DeePMD-kit

DeepModeling

Jan 23, 2024
## GETTING STARTED

### 1 Getting Started

1. **Easy install**
   - 1.1 Install off-line packages
   - 1.2 Install with conda
   - 1.3 Install with docker
   - 1.4 Install Python interface with pip

2. **DeepMD-kit Quick Start Tutorial**
   - 2.1 Task
   - 2.2 Table of contents
   - 2.3 Get tutorial data via git
   - 2.4 General Introduction
   - 2.5 Data preparation
   - 2.6 Prepare input script
   - 2.7 Train a model
   - 2.8 Freeze a model
   - 2.9 Test a model
   - 2.10 Run MD with LAMMPS

### 2 Installation

1. **Easy install**
   - 1.1 Install off-line packages
   - 1.2 Install with conda
   - 1.3 Install with docker
   - 1.4 Install Python interface with pip

2. **Install from source code**
   - 2.1 Install the python interface
   - 2.2 Install the C++ interface

3. **Install from pre-compiled C library**
   - 3.1 Use Pre-compiled C Library to build the LAMMPS plugin and GROMACS patch

4. **Install LAMMPS**
   - 4.1 Install LAMMPS’s DeePMD-kit module (built-in mode)
   - 4.2 Install LAMMPS (plugin mode)

5. **Install i-PI**

6. **Install GROMACS with DeepMD**
   - 6.1 Patch source code of GROMACS
   - 6.2 Compile GROMACS with deepmd-kit

7. **Building conda packages**

8. **Install Node.js interface**
   - 8.1 Install from npm
   - 8.2 Build from source
4.11 Fit electronic density of states (DOS) ................................................................. 63
4.11.1 The fitting Network .................................................................. 63
4.11.2 Loss ....................................................................................... 64
4.11.3 Training Data Preparation .......................................................... 64
4.11.4 Train the Model ......................................................................... 64
4.10 Fit tensor like Dipole and Polarizability .................................................... 60
4.10.1 Theory .................................................................................... 60
4.10.2 The fitting Network .................................................................. 61
4.10.3 Loss ....................................................................................... 61
4.10.4 Training Data Preparation .......................................................... 62
4.10.5 Train the Model ......................................................................... 62
4.9 Fit spin energy .................................................................................... 58
4.9.1 Spin ......................................................................................... 59
4.9.2 Spin Loss .................................................................................. 59
4.8 Fit energy .......................................................................................... 56
4.8.1 Theory .................................................................................... 56
4.8.2 The fitting network .................................................................. 57
4.8.3 Loss ....................................................................................... 58
4.7 Determine sel .................................................................................... 56
4.6 Descriptor "hybrid" ........................................................................... 55
4.6.1 Theory .................................................................................... 55
4.6.2 Instructions ............................................................................... 55
4.5 Descriptor "se_atten" ......................................................................... 51
4.5.1 DPA–1: Pretraining of Attention-based Deep Potential Model for Molecular Simulation 51
4.5.2 Theory .................................................................................... 51
4.5.3 Introduction to new features of DPA-1 .......................................... 52
4.5.4 Data format ............................................................................. 54
4.5.5 Training example ....................................................................... 55
4.4 Descriptor "se_e3" ........................................................................... 50
4.4.1 Theory .................................................................................... 50
4.4.2 Instructions ............................................................................... 50
4.3 Descriptor "se_e2_a" ........................................................................ 46
4.3.1 Theory .................................................................................... 47
4.3.2 Instructions ............................................................................... 47
4.2 Descriptor "se_e2_r" ......................................................................... 48
4.2.1 Theory .................................................................................... 48
4.2.2 Instructions ............................................................................... 49
4.1 Overall ............................................................................................. 45
4.1.1 Theory .................................................................................... 45
4.1.2 Instructions ............................................................................... 45
4.0.2 The fitting network .................................................................. 47
4.0.3 Instructions ............................................................................... 47
3.3 Prepare data with dpdata ................................................................. 42
3.2 Formats of a system ........................................................................... 41
3.2.1 NumPy format ......................................................................... 41
3.2.2 HDF5 format ........................................................................... 41
3.2.3 Raw format and data conversion .................................................. 42
3.1 System ............................................................................................. 39
2.9 Easy install the latest development version ............................................. 38
2.9.1 Install with docker .................................................................... 38
2.9.2 Install with pip .......................................................................... 38
2.9.3 Download pre-compiled C Library ............................................... 38
2.8 DPA-1: Pretraining of Attention-based Deep Potential Model for Molecular Simulation 38
2.7 Instructions ..................................................................................... 38
2.6 Theory ............................................................................................... 37
2.5 Introduction to new features of DPA-1 ................................................ 38
2.4 Data format ..................................................................................... 36
2.3 Training example .............................................................................. 37
2.2 The fitting network ............................................................................ 36
2.1 Introduction to new features of DPA-1 ................................................ 37

## 7 Test

7.1 Test a model ......................................................... 147
7.2 Calculate Model Deviation ........................................... 148
   7.2.1 Theory ...................................................... 148
   7.2.2 Instructions ................................................ 148
   7.2.3 Relative model deviation ................................. 149

## 8 Inference

8.1 Python interface .................................................... 151
   8.1.1 External neighbor list algorithm .......................... 151
8.2 C/C++ interface .................................................... 152
   8.2.1 C++ interface .............................................. 152
   8.2.2 C interface ................................................ 153
   8.2.3 Header-only C++ library interface (recommended) .... 154
8.3 Node.js interface .................................................. 155

## 9 Command line interface

9.1 Named Arguments .................................................. 157
9.2 Valid subcommands ............................................... 157
9.3 Sub-commands ..................................................... 157
   9.3.1 transfer .................................................... 157
   9.3.2 train ........................................................ 158
   9.3.3 freeze ....................................................... 159
   9.3.4 test .......................................................... 160
   9.3.5 compress ................................................... 161
   9.3.6 doc-train-input .......................................... 162
   9.3.7 model-devi ................................................ 162
   9.3.8 convert-from ............................................. 163
   9.3.9 neighbor-stat ............................................. 164
   9.3.10 train-nvmd ............................................... 165
   9.3.11 gui ........................................................ 165

## 10 Integrate with third-party packages

10.1 Use deep potential with ASE ..................................... 167
10.2 Run MD with LAMMPS ............................................. 167
   10.2.1 units ....................................................... 167
   10.2.2 Enable DeePMD-kit plugin (plugin mode) ............. 168
   10.2.3 pair_style deepmd ....................................... 168
   10.2.4 Compute tensorial properties ............................ 170
   10.2.5 Long-range interaction .................................. 170
   10.2.6 Use of the centroid/stress/atom to get the full 3x3 “atomic-virial” ............................................. 171
   10.2.7 Computation of heat flux ................................ 171
10.3 Run path-integral MD with i-PI ................................ 172
10.4 Running MD with GROMACS ...................................... 173
   10.4.1 DP/MM Simulation ....................................... 173
   10.4.2 All-atom DP Simulation ................................. 175
10.5 Interfaces out of DeePMD-kit ................................. 176
   10.5.1 dpdata ..................................................... 176
   10.5.2 OpenMM plugin for DeePMD-kit ........................ 176
   10.5.3 Amber interface to DeePMD-kit ......................... 176
   10.5.4 CP2K interface to DeePMD-kit ......................... 176
   10.5.5 DP-GEN .................................................... 177
   10.5.6 MLatom ..................................................... 177
   10.5.7 ABACUS .................................................... 177
   10.5.8 dpdata ..................................................... 176
11 Use NVNMD

