.tf.env.op_module), 1116
tabulate_fusion_se_atten() (in module deepmd.tf.env.op_module), 1117
tabulate_fusion_se_atten_grad() (in module deepmd.tf.env.op_module), 1117
tabulate_fusion_se_atten_grad_grad() (in module deepmd.tf.env.op_module), 1118
tabulate_fusion_se_r() (in module deepmd.tf.env.op_module), 1118
tabulate_fusion_se_r_grad() (in module deepmd.tf.env.op_module), 1118
tabulate_fusion_se_r_grad_grad() (in module deepmd.tf.env.op_module), 1119
tabulate_fusion_se_t() (in module deepmd.tf.env.op_module), 1119
tabulate_fusion_se_t_grad() (in module deepmd.tf.env.op_module), 1119
tabulate_fusion_se_t_grad_grad() (in module deepmd.tf.env.op_module), 1120
TabulateCheckerGPUExecuteFunctor (C++ struct), 1276
TabulateCheckerGPUExecuteFunctor::operator() (C++ function), 1276
TabulateFusion() (in module deepmd.tf.env.op_module), 1088
TabulateFusionGPUExecuteFunctor (C++ struct), 1276
TabulateFusionGPUExecuteFunctor::operator() (C++ function), 1276
TabulateFusionGrad() (in module deepmd.tf.env.op_module), 1089
TabulateFusionGradGPUExecuteFunctor (C++ struct), 1277
TabulateFusionGradGPUExecuteFunctor::operator() (C++ function), 1277
TabulateFusionGradGrad() (in module deepmd.tf.env.op_module), 1089
TabulateFusionSeA() (in module deepmd.tf.env.op_module), 1090
TabulateFusionSeAGrad() (in module deepmd.tf.env.op_module), 1090
TabulateFusionSeAGradGrad() (in module deepmd.tf.env.op_module), 1090
TabulateFusionSeAtten() (in module deepmd.tf.env.op_module), 1091
TabulateFusionSeAttenGrad() (in module deepmd.tf.env.op_module), 1091
TabulateFusionSeAttenGradGrad() (in module deepmd.tf.env.op_module), 1091
TabulateFusionSeR() (in module deepmd.tf.env.op_module), 1092
TabulateFusionSeRGrad() (in module deepmd.tf.env.op_module), 1092
TabulateFusionSeRGradGrad() (in module deepmd.tf.env.op_module), 1093
TabulateFusionSeT() (in module deepmd.tf.env.op_module), 1093
TabulateFusionSeTGrad() (in module deepmd.tf.env.op_module), 1093
TabulateFusionSeTGradGrad() (in module deepmd.tf.env.op_module), 1094
take_deriv() (in module deepmd.pt.model.model.transform_output), 540
tanh4() (in module deepmd.tf.nvnmd.utils.network), 917
tanh4_flt_nvnmd() (in module deepmd.tf.env.op_module), 1120
Tanh4FltNvnmd() (in module deepmd.tf.env.op_module), 1094
task_deriv_one() (in module deepmd.pt.model.model.transform_output), 540
TaskLoss (class in deepmd.pt.loss), 431
TaskLoss (class in deepmd.pt.loss.loss), 425
tebd_dim:
    model[standard]/descriptor[dpa2]/tebd_dim (Argument), 132
    model[standard]/descriptor[se_atten]/tebd_dim (Argument), 128
tebd_input_mode:
    model[standard]/descriptor[se_atten]/tebd_input_mode (Argument), 128
tebd_mode (deepmd.pt.model.network.network.TypeFilter attribute), 556
temperature:
    model[standard]/descriptor[se_atten]/temperature (Argument), 129
Tensor() (in module deepmd.pt.model.network.network), 553
tensorboard:
    training/tensorboard (Argument), 158
tensorboard_freq:
    training/tensorboard_freq (Argument), 158
tensorboard_log_dir:
    training/tensorboard_log_dir (Argument),