11.1 Introduction ......................................................... 179
11.1.1 Training ......................................................... 179
11.1.2 Input script ...................................................... 180
11.1.3 Training ......................................................... 182
11.2 Testing .............................................................. 183
11.3 Running MD in Bohrium .............................................. 183
11.3.1 Registration ...................................................... 183
11.3.2 Top-up and create a project ................................. 184
11.3.3 Run job .......................................................... 185
11.3.4 Check job status ................................................ 186
11.3.5 Terminate and delete jobs ................................... 187
11.3.6 Download Results .............................................. 187
11.4 Running MD in Nvnmd website ..................................... 188
11.4.1 Account application ............................................ 188
11.4.2 Adding task ...................................................... 188
11.4.3 Cancelling calculation ........................................ 190
11.4.4 Downloading results .......................................... 191
11.4.5 Deleting record ................................................ 191
11.4.6 Clearing records ............................................... 192

12 FAQs

12.1 How to tune Fitting/Embedding-net size? ......................... 193
12.1.1 Al2O3 ........................................................... 193
12.1.2 Cu ............................................................... 194
12.1.3 Water ............................................................ 195
12.1.4 Mg-Al ............................................................ 196
12.2 How to control the parallelism of a job? ......................... 196
12.2.1 MPI (optional) ................................................ 196
12.2.2 Parallelism between independent operators ................. 197
12.2.3 Parallelism within an individual operators ................ 197
12.2.4 Tune the performance ...................................... 198
12.3 Do we need to set rcut < half boxsize? ......................... 198
12.4 How to set sel? ..................................................... 198
12.5 Installation .......................................................... 199
12.5.1 Inadequate versions of gcc/g++ .......................... 199
12.5.2 Build files left in DeePMD-kit ....................... 199
12.6 The temperature undulates violently during the early stages of MD 199
12.7 MD: cannot run LAMMPS after installing a new version of DeePMD-kit 199
12.8 Model compatibility ............................................... 200
12.9 Why does a model have low precision? ......................... 200
12.9.1 Data ........................................................... 200
12.9.2 Model .......................................................... 201
12.9.3 Training ......................................................... 201

13 Find DeePMD-kit C/C++ library from CMake .......................... 203

14 Create a model .................................................................. 205
14.1 Design a new component ............................................ 205
14.2 Register new arguments ............................................. 205
14.3 Package new codes .................................................. 206

15 Atom Type Embedding .................................................. 207
15.1 Overview .............................................................. 207
15.2 Preliminary ............................................................ 207
15.3 How to use ................................................................. 208
15.4 Code Modification ..................................................... 208
  15.4.1 trainer (train/trainer.py) ........................................ 208
  15.4.2 model (model/ener.py) ........................................ 208
  15.4.3 embedding net (descriptor/se*.py) ............................ 209
  15.4.4 fitting net (fit/ener.py) ....................................... 209

16 Coding Conventions .................................................. 211
  16.1 Preface ...................................................................... 211
  16.2 Python ................................................................. 211
    16.2.1 Rules .............................................................. 211
    16.2.2 Whitespace ...................................................... 212
    16.2.3 General advice ................................................. 212
    16.2.4 Writing documentation in the code ......................... 212
  16.3 C++ ................................................................. 213
  16.4 Run scripts to check the code .................................... 213

17 CI/CD ......................................................................... 215
  17.1 CI ........................................................................... 215
    17.1.1 Test CUDA ...................................................... 215
  17.2 CD .......................................................................... 215

18 Python API ............................................................... 217
  18.1 backend package ...................................................... 217
    18.1.1 Submodules ...................................................... 217
    18.1.2 backend.dp_backend module ................................ 217
    18.1.3 backend.dynamic_metadata module ....................... 217
    18.1.4 backend.find_tensorflow module ......................... 217
    18.1.5 backend.read_env module ................................... 218
  18.2 deepmd package ...................................................... 219
    18.2.1 Subpackages .................................................... 225
    18.2.2 Submodules ..................................................... 552
    18.2.3 deepmd.calculator module ................................ 552
    18.2.4 deepmd.common module .................................... 554
    18.2.5 deepmd.env module .......................................... 559
    18.2.6 deepmd.lmp module .......................................... 559
  18.3 deepmd-utils package .............................................. 560
    18.3.1 Subpackages .................................................... 560
    18.3.2 Submodules ..................................................... 601
    18.3.3 deepmd-utils.common module .............................. 601
    18.3.4 deepmd-utils.env module .................................. 603
    18.3.5 deepmd-utils.main module ................................ 604

19 OP API ........................................................................ 607
  19.1 op_module ........................................................... 607
  19.2 op_grads_module .................................................... 659

20 C++ API ..................................................................... 669
  20.1 Class Hierarchy ....................................................... 669
  20.2 File Hierarchy ........................................................ 669
  20.3 Full API ............................................................... 670
    20.3.1 Namespaces ...................................................... 670
    20.3.2 Classes and structs ........................................... 672
    20.3.3 Enums ............................................................. 707
    20.3.4 Functions ......................................................... 708
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:


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

- 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 in the Releases page.

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

```
cat deepmd-kit-2.1.1-cuda11.6_gnu-Linux-x86_64.sh.0 deepmd-kit-2.1.1-cuda11.6_gnu-Linux-x86_64.sh
>1 > deepmd-kit-2.1.1-cuda11.6_gnu-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 or Miniconda first.

**Official channel**

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

```
conda create -n deepmd deepmd-kit=2.1.1=cpu libdeepmd=2.1.1=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=2.1.1=cpu libdeepmd=2.1.1=cpu 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 2.1.1 using

```
conda create -n deepmd deepmd-kit=2.1.1=cpu libdeepmd=2.1.1=cpu lammps horovod -c https://conda.deepmodeling.com -c defaults
```

One may enable the environment using

```
conda activate deepmd
```

**conda-forge channel**

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

```
conda create -n deepmd deepmd-kit=2.1.1=cpu lammps horovod -c conda-forge
```

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

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:2.1.1_cpu
```

To pull the GPU version:

```
docker pull ghcr.io/deepmodeling/deepmd-kit:2.1.1_cuda11.6_gpu
```

To pull the ROCm version:

```
docker pull deepmodeling/dpdkit-rocm:dp2.0.3-rocm4.5.2-tf2.6-lmp29Sep2021
```
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]
```

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

To install the package built against CUDA 11.8, use

```
pip install deepmd-kit-cu11[gpu,cu11]
```

Or install the CPU version without CUDA supported:

```
pip install deepmd-kit[cpu]
```

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

```
pip install deepmd-kit[gpu,cu12,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 which is compatible with DeePMD-kit will be also installed.

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

1.2 DeePMD-kit Quick Start Tutorial

DeePMD-kit is a deep learning package for many-body potential energy representation and molecular dynamics.

This tutorial can be directly run on Bohrium Notebook, you can click the Open in Bohrium button above to quickly run this document in Bohrium.

After opening Bohrium Notebook, click the button connect, and choose deepmd-kit:2.2.1-cuda11.6-notebook as image and c4_m8_cpu as computing resources. Wait a minute and you can get started.
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!

1.2.2 Table of contents

• Get tutorial data via git
• General Introduction
• Data preparation
• Prepare input script
• Train a model
• Freeze a model
• Test a model
• Run MD with LAMMPS

1.2.3 Get tutorial data via git

```bash
if ! [ -e colombo-academy-tutorials ]; then git clone https://gitee.com/deepmodeling/colombo-academy-tutorials.git; fi;
```

Cloning into 'colombo-academy-tutorials'...
remote: Enumerating objects: 7164, done.
remote: Counting objects: 100% (174/174), done.
remote: Compressing objects: 100% (138/138), done.
remote: Total 7164 (delta 78), reused 71 (delta 32), pack-reused 6990
Receiving objects: 100% (7164/7164), 45.31 MiB | 3.85 MiB/s, done.
Resolving deltas: 100% (3378/3378), done.
Updating files: 100% (185/185), done.

1.2.4 General Introduction

This tutorial will introduce you to the basic usage of the DeePMD-kit, taking a gas phase methane molecule as an example. DeePMD-kit’s documentation is recommended as the complete reference.

The DP model is generated using the DeePMD-kit package (v2.1.5). The training data is converted into the format of DeePMD-kit using a tool named dpdata (v0.2.14).

Details of dpdata can be found in the dpdata documentation.
We’ve prepared initial data for \( CH_4 \) for you, and put them in the folder `colombo-academy-tutorials/DeePMD-kit/00.data`

```python
import os
prefix_path = os.getcwd()

Folder abacus_md is obtained by performing ab-initio molecular dynamics with ABACUS. Detailed instructions on ABACUS can be found in its document.

```python
os.chdir(os.path.join(prefix_path, "colombo-academy-tutorials", "DeePMD-kit", "00.data"))
```

```python
os.listdir("abacus_md")
```

```python
['C_ONCV_PBE-1.2.upf', 'C_gga_6au_100Ry_2s2p1d.orb', 'H_ONCV_PBE-1.2.upf', 'H_gga_6au_100Ry_2s1p.orb', 'INPUT', 'KPT', 'OUT.ABACUS', 'STRU']
```

### 1.2.5 Data preparation

The training data utilized by DeePMD-kit comprises essential information such as atom type, simulation box, atom coordinate, atom force, system energy, and virial. A snapshot of a molecular system that includes this data is called a frame. Multiple frames with the same number of atoms and atom types make up a system of data. For instance, a molecular dynamics trajectory can be converted into a system of data, with each time step corresponding to a frame in the system.

To simplify the process of converting data generated by popular simulation software like CP2K, Gaussian, Quantum-Espresso, ABACUS, and LAMMPS into the compressed format of DeePMD-kit, we offer a convenient tool called `dpdata`.

Next, the data from AIMD is split randomly as training and validation data.

```python
import dpdata
import numpy as np

# load data of abacus/md format
data = dpdata.LabeledSystem("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)

# all training data put into directory:"training_data"
data_training.to_deepmd_npy("training_data")
```

(continues on next page)
# all validation data put into directory: "validation_data"
data_validation.to_deepmd_npy("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 data.

The DeePMD-kit adopts a compressed data format. All training data should first be converted into this format and can then be used by DeePMD-kit. The data format is explained in detail in the DeePMD-kit manual that can be found in the DeePMD-kit Data Introduction.

```
! tree training_data

training_data
    set.000
        box.npy
        coord.npy
        energy.npy
        force.npy
        virial.npy
        type.raw
        type_map.raw

1 directory, 7 files
```

Let’s have a look at type.raw:

```
! cat training_data/type.raw

0
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 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 doc about Data conversion can be found here.
1.2.6 Prepare input script

Once the data preparation is done, we can go on with training. Now go to the training directory

```python
os.chdir(
    os.path.join(prefix_path, "colombo-academy-tutorials", "DeePMD-kit", "01.train")
)
```

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.

```json
"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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>type_map</td>
<td>the name of each type of atom</td>
</tr>
<tr>
<td>descriptor &gt; type</td>
<td>the type of descriptor</td>
</tr>
<tr>
<td>descriptor &gt; rcut</td>
<td>cut-off radius</td>
</tr>
<tr>
<td>descriptor &gt; rcut_smth</td>
<td>where the smoothing starts</td>
</tr>
<tr>
<td>descriptor &gt; sel</td>
<td>the maximum number of type i atoms in the cut-off radius</td>
</tr>
<tr>
<td>descriptor &gt; neuron</td>
<td>size of the embedding neural network</td>
</tr>
<tr>
<td>descriptor &gt; axis_neuron</td>
<td>the size of the submatrix of G (embedding matrix)</td>
</tr>
<tr>
<td>fitting_net &gt; neuron</td>
<td>size of the fitting neural network</td>
</tr>
</tbody>
</table>

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.

```json
"learning_rate": {
    "type": "exp",
    "decay_steps": 50,
    "start_lr": 0.001,
}
```

(continues on next page)
"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"
  },
  "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](#).

### 1.2.7 Train a model

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

```
! dp train input.json
```

WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/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

(continues on next page)
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:root:Environment variable KMP_BLOCKTIME is empty. Use the default value 0

WARNING:root:Environment variable KMP_AFFINITY is empty. Use the default value granularity=fine,verbose,compact,1,0

DeePMD INFO Calculate neighbor statistics... (add --skip-neighbor-stat to skip this step)

DEEPMD INFO training data with min nbor dist: 1.045920568611028

DEEPMD INFO training data with max nbor size: [4 1]
Please read and cite:

DeePMD-kit
installed to: /home/conda/feedstock_root/build_artifacts/deepmd-kit_
`--1678943793317/work/_skbuild/linux-x86_64-cmake-install`

source : v2.2.1
source branch: HEAD
source commit: 3ac8c4c7
source commit at: 2023-03-16 12:33:24 +0800
build float prec: double
build variant: cuda
build with tf inc: /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/tensorflow/
`include;/opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/tensorflow/../../../../include`

---Summary of the training---------------------------------------
running on: bohrium-14076-1013950
computing device: cpu:0
CUDA_VISIBLE_DEVICES: unset
Count of visible GPU: 0
num_intra_threads: 0
num_inter_threads: 0
---Summary of DataSystem: training ---------------------------------------------
found 1 system(s):
```
  system natoms bch_sz n_bch prob pbc
../00.data/training_data 5 7 23 1.000 T
```
---Summary of DataSystem: validation ---------------------------------------------
found 1 system(s):
```
  system natoms bch_sz n_bch prob pbc
../00.data/validation_data 5 7 5 1.000 T
```
---Summary of DataSystem: training ---------------------------------------------

---Summary of DataSystem: validation ---------------------------------------------