158
TensorFlowBackend (class in deepmd.backend.tensorflow), 259
TensorLoss (class in deepmd.pt.loss), 432
TensorLoss (class in deepmd.pt.loss.tensor), 426
TensorLoss (class in deepmd.tf.loss), 849
TensorLoss (class in deepmd.tf.loss.tensor), 844
TensorModel (class in deepmd.tf.model.tensor), 877
tensors (deepmd.tf.infer.deep_tensor.DeepTensor attribute), 812
test() (in module deepmd.entrypoints.test), 386
test() (in module deepmd.tf.entrypoints), 758
test() (in module deepmd.tf.entrypoints.test), 752
Tester (class in deepmd.pt.infer.inference), 419
tf_dir (in module deepmd.tf.lmp), 1005
tf_py_version (in module deepmd.tf.env), 1002
TF_VERSION (in module deepmd.tf.env), 1003
time_training:
    training/time_training (Argument), 157
to_dict() (deepmd.utils.data.DataRequirementItem method), 1029
to_face_distance() (in module deepmd.dpmodel.utils), 373
to_face_distance() (in module deepmd.dpmodel.utils.region), 362
to_face_distance() (in module deepmd.pt.utils.region), 632
to_face_distance() (in module deepmd.tf.utils.region), 977
to_numpy_array() (in module deepmd.pt.utils.utils), 637
to_torch_tensor() (in module deepmd.pt.utils.utils), 637
torch_linear() (in module deepmd.pt.model.descriptor.repformer_layer), 484
torch_linear() (in module deepmd.pt.model.descriptor.repformers), 489
TPB (C macro), 1325
train() (deepmd.tf.train.trainer.DPTrainer method), 935
train() (in module deepmd.pt.entrypoints.main), 415
train() (in module deepmd.tf.entrypoints.train), 753
train_dp() (in module deepmd.tf.entrypoints), 759
train_nvnmd() (in module deepmd.tf.nvnmd.entrypoints.train), 904
trainable:
    model/type_embedding/trainable (Argument), 113
    model[standard]/descriptor[se_a_ebd_v2]/trainable (Argument), 137
    model[standard]/descriptor[se_a_mask]/trainable (Argument), 139
    model[standard]/descriptor[se_a_tpe]/trainable (Argument), 122
    model[standard]/descriptor[se_atten_v2]/trainable (Argument), 130
    model[standard]/descriptor[se_atten]/trainable (Argument), 126
    model[standard]/descriptor[se_e2_a]/trainable (Argument), 118
    model[standard]/descriptor[se_e2_r]/trainable (Argument), 124
    model[standard]/descriptor[se_e3]/trainable (Argument), 120
    model[standard]/fitting_net[dos]/trainable (Argument), 143
    model[standard]/fitting_net[ener]/trainable (Argument), 141
trainable_ln:
    model[standard]/descriptor[se_atten]/trainable_ln (Argument), 128
Trainer (class in deepmd.pt.train.training), 609
training (Argument)
    training:, 154
training/data_dict (Argument)
    data_dict:, 158
training/disp_file (Argument)
    disp_file:, 157
training/disp_freq (Argument)
    disp_freq:, 157
training/disp_training (Argument)
    disp_training:, 157
training/enable_profiler (Argument)
    enable_profiler:, 158
training/fitting_weight (Argument)
    fitting_weight:, 158
training/gradient_max_norm (Argument)
    gradient_max_norm:, 159
training/max_ckpt_keep (Argument)
    max_ckpt_keep:, 157
training/mixed_precision (Argument)
    mixed_precision:, 156
training/mixed_precision/compute_prec (Argument)
    compute_prec:, 156
training/mixed_precision/output_prec (Argument)
    output_prec:, 156
training/numb_steps (Argument)
    numb_steps:, 157
training/opt_type (Argument)
    opt_type:, 159
training/profiling (Argument)