---Summary of DataSystem: training ---------------------------------------------

---Summary of DataSystem: validation ---------------------------------------------

training without frame parameter
data stating... (this step may take long time)
OMP: Info #254: KMP_AFFINITY: pid 118 tid 118 thread 0 bound to OS proc set 0
built lr
built network
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.
initialize model from scratch
start training at lr 1.00e-03 (== 1.00e-03), decay_step 50, decay_rate 0.950006,
---final lr will be 3.51e-08
```python
batch 200 training time 6.10 s, testing time 0.02 s
batch 400 training time 4.83 s, testing time 0.02 s
batch 600 training time 4.84 s, testing time 0.02 s
batch 800 training time 4.85 s, testing time 0.02 s
batch 1000 training time 4.85 s, testing time 0.02 s
saved checkpoint model.ckpt
batch 1200 training time 4.86 s, testing time 0.02 s
batch 1400 training time 5.39 s, testing time 0.02 s
batch 1600 training time 4.94 s, testing time 0.02 s
```
<table>
<thead>
<tr>
<th>Batch</th>
<th>Training Time</th>
<th>Testing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1800</td>
<td>4.86 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>2000</td>
<td>4.84 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>2200</td>
<td>4.86 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>2400</td>
<td>4.90 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>2600</td>
<td>4.87 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>2800</td>
<td>4.84 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>3000</td>
<td>4.86 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>3200</td>
<td>4.86 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>3400</td>
<td>4.99 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>3600</td>
<td>4.88 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>3800</td>
<td>4.85 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>4000</td>
<td>4.88 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>4200</td>
<td>4.88 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>4400</td>
<td>4.86 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>4600</td>
<td>4.92 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>4800</td>
<td>4.86 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>5000</td>
<td>4.86 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>5200</td>
<td>4.87 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>5400</td>
<td>4.88 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>5600</td>
<td>4.87 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>5800</td>
<td>4.87 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>6000</td>
<td>4.90 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>6200</td>
<td>4.86 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>6400</td>
<td>4.87 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>6600</td>
<td>4.86 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>6800</td>
<td>4.84 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>7000</td>
<td>4.93 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>7200</td>
<td>4.89 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>7400</td>
<td>4.88 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>7600</td>
<td>4.88 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>7800</td>
<td>4.87 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>8000</td>
<td>4.86 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>8200</td>
<td>4.87 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>8400</td>
<td>4.85 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>8600</td>
<td>4.86 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>8800</td>
<td>4.87 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>9000</td>
<td>4.83 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>9200</td>
<td>4.87 s</td>
<td>0.02 s</td>
</tr>
</tbody>
</table>
On the screen, you will see the information of the data system(s)

```
DEEPMD INFO found 1 system(s):
      system natoms  bch_sz  n_bch  prob  pbc
./00.data/training_data  5    7    23    1.000    T
```

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
```

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

```
```

(continued from previous page)

```
DEEPMD INFO
batch 9000 training time 4.88 s, testing time 0.02 s
DEEPMD INFO
batch 10000 training time 4.88 s, testing time 0.02 s
DEEPMD INFO
```

```
DEEPMD INFO
```

```
DEEPMD INFO
```

```
DEEPMD INFO
```

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

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

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

with open("lcurve.out") as f:
    headers = f.readline().split()[1:]
lcurve = pd.DataFrame(np.loadtxt("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()
```
1.2.8 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

```bash
! dp freeze -o graph.pb
```

(continues on next page)
DeePMD-kit

The following nodes will be frozen: ['model_type', 'descrpt_attr/rcut', 'descrpt_attr/ntypes', 'model_attr/tmap', 'model_attr/model_type', 'model_attr/model_version', 'train_attr/min_nbor_dist', 'train_attr/training_script', 'o_energy', 'o_force', 'o_virial', 'o_atom_energy', 'o_atom_virial', 'fitting_attr/dfparam', 'fitting_attr/daparam']

WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/deepmd/entrypoints/...freeze.py:354: 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 /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/deepmd/entrypoints/...freeze.py:354: 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 /opt/deepmd-kit-2.2.1/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 /opt/deepmd-kit-2.2.1/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 1211 ops in the final graph.

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

### 1.2.9 Test a model

We can check the quality of the trained model by running

```
$ dp test -m graph.pb -s ../00.data/validation_data
```

WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/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_PARALELISM_THREADS, and TF_INTER_OP_PARAllelISM_THREADS. See https://deepmd.rtfd.io/parallelism/ for more information.

WARNING:root:Environment variable KMP_BLOCKTIME is empty. Use the default value 0

WARNING:root:Environment variable KMP_AFFINITY is empty. Use the default value granularity=fine,verbose,compact,1,0

/opt/deepmd-kit-2.2.1/lib/python3.10/importlib/__init__.py:169: UserWarning: The NumPy module was reloaded (imported a second time). This can in some cases result in small but subtle issues and is discouraged.

_bootstrap._exec(spec, module)


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

```python
import dpdata

training_systems = dpdata.LabeledSystem("../00.data/training_data", fmt="deepmd/npy")
```
predict = training_systems.predict("graph.pb")

2023-04-20 23:40:32.104716: 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: AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-04-20 23:40:34.426193: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: cannot open shared object file: No such file or directory; LD_LIBRARY_PATH: /usr/local/nvidia/lib:/usr/local/nvidia/lib64
2023-04-20 23:40:34.427318: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libnvinfer_plugin.so.7'; dlerror: libnvinfer_plugin.so.7: cannot open shared object file: No such file or directory; LD_LIBRARY_PATH: /usr/local/nvidia/lib:/usr/local/nvidia/lib64
2023-04-20 23:40:34.427332: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot dlopen some TensorRT libraries. If you would like to use Nvidia GPU with TensorRT, please make sure the missing libraries mentioned above are installed properly.

WARNING:tensorflow:From /opt/mamba/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 /opt/mamba/lib/python3.10/site-packages/deepmd/utils/batch_size.py:61: 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.

2023-04-20 23:40:36.161142: 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: AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-04-20 23:40:36.181810: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:357] MLIR V1 optimization pass is not enabled
WARNING:tensorflow:From /opt/mamba/lib/python3.10/site-packages/deepmd/utils/batch_size.py:61: 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.

1.2. DeePMD-kit Quick Start Tutorial
```python
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()
```

### 1.2.10 Run MD with LAMMPS

The model can drive molecular dynamics in LAMMPS.

```bash
! cd ../02.lmp && cp ../01.train/graph.pb ./ && ls

conf.lmp graph.pb in.lammps
```

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 compatible version of LAMMPS, the deep potential molecular dynamics can be performed via

lmp -i input.lammps

\! cd ../02.lmp \&\& cp ../01.train/graph.pb ./ \&\& lmp -i in.lammps

Warning:
This LAMMPS executable is in a conda environment, but the environment has not been activated. Libraries may fail to load. To activate this environment please see https://conda.io/activation.

LAMMPS (23 Jun 2022 - 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 /opt/deepmd-kit-2.2.1/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 = 0.004 seconds
Summary of lammps deepmd module ...
>> Info of deepmd-kit:
  installed to: /opt/deepmd-kit-2.2.1
  source: v2.2.1
  source branch: HEAD
  source commit: 3ac8c4c7
  source commit at: 2023-03-16 12:33:24 +0800
  surpport model ver.:1.1
  build variant: cuda
  build with tf inc: /opt/deepmd-kit-2.2.1/include;/opt/deepmd-kit-2.2.1/include
  build with tf lib: /opt/deepmd-kit-2.2.1/lib/libtensorflow_cc.so
  set tf intra_op_parallelism_threads: 0
  set tf inter_op_parallelism_threads: 0
>> Info of lammps module:
DeePMD-kit: Successfully load libcudart.so
2023-04-20 23:40:39.637091: 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.
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:
The log file lists these citations in BibTeX format.

Generated 0 of 1 mixed pair_coeff terms from geometric mixing rule

Neighbor list info ...
  update every 10 steps, delay 0 steps, check no
  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) = 3.809 | 3.809 | 3.809 Mbytes

<table>
<thead>
<tr>
<th>Step</th>
<th>PotEng</th>
<th>KinEng</th>
<th>TotEng</th>
<th>Temp</th>
<th>Press</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-219.77011</td>
<td>0.025852029</td>
<td>-219.74426</td>
<td>50</td>
<td>-810.10259</td>
<td>1060.5429</td>
</tr>
<tr>
<td>100</td>
<td>-219.76784</td>
<td>0.023303362</td>
<td>-219.74454</td>
<td>45.070664</td>
<td>-605.50113</td>
<td>1060.5429</td>
</tr>
<tr>
<td>200</td>
<td>-219.77863</td>
<td>0.032400378</td>
<td>-219.74622</td>
<td>62.665059</td>
<td>53.929107</td>
<td>1060.5429</td>
</tr>
<tr>
<td>300</td>
<td>-219.77403</td>
<td>0.027115352</td>
<td>-219.74692</td>
<td>52.443373</td>
<td>642.24342</td>
<td>1060.5429</td>
</tr>
<tr>
<td>400</td>
<td>-219.77126</td>
<td>0.023079501</td>
<td>-219.74818</td>
<td>44.637697</td>
<td>861.365</td>
<td>1060.5429</td>
</tr>
<tr>
<td>500</td>
<td>-219.786</td>
<td>0.034433001</td>
<td>-219.75156</td>
<td>66.596322</td>
<td>256.47994</td>
<td>1060.5429</td>
</tr>
<tr>
<td>600</td>
<td>-219.78295</td>
<td>0.029039598</td>
<td>-219.75391</td>
<td>56.165027</td>
<td>-527.21506</td>
<td>1060.5429</td>
</tr>
<tr>
<td>700</td>
<td>-219.777</td>
<td>0.020227709</td>
<td>-219.75677</td>
<td>39.12091</td>
<td>-696.11258</td>
<td>1060.5429</td>
</tr>
<tr>
<td>800</td>
<td>-219.78394</td>
<td>0.02289598</td>
<td>-219.75391</td>
<td>44.277408</td>
<td>-77.227892</td>
<td>1060.5429</td>
</tr>
<tr>
<td>900</td>
<td>-219.77998</td>
<td>0.015506508</td>
<td>-219.76447</td>
<td>29.990893</td>
<td>663.84491</td>
<td>1060.5429</td>
</tr>
<tr>
<td>1000</td>
<td>-219.78328</td>
<td>0.015178419</td>
<td>-219.76105</td>
<td>39.12091</td>
<td>663.84491</td>
<td>1060.5429</td>
</tr>
<tr>
<td>1100</td>
<td>-219.7903</td>
<td>0.018763273</td>
<td>-219.77154</td>
<td>36.28975</td>
<td>273.19351</td>
<td>1060.5429</td>
</tr>
<tr>
<td>1200</td>
<td>-219.78639</td>
<td>0.012922048</td>
<td>-219.77347</td>
<td>24.992328</td>
<td>577.90459</td>
<td>1060.5429</td>
</tr>
<tr>
<td>1300</td>
<td>-219.79131</td>
<td>0.015848131</td>
<td>-219.77546</td>
<td>30.65162</td>
<td>-129.85247</td>
<td>1060.5429</td>
</tr>
<tr>
<td>1400</td>
<td>-219.78829</td>
<td>0.011969602</td>
<td>-219.77632</td>
<td>23.150218</td>
<td>545.58517</td>
<td>1060.5429</td>
</tr>
<tr>
<td>1500</td>
<td>-219.78735</td>
<td>0.010610097</td>
<td>-219.77674</td>
<td>20.520921</td>
<td>356.36805</td>
<td>1060.5429</td>
</tr>
</tbody>
</table>

(continues on next page)
Loop time of 12.1251 on 1 procs for 5000 steps with 5 atoms

Performance: 35.629 ns/day, 0.674 hours/ns, 412.369 timesteps/s
242.0% CPU use with 1 MPI tasks x 1 OpenMP threads

MPI task timing breakdown:
Section | min time | avg time | max time | %varavg | %total
---------------------------------------------------------------
Pair | 12.072 | 12.072 | 12.072 | 0.0 | 99.56
Negh | 0.0066181 | 0.0066181 | 0.0066181 | 0.0 | 0.05
Comm | 0.012792 | 0.012792 | 0.012792 | 0.0 | 0.11
Output | 0.0044695 | 0.0044695 | 0.0044695 | 0.0 | 0.04
Modify | 0.022737 | 0.022737 | 0.022737 | 0.0 | 0.19
Other | 0.006263 | | | 0.0 | 0.05

Nlocal: 5 ave 5 max 5 min
Histogram: 1 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

Neigh: 0 ave 0 max 0 min
Histogram: 1 0 0 0 0 0 0 0 0

(continues on next page)
<table>
<thead>
<tr>
<th>FullNghs:</th>
<th>20 ave</th>
<th>20 max</th>
<th>20 min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Histogram:</td>
<td>1 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total # of neighbors = 20
Ave neighs/atom = 4
Neighbor list builds = 500
Dangerous builds not checked
Total wall time: 0:00:13
2.1 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: 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.

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

2.1.1 Install off-line packages

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

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

```
cat deepmd-kit-2.1.1-cuda11.6_gpu-Linux-x86_64.sh.0 deepmd-kit-2.1.1-cuda11.6_gpu-Linux-x86_64.sh.1 > deepmd-kit-2.1.1-cuda11.6_gpu-Linux-x86_64.sh
```

One may enable the environment using

```
conda activate /path/to/deepmd-kit
```
2.1.2 Install with conda

DeePMD-kit is available with conda. Install Anaconda or Miniconda first.

Official channel

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

```bash
conda create -n deepmd deepmd-kit=2.1.1=cpu libdeepmd=2.1.1=cpu lammps -c https://conda.deepmodeling.com, defaults
```

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

```bash
conda create -n deepmd deepmd-kit=2.1.1=cpu libdeepmd=2.1.1=cpu lammps cudatoolkit=11.6 horovod -c https://conda.deepmodeling.com, defaults
```

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

One may specify the DeePMD-kit version such as 2.1.1 using

```bash
conda create -n deepmd deepmd-kit=2.1.1=cpu libdeepmd=2.1.1=cpu lammps horovod -c https://conda.deepmodeling.com, defaults
```

One may enable the environment using

```bash
conda activate deepmd
```

conda-forge channel

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

```bash
conda create -n deepmd deepmd-kit=2.1.1 lammps horovod -c conda-forge
```

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

2.1.3 Install with docker

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

To pull the CPU version:

```bash
docker pull ghcr.io/deepmodeling/deepmd-kit:2.1.1_cpu
```

To pull the GPU version:

```bash
docker pull ghcr.io/deepmodeling/deepmd-kit:2.1.1_cuda11.6_gpu
```

To pull the ROCm version:

```bash
docker pull deepmodeling/dpmdkit-rocm:dp2.0.3-rocm4.5.2-tf2.6-lmp29Sep2021
```
2.1.4 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[cpu,cu12]
```

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

To install the package built against CUDA 11.8, use

```
pip install deepmd-kit-cu11[cpu,cu11]
```

Or install the CPU version without CUDA supported:

```
pip install deepmd-kit[cpu]
```

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

```
pip install deepmd-kit[cpu,cu12,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 which is compatible with DeePMD-kit will be also installed.