profiling:, 157
training/profiling_file (Argument)
profiling_file:, 158
training/save_ckpt (Argument)
save_ckpt:, 157
training/save_freq (Argument)
save_freq:, 157
training/seed (Argument)
seed:, 157
training/stat_file (Argument)
stat_file:, 159
training/tensorboard (Argument)
tensorboard:, 158
training/tensorboard_freq (Argument)
tensorboard_freq:, 158
training/tensorboard_log_dir (Argument)
tensorboard_log_dir:, 158
training/time_training (Argument)
time_training:, 157
training/training_data (Argument)
training_data:, 154
training/training_data/auto_prob (Argument)
auto_prob:, 155
training/training_data/batch_size (Argument)
batch_size:, 154
training/training_data/sys_probs (Argument)
sys_probs:, 155
training/training_data/systems (Argument)
systems:, 154
training/validation_data (Argument)
validation_data:, 155
training/validation_data/auto_prob (Argument)
auto_prob:, 156
training/validation_data/batch_size (Argument)
batch_size:, 155
training/validation_data/numb_btch (Argument)
numb_btch:, 156
training/validation_data/sys_probs (Argument)
sys_probs:, 156
training/validation_data/systems (Argument)
systems:, 155
training/warmup_steps (Argument)
warmup_steps:, 158
training:
training (Argument), 154
training_args() (in module deepmd.utils.argcheck), 1018
training_data:
training/training_data (Argument), 154
training_data_args() (in module deepmd.utils.argcheck), 1018
training[LKF]/kf_blocksize (Argument)
kf_blocksize:, 159
transfer() (in module deepmd.tf.entrypoints), 759
transfer() (in module deepmd.tf.entrypoints.transfer), 754
TRANSFER_PATTERN (in module deepmd.tf.env), 1003
Transition (class in deepmd.pt.model.network.network), 558
traverse_model_dict() (in module deepmd.dpmodel.utils), 373
traverse_model_dict() (in module deepmd.dpmodel.utils.serialization), 363
TriangleMultiplication (class in deepmd.pt.model.network.network), 573
type:
learning_rate/type (Argument), 147
loss/type (Argument), 148
model/modifier/type (Argument), 113
model/type (Argument), 115
model[standard]/descriptor/type (Argument), 115
model[standard]/fitting_net/type (Argument), 140
type_embedding:
model/type_embedding (Argument), 112
type_embedding_args() (in module deepmd.tf.utils.argcheck), 937
type_embedding_args() (in module deepmd.utils.argcheck), 1016
TYPE_EMBEDDING_PATTERN (in module deepmd.tf.env), 1002
type_map (in module deepmd.utils.econf_embd), 1035
type_map:
model/type_map (Argument), 111
type_nchanl:
model[standard]/descriptor[se_a_tpe]/type_nchanl (Argument), 122
type_nlayer:
model[standard]/descriptor[se_a_tpe]/type_nlayer (Argument), 123
type_one_side:
model[standard]/descriptor[se_a_ebd_v2]/type_one_side (Argument), 137
model[standard]/descriptor[se_a_mask]/type_one_side (Argument), 139
model[standard]/descriptor[se_a_tpe]/type_one_side (Argument), 121
model[standard]/descriptor[se_atten_v2]/type_one_side (Argument), 130
model[standard]/descriptor[se_atten]/type_one_side (Argument), 126

model[standard]/descriptor[se_e2_a]/type_one_side (Argument), 118
model[standard]/descriptor[se_e2_r]/type_one_side (Argument), 124
TypeEmbedNet (class in deepmd.dpmodel.utils.type_embed), 364
TypeEmbedNet (class in deepmd.pt.model.network.network), 563
TypeEmbedNet (class in deepmd.tf.utils.type_embed), 982
TypeEmbedNetConsistent (class in deepmd.pt.model.network.network), 564
TypeFilter (class in deepmd.pt.model.network.network), 556
TypePredictNet (class in deepmd.pt.model.task), 605
TypePredictNet (class in deepmd.pt.model.task.type_predict), 595

U

U_Flt64_Int64 (C++ union), 1280
U_Flt64_Int64::nflt (C++ member), 1280
U_Flt64_Int64::nint (C++ member), 1280
uint_64 (C++ type), 1326
unaggregated_dy2_dx() (in module deepmd.tf.env.op_module), 1120
unaggregated_dy2_dx_s() (in module deepmd.tf.env.op_module), 1121
unaggregated_dy_dx() (in module deepmd.tf.env.op_module), 1121
unaggregated_dy_dx_s() (in module deepmd.tf.env.op_module), 1121
UnaggregatedDy2Dx() (in module deepmd.tf.env.op_module), 1094
UnaggregatedDy2DxS() (in module deepmd.tf.env.op_module), 1094
UnaggregatedDyDx() (in module deepmd.tf.env.op_module), 1095
UnaggregatedDyDxS() (in module deepmd.tf.env.op_module), 1095
update() (deepmd.tf.nvnmd.utils.fio.FioDic method), 916
update() (deepmd.tf.nvnmd.utils.FioDic method), 922
update_attention_layers_serialize() (deepmd.tf.descriptor.se_atten.DescrptDPA1Compat method), 687
update_config() (deepmd.tf.nvnmd.utils.config.NvnmdConfig method), 911
update_deepmd_input() (in module deepmd.tf.utils.compat), 939
update_deepmd_input() (in module deepmd.utils.compat), 1025
update_denoise_coord() (deepmd.pt.optimizer.KFOptimizerWrapper method), 608
update_denoise_coord() (deepmd.pt.optimizer.KFWrapper.KFOptimizerWrapper method), 606
update_energy() (deepmd.pt.optimizer.KFOptimizerWrapper method), 608
update_energy() (deepmd.pt.optimizer.KFWrapper.KFOptimizerWr method), 606
update_force() (deepmd.pt.optimizer.KFOptimizerWrapper method), 608
update_force() (deepmd.pt.optimizer.KFWrapper.KFOptimizerWra method), 606
update_one_sel() (deepmd.utils.update_sel.BaseUpdateSel method), 1056
update_pair() (deepmd.pt.model.network.network.Evoformer3bEnc method), 576
update_sel() (deepmd.dpmodel.descriptor.DescrptDPA1 class method), 302
update_sel() (deepmd.dpmodel.descriptor.DescrptHybrid class method), 303
update_sel() (deepmd.dpmodel.descriptor.DescrptSeA class method), 307
update_sel() (deepmd.dpmodel.descriptor.DescrptSeR class method), 309
update_sel() (deepmd.dpmodel.descriptor.dpa1.DescrptDPA1 class method), 286
update_sel() (deepmd.dpmodel.descriptor.hybrid.DescrptHybrid class method), 289
update_sel() (deepmd.dpmodel.descriptor.se_e2_a.DescrptSeA class method), 294
update_sel() (deepmd.dpmodel.descriptor.se_r.DescrptSeR class method), 297
update_sel() (deepmd.dpmodel.DPModelCommon class method), 380
update_sel() (deepmd.dpmodel.model.dp_model.DPModelCommon class method), 342
update_sel() (deepmd.dpmodel.model.DPModelCommon class method), 348
update_sel() (deepmd.pt.model.descriptor.DescrptDPA1 class method), 514
update_sel() (deepmd.pt.model.descriptor.DescrptDPA2 class method), 517
update_sel() (deepmd.pt.model.descriptor.DescrptHybrid class method), 520
update_sel() (deepmd.pt.model.descriptor.DescrptSeA class method), 526
update_sel() (deepmd.pt.model.descriptor.DescrptSeR class method), 528
update_sel() (deepmd.pt.model.descriptor.dpa1.DescrptDPA1 class method), 475
update_sel() (deepmd.pt.model.descriptor.dpa2.DescrptDPA2 class method), 478