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

2.2 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`
```
2.2.1 Install the python interface

Install Tensorflow’s python interface

First, check the python version on your machine

```
python --version
```

We follow the virtual environment approach to install TensorFlow’s Python interface. The full instruction can be found on the official TensorFlow website. TensorFlow 1.8 or later is supported. Now we assume that the Python interface will be installed to the virtual environment directory `$tensorflow_venv`

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

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 $tensorflow_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 $tensorflow_venv
```

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
```

To verify the installation, run

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

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

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

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. 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)"`.

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

<table>
<thead>
<tr>
<th>Environment variables</th>
<th>Allowed value</th>
<th>Default value</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP_VARIANT</td>
<td>cpu, cuda, rocm</td>
<td>cpu</td>
<td>Build CPU variant or GPU variant with CUDA or ROCM support.</td>
</tr>
<tr>
<td>CUDA-Toolkit_ROOT</td>
<td>Path</td>
<td>Detected automatically</td>
<td>The path to the CUDA toolkit directory. CUDA 9.0 or later is supported. NVCC is required.</td>
</tr>
<tr>
<td>ROCM_ROOT</td>
<td>Path</td>
<td>Detected automatically</td>
<td>The path to the ROCM toolkit directory.</td>
</tr>
<tr>
<td>TENSORFLOW_ROOT</td>
<td>Path</td>
<td>Detected automatically</td>
<td>The path to TensorFlow Python library. By default the installer only finds TensorFlow under user site-package directory (<code>site.getusersitepackages()</code>) or system site-package directory (<code>sysconfig.get_path(&quot;purelib&quot;)</code>) due to limitation of PEP-517. If not found, the latest TensorFlow (or the environment variable <code>TENSORFLOW_VERSION</code> if given) from PyPI will be built against.</td>
</tr>
<tr>
<td>DP_ENABLE_NATIVE_OPTIMIZATION</td>
<td>0, 1</td>
<td>0</td>
<td>Enable compilation optimization for the native machine’s CPU type. Do not enable it if generated code will run on different CPUs.</td>
</tr>
<tr>
<td>CMAKE_ARGS</td>
<td>str</td>
<td>-</td>
<td>Additional CMake arguments</td>
</tr>
<tr>
<td>&lt;LANG&gt;FLAGS</td>
<td>-</td>
<td>Default compilation flags to be used when compiling <code>&lt;LANG&gt;</code> files. See CMake documentation.</td>
<td></td>
</tr>
</tbody>
</table>

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} ...
```

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.
2.2.2 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 Tensorflow’s C++ interface (optional)

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.

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
```

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

```
cmake -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}`.

One may add the following arguments to `cmake`:
<table>
<thead>
<tr>
<th>CMake Arguments</th>
<th>Allowed value</th>
<th>Default value</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>-DTensorFlow_ROOT</td>
<td>Path</td>
<td>-</td>
<td>The Path to TensorFlow’s C++ interface.</td>
</tr>
<tr>
<td>-DCMAKE_INSTALL_PREFIX</td>
<td>Path</td>
<td>-</td>
<td>The Path where DeePMD-kit will be installed.</td>
</tr>
<tr>
<td>-DUSE_CUDA_TOOLKIT</td>
<td>TRUE or FALSE</td>
<td>FALSE</td>
<td>If TRUE, Build GPU support with CUDA toolkit.</td>
</tr>
<tr>
<td>-DCUDAToolkit_ROOT</td>
<td>Path</td>
<td>Detected auto-</td>
<td>The path to the CUDA toolkit directory. CUDA 9.0 or later is supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matically</td>
<td>NVCC is required.</td>
</tr>
<tr>
<td>-DUSE_ROCM_TOOLKIT</td>
<td>TRUE or FALSE</td>
<td>FALSE</td>
<td>If TRUE, Build GPU support with ROCM toolkit.</td>
</tr>
<tr>
<td>-DCMAKE_HIP_COMPILER_ROCM_ROOT</td>
<td>Path</td>
<td>Detected auto-</td>
<td>The path to the ROCM toolkit directory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>matically</td>
<td></td>
</tr>
<tr>
<td>-DLAMMPS_SOURCE_ROOT</td>
<td>Path</td>
<td>-</td>
<td>Only necessary 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.</td>
</tr>
<tr>
<td>-DUSE_TF_PYTHON_LIBS</td>
<td>TRUE or FALSE</td>
<td>FALSE</td>
<td>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.</td>
</tr>
<tr>
<td>-DENABLE_NATIVE_OPTIMIZATION</td>
<td>TRUE or FALSE</td>
<td>FALSE</td>
<td>Enable compilation optimization for the native machine’s CPU type. Do not enable it if generated code will run on different CPUs.</td>
</tr>
<tr>
<td>-DCMAKE_&lt;LANG&gt;_FLAGS</td>
<td>str</td>
<td>-</td>
<td>Default compilation flags to be used when compiling &lt;LANG&gt; files. See CMake documentation.</td>
</tr>
</tbody>
</table>

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

```bash
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
```
2.3 Install from pre-compiled C library

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.

2.3.1 Use Pre-compiled C Library to build the LAMMPS plugin 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 $deepmd_source_dir/source
mkdir build
mkdir build
mkdir build
make -DDEEPMD_C_ROOT=/path/to/libdeepmd_c -DCMAKE_INSTALL_PREFIX=$deepmd_root ..
makes
make install
```

Then one can follow the manual Install LAMMPS and/or Install GROMACS.

2.4 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.

2.4.1 Install LAMMPS’s DeePMD-kit module (built-in mode)

Before following this section, DeePMD-kit C++ interface should have been 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 lammmps
```

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
tar xf stable_2Aug2023_update2.tar.gz
```

2.3. Install from pre-compiled C library
The source code of LAMMPS is stored in the directory `lammps-stable_2Aug2023_update2`. 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:

```bash
cd lammps-stable_2Aug2023_update2/src/
cp -r $deepmd_source_dir/source/build/USER-DEEPMD .
makexyes-kspace
makexyes-extra-fix
makexyes-user-deepmd
```

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

```bash
makempij4
```

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

```bash
./lmp_mpi -h
```

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

```bash
makeno-user-deepmd
```

**With CMake**

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

```bash
mkdir -p lammps-stable_2Aug2023_update2/build/
cd lammps-stable_2Aug2023_update2/build/
```

Patch the LAMMPS `CMakeLists.txt` file:

```bash
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.

```bash
cmakewithout -D LAMMPS_INSTALL_RPATH=ON -D BUILD_SHARED_LIBS=yes -D CMAKE_INSTALL_PREFIX=$deepmd_root -D CMAKE_PREFIX_PATH=$deepmd_root ../cmake
makewithout -j4
makewithout install
```

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

```bash
$deepmd_root/bin/lmp -h
```
2.4.2 Install LAMMPS (plugin mode)

Starting from 8 Apr 2021, LAMMPS also provides a plugin mode, allowing one to build LAMMPS and a plugin separately.

Now download the LAMMPS code (8 Apr 2021 or later), and uncompress it:

```
cd /some/workspace
tar xf stable_2Aug2023_update2.tar.gz
```

The source code of LAMMPS is stored in the directory `lammps-stable_2Aug2023_update2`. 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_update2/build/
cd lammps-stable_2Aug2023_update2/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`:

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

2.5 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
```
2.6 Install GROMACS with DeepMD

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

2.6.1 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:

```bash
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.

2.6.2 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:

```bash
#!/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
```

2.7 Building conda packages

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
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 them executing

```bash
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:

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

### 2.8 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.

#### 2.8.1 Install from npm

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

#### 2.8.2 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):

```bash
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:

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

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

```javascript
const deepmd = require("deepmd-kit");
```
2.9 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.

2.9.1 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.

2.9.2 Install with pip

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

```
```

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

2.9.3 Download pre-compiled C Library

The pre-compiled 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
```
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.

### 3.1 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_attn.

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

<table>
<thead>
<tr>
<th>ID</th>
<th>Property</th>
<th>Raw file</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Atom type indexes</td>
<td>type</td>
<td>Required</td>
<td>N atom 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 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)</td>
</tr>
<tr>
<td>type</td>
<td>name</td>
<td>type</td>
<td>Optional</td>
<td>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</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>ID</th>
<th>Property</th>
<th>Raw file</th>
<th>Unit</th>
<th>Shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coord</td>
<td>Atomic coordinates</td>
<td>coord.raw</td>
<td>Å</td>
<td>Nframes * 3</td>
<td>Required</td>
</tr>
<tr>
<td>box</td>
<td>Boxes</td>
<td>box.raw</td>
<td>Å</td>
<td>Nframes * 3</td>
<td>Required if periodic</td>
</tr>
<tr>
<td>tparam</td>
<td>Extra frame parameters</td>
<td>tparam.raw</td>
<td>Any</td>
<td>Nframes</td>
<td>Optional</td>
</tr>
<tr>
<td>aparam</td>
<td>Extra atomic parameters</td>
<td>aparam.raw</td>
<td>Any</td>
<td>Nframes</td>
<td>Optional</td>
</tr>
<tr>
<td>numb_copy</td>
<td>Each frame is copied by the numb_copy (int) times</td>
<td>prob.raw1</td>
<td>Any</td>
<td>Nframes</td>
<td>Optional</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>ps</td>
</tr>
<tr>
<td>Length</td>
<td>Å</td>
</tr>
<tr>
<td>Energy</td>
<td>eV</td>
</tr>
<tr>
<td>Force</td>
<td>eV/A</td>
</tr>
<tr>
<td>Virial</td>
<td>eV</td>
</tr>
<tr>
<td>Pressure</td>
<td>Bar</td>
</tr>
</tbody>
</table>
3.2 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.

3.2.1 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  
s000/coord.npy 
s000/energy.npy  
s000/force.npy  
s001/coord.npy  
s001/energy.npy  
s001/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
0 H
```

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

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

3.2.2 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.
3.2.3 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
line 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.

3.3 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

```python
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.
4.1 Overall

4.1.1 Theory

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

$$y_i = \mathcal{M}(x_i, \{x_j\}_{j \in n(i)}; \theta) = \mathcal{F}(\mathcal{D}(x_i, \{x_j\}_{j \in n(i)}; \theta_d); \theta_f),$$

where $y_i$ is the fitting properties, $\mathcal{F}$ is the fitting network, $\mathcal{D}$ is the descriptor. $x = (r_i, \alpha_i)$, with $r_i$ being the Cartesian coordinates and $\alpha_i$ 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 $\theta = \{\theta_d, \theta_f\}$, where $\theta_d$ and $\theta_f$ 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

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

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.\(^1\)

4.1.2 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, Qiuyu Zeng, Han Bao, Yu Xia, Jianeng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Leland 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 Wentzovitch, Jiequun 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.
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**: Concatenate 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.

### 4.2 Descriptor "se_e2_a"

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).
4.2.1 Theory

The two-body embedding smooth edition of the DP descriptor $D^i \in \mathbb{R}^{M \times M_c}$, 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

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

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 $N_c$. $R^i \in \mathbb{R}^{N_c \times 4}$ is the coordinate matrix, and each row of $R^i$ can be constructed as

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

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

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

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.

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

$$\left( G^i \right)_j = N_{c,2}(s(r_{ij})), $$

where the subscript $c, 2$ 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. $G^i < \in \mathbb{R}^{N_c \times M_c}$ only takes first $M_c$ columns of $G^i$ to reduce the size of $D^i$. $r_s, r_c, M$ and $M_c$ are hyperparameters provided by the user. The DeepPot-SE is continuous up to the second-order derivative in its domain.\(^1\)

4.2.2 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, 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 Leland Bore, Chunj Cui, Yimin 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, Jeequin 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.

4.3 Descriptor "se_e2_r"

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.

4.3.1 Theory

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

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

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

\[
(G^i)_j = \mathcal{N}_{g,2}(s(r_{ij})),
\]

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

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

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.\(^1\)

### 4.3.2 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 \texttt{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],
    "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, Yinxiao Chen, Marián Rynik, Li’ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tao, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jianeng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sighjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rihys 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.
4.4 Descriptor "se_e3"

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).

4.4.1 Theory

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

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

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

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

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

$$(G^i)_{jk} = N_{e,3}((\theta_{ij})_{jk}),$$

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

4.4.2 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,
   "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, Qiuyu Zeng, Han Bao, Yu Xia, Jianneng Huang, Koki Murakoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Leland 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 Wentzovitch, Jiequn Tan, Wei Li, 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.
4.5 Descriptor "se_atten"

4.5.1 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.

4.5.2 Theory

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

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

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

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

$$\left( Q^{i,l} \right)_j = Q_l \left( (G^{i,l-1})_j \right),$$

$$\left( K^{i,l} \right)_j = K_l \left( (G^{i,l-1})_j \right),$$

$$\left( V^{i,l} \right)_j = V_l \left( (G^{i,l-1})_j \right),$$

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 $G^{i,0}$, is chosen as the two-body embedding matrix.

Then the scaled dot-product attention method is adopted:

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

\[
\varphi(\mathbf{Q}^{i,l}, \mathbf{K}^{i,l}, \mathbf{R}^{i,l}) = \text{softmax}(\frac{\mathbf{Q}^{i,l}(\mathbf{K}^{i,l})^T}{\sqrt{d_k}}) \odot \hat{\mathbf{R}}^i(\hat{\mathbf{R}}^i)^T,
\]

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

Then layer normalization is added in a residual way to finally obtain the self-attention local embedding matrix \( \mathbf{G}^{i,L} \) after \( L_a \) attention layers:

\[
\mathbf{G}^{i,l} = \mathbf{G}^{i,l-1} + \text{LayerNorm}(A(\mathbf{Q}^{i,l}, \mathbf{K}^{i,l}, \mathbf{V}^{i,l}, \mathbf{R}^{i,l})).
\]

### 4.5.3 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:

\[$\text{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

```json
"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
}
```

---

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, Qiuyu Zeng, Han Bao, Yu Xia, Jianeng Huang, Koki Muraoka, Yibo Wang, Junhan Zhang, Fengbo Yuan, Sigbjørn Leland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayun Xu, Jia-Xin Zhu, Chenxing Luo, Yuji Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jieyun 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.
• 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.

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:

```json
"stripped_type_embedding": true,
"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".