update_sel() (deepmd.pt.model.descriptor.hybrid.DescrptHybrid class method), 483
update_sel() (deepmd.pt.model.descriptor.se_a.DescrptSeA class method), 494
update_sel() (deepmd.pt.model.descriptor.se_r.DescrptSeR class method), 504
update_sel() (deepmd.pt.model.model.dp_model.DPModelCommon class method), 530
update_sel() (deepmd.pt.model.model.dp_zbl_model.DPZBLModel class method), 532
update_sel() (deepmd.pt.model.model.DPModelCommon class method), 541
update_sel() (deepmd.pt.model.model.DPZBLModel class method), 542
update_sel() (deepmd.pt.model.model.frozen.FrozenModel class method), 534
update_sel() (deepmd.pt.model.model.FrozenModel class method), 544
update_sel() (deepmd.tf.descriptor.Descriptor class method), 705
update_sel() (deepmd.tf.descriptor.descriptor.Descriptor class method), 647
update_sel() (deepmd.tf.descriptor.DescrptHybrid class method), 709
update_sel() (deepmd.tf.descriptor.DescrptLocFrame class method), 713
update_sel() (deepmd.tf.descriptor.DescrptSeAMask class method), 729
update_sel() (deepmd.tf.descriptor.DescrptSeAtten class method), 734
update_sel() (deepmd.tf.descriptor.hybrid.DescrptHybrid class method), 652
update_sel() (deepmd.tf.descriptor.loc_frame.DescrptLocFrame class method), 656
update_sel() (deepmd.tf.descriptor.se.DescrptSe class method), 658
update_sel() (deepmd.tf.descriptor.se_a_mask.DescrptSeAMask class method), 676
update_sel() (deepmd.tf.descriptor.se_atten.DescrptSeAtten class method), 682
update_sel() (deepmd.tf.model.frozen.FrozenModel class method), 857
update_sel() (deepmd.tf.model.FrozenModel class method), 883
update_sel() (deepmd.tf.model.linear.LinearModel class method), 859
update_sel() (deepmd.tf.model.model.Model class method), 865
update_sel() (deepmd.tf.model.model.StandardModel class method), 867
update_sel() (deepmd.tf.model.multi.MultiModel class method), 871
update_sel() (deepmd.tf.model.MultiModel class method), 896
update_sel() (deepmd.tf.model.pairtab.PairTabModel class method), 873
update_sel() (deepmd.tf.model.PairTabModel class method), 886
update_sel() (deepmd.tf.model.pairwise_dprc.PairwiseDPRc class method), 876
update_sel() (deepmd.tf.model.PairwiseDPRc class method), 888
UpdateSel (class in deepmd.dpmodel.utils.update_sel), 365
UpdateSel (class in deepmd.pt.utils.update_sel), 636
UpdateSel (class in deepmd.tf.utils.update_sel), 984
use_aparam_as_mask:
    model[standard]/fitting_net[ener]/use_aparam_as_mask (Argument), 142
use_spin:
    model/spin/use_spin (Argument), 114
use_srtab:
    model/use_srtab (Argument), 111
use_tebd (deepmd.pt.model.network.network.TypeFilter attribute), 556
use_timestep (deepmd.pt.model.network.network.SimpleLinear attribute), 557

V

VALID_ACTIVATION (in module deepmd.common), 1060
valid_on_the_fly() (deepmd.tf.train.trainer.DPTrainer method), 935
VALID_PRECISION (in module deepmd.common), 1060
validation_data:
    training/validation_data (Argument), 155
validation_data_args() (in module deepmd.utils.argcheck), 1018
value() (deepmd.pt.utils.learning_rate.LearningRateExp method), 622
value() (deepmd.tf.utils.learning_rate.LearningRateExp method), 959
value() (deepmd.tf.utils.LearningRateExp method), 992
variable_summaries() (in module deepmd.tf.utils.network), 964
VariantABCMeta (class in deepmd.pt.utils.plugin), 629
VariantABCMeta (class in deepmd.tf.utils.plugin), 975
VariantABCMeta (class in deepmd.utils.plugin), 1052
VariantMeta (class in deepmd.pt.utils.plugin), 629
VariantMeta (class in deepmd.tf.utils.plugin), 975
VariantMeta (class in deepmd.utils.plugin), 1052
version:
    nvnmd/version (Argument), 159
virtual_len:

model/spin/virtual_len (Argument), 114
virtual_scale:
model/spin/virtual_scale (Argument), 115

W

warmup_steps:
training/warmup_steps (Argument), 158
weight (deepmd.pt.model.network.network.Embedding attribute), 559
weight (deepmd.pt.model.network.network.Linear attribute), 558
weight_file:
nvnmd/weight_file (Argument), 160
weighted_average() (in module deepmd.tf.utils.weight_avg), 984
weighted_average() (in module deepmd.utils.weight_avg), 1057
weights:
model[linear_ener]/weights (Argument), 147
WELCOME (deepmd.utils.summary.SummaryPrinter attribute), 1055
WFCModel (class in deepmd.tf.model), 897
WFCModel (class in deepmd.tf.model.tensor), 879
world_size (deepmd.tf.train.run_options.RunOptions attribute), 933
Wrap (class in deepmd.tf.nvnmd.entrypoints), 908
Wrap (class in deepmd.tf.nvnmd.entrypoints.wrap), 905
wrap() (deepmd.tf.nvnmd.entrypoints.Wrap method), 909
wrap() (deepmd.tf.nvnmd.entrypoints.wrap.Wrap method), 906
wrap() (in module deepmd.tf.nvnmd.entrypoints.wrap), 906
wrap_bias() (deepmd.tf.nvnmd.entrypoints.Wrap method), 909
wrap_bias() (deepmd.tf.nvnmd.entrypoints.wrap.Wrap method), 906
wrap_dscp() (deepmd.tf.nvnmd.entrypoints.Wrap method), 909
wrap_dscp() (deepmd.tf.nvnmd.entrypoints.wrap.Wrap method), 906
wrap_fitn() (deepmd.tf.nvnmd.entrypoints.Wrap method), 909
wrap_fitn() (deepmd.tf.nvnmd.entrypoints.wrap.Wrap method), 906
wrap_head() (deepmd.tf.nvnmd.entrypoints.Wrap method), 909
wrap_head() (deepmd.tf.nvnmd.entrypoints.wrap.Wrap method), 906
wrap_lut() (deepmd.tf.nvnmd.entrypoints.Wrap method), 909
wrap_lut() (deepmd.tf.nvnmd.entrypoints.wrap.Wrap method), 906
wrap_map() (deepmd.tf.nvnmd.entrypoints.Wrap method), 909
wrap_map() (deepmd.tf.nvnmd.entrypoints.wrap.Wrap method), 906
wrap_up_4() (deepmd.utils.update_sel.BaseUpdateSel method), 1056
wrap_weight() (deepmd.tf.nvnmd.entrypoints.Wrap method), 909
wrap_weight() (deepmd.tf.nvnmd.entrypoints.wrap.Wrap method), 906
write_model_devi_out() (in module deepmd.infer.model_devi), 403
write_model_devi_out() (in module deepmd.tf.infer.model_devi), 819

Z

zero_init() (deepmd.pt.model.network.network.NodeTaskHead method), 570
zero_init() (deepmd.pt.model.network.network.NonLinear method), 562