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:

```json
"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:

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

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

### 4.5.4 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:

<table>
<thead>
<tr>
<th>ID</th>
<th>Property</th>
<th>File</th>
<th>Required/Optional</th>
<th>Shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Atom type indexes (placeholder)</td>
<td>type.raw</td>
<td>Required</td>
<td>Natoms</td>
<td>All zeros to fake the type input</td>
</tr>
<tr>
<td>type</td>
<td>Atom type names</td>
<td>type_map.raw</td>
<td>Required</td>
<td>Ntypes</td>
<td>Atom names that map to atom type contained in all the frames, which is unnecessary to be contained in the periodic table</td>
</tr>
<tr>
<td>type</td>
<td>Atom type indexes of each frame</td>
<td>real_atom_types.npy</td>
<td>Required*</td>
<td>Natoms</td>
<td>Integers that describe atom types in each frame, corresponding to indexes in type_map. -1 means virtual atoms.</td>
</tr>
</tbody>
</table>

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_attn descriptor.
To put frames with different \(N_{\text{atoms}}\) 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 \textit{mixed_type} format is available on \texttt{dpdata} for a more convenient experience.

### 4.5.5 Training example

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

### 4.6 Descriptor "hybrid"

This descriptor hybridizes multiple descriptors to form a new descriptor. For example, we have a list of descriptors denoted by \(D_1, D_2, \ldots, D_N\), the hybrid descriptor this the concatenation of the list, i.e. \(D = (D_1, D_2, \ldots, D_N)\).

#### 4.6.1 Theory

A hybrid descriptor \(D_{\text{hyb}}^i\) concatenates multiple kinds of descriptors into one descriptor:

\[
D_{\text{hyb}}^i = \{D_1^i, D_2^i, \ldots, D_N^i\}.
\]

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.\(^1\)

#### 4.6.2 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 \texttt{list},

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

\(^1\) 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, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiuyu Zeng, Han Bao, Yu Xia, Jianeng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Leland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayuan 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.
A complete training input script of this example can be found in the directory

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

### 4.7 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 not be 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
```

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.

### 4.8 Fit energy

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

#### 4.8.1 Theory

In the DP model, we let the fitting network $F_0$ maps the descriptor $D^i$ 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

$$E = \sum_i E_i = \sum_i F_0(D^i),$$

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 $P^i \in \mathbb{R}^{N_p}$ (with $N_p$ being the dimension) and $Q \in \mathbb{R}^{N_q}$ (with $N_q$ being the dimension), respectively.

$$E_i = F_0([D^i, P^i, Q]).$$

The atomic force $F_i$ and the virial tensor $\Xi = (\Xi_{\alpha\beta})$ (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 \( \alpha \)-th component of the coordinate and force of atom \( i \). \( h_{\alpha\beta} \) is the \( \beta \)-th component of the \( \alpha \)-th basis vector of the simulation region.

The properties \( \eta \) of the energy loss function could be energy \( E \), force \( F \), virial \( \Xi \), relative energy \( \Delta E \), or any combination among them, and the loss functions of them are

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

\[ L_F(x; \theta) = \frac{1}{3N} \sum_{k=1}^{N} \sum_{\alpha=1}^{3} (F_{k,\alpha}(x; \theta) - F_{k,\alpha}^*)^2, \]

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

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

where \( F_{k,\alpha} \) is the \( \alpha \)-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:

\[ L_p^F(x; \theta) = \frac{1}{3N} \sum_{k=1}^{N} \sum_{\alpha} q_k (F_{k,\alpha}(x; \theta) - F_{k,\alpha}^*)^2. \]

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:

\[ L_r^F(x; \theta) = \frac{1}{3N} \sum_{k=1}^{N} \sum_{\alpha} \left( \frac{F_{k,\alpha}(x; \theta) - F_{k,\alpha}^*}{|F_{k,\alpha}^*| + \nu} \right)^2. \]

where \( \nu \) is a small constant used to protect an atom where the magnitude of \( F_{k,\alpha}^* \) is small from having a large \( L_r^F \). Benefiting from the relative force loss, small forces can be fitted more accurately.\(^1\)

### 4.8.2 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
},
```

\(^1\) 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, Qiuyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Leland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Riyhs E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weiman E, Roberto Car, Linfeng Zhang, Han Wang. J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.
DeePMD-kit

- 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.

4.8.3 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 prefactors 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 $p_f^\infty$ 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.

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

The loss section in the input.json is

```json
"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.

4.9 Fit spin energy

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

The construction of the fitting net is given by section spin

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

- `use_spin` determines whether to turn on the magnetism of the atoms. 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.

4.9.2 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, magnetic force and virial, respectively. $p_e$, $p_{fr}$, $p_{fm}$ and $p_v$ give the prefactors of the energy, atomic force, magnetic force and virial losses.

The prefactors 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 $p_{fr}^\infty$ specifies the $p_{fr}$ at the start of the training and at the limit of $t \to \infty$ (set by `start_pref_fr` and `limit_pref_fr`, respectively), i.e.

$$p_{fr}(t) = start_{pref_{fr}} * lr(t)/start_{lr} + limit_{pref_{fr}} * (1-lr(t)/start_{lr})$$

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, magnetic 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.

4.9. Fit spin energy
4.10 Fit tensor like Dipole and Polarizability

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

$\text{deepmd_source_dir}/examples/water_tensor/dipole/dipole_input.json
$\text{deepmd_source_dir}/examples/water_tensor/polar/polar_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 a tensor, one needs to modify model/fitting_net and loss.

4.10.1 Theory

To represent the first-order tensorial properties (i.e. vector properties), we let the fitting network, denoted by $F_1$, output an $M$-dimensional vector; then we have the representation,

$$T^{(1)}_i(\alpha) = \frac{1}{N_c} \sum_{j=1}^{N_c} \sum_{m=1}^{M} (G^i)_jm (R^i)_j,\alpha + 1 (F_1(D^i)_m), \alpha = 1, 2, 3.$$

We let the fitting network $F_2$ output an $M$-dimensional vector, and the second-order tensorial properties (matrix properties) are formulated as

$$T^{(2)}_i(\alpha, \beta) = \frac{1}{N_c^2} \sum_{j=1}^{N_c} \sum_{k=1}^{N_c} \sum_{m=1}^{M} (G^i)_jm (R^i)_j,\alpha + 1 (R^i)_k,\beta + 1 (G^i)_km (F_2(D^i)_m), \alpha, \beta = 1, 2, 3,$$

where $G^i$ and $R^i$ 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. $F_1$ and $F_2$ are the neural network functions. The total tensor $T$ (total dipole $T^{(1)}$ or total polarizability $T^{(2)}$) is the sum of the atomic tensor:

$$T = \sum_i T_i.$$

The tensorial models can be used to calculate IR spectrum and Raman spectrum.\footnote{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 Tao, Jiabin Yang, Ye Ding, Yifan Li, Davide Tsi, Qiyu Zeng, Han Bao, Yu Xia, Jianmeng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayin 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.}
4.10.2 The fitting Network

The fitting_net section tells DP which fitting net to use.

The JSON of dipole type should be provided like

```json
"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

```json
"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.

4.10.3 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 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, YY, YZ, ZX, 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

```json
"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 cannot be left unset. If set to 0, the corresponding label will NOT be included in the training process.

### 4.10.4 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.

### 4.10.5 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`:

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

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
```

(continues on next page)
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.

### 4.11 Fit electronic density of states (DOS)

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`.

#### 4.11.1 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.
4.11.2 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 \(1 \times \text{numb\_dos}\) 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 \(\text{natom} \times \text{numb\_dos}\) matrices.

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

\[
\text{loss} = \text{pref} \times \text{global\_loss} + \text{pref\_atomic} \times \text{atomic\_loss}
\]

The loss section should be provided like

```json
"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

\[
D(\epsilon) = \int_{\epsilon_{\text{min}}}^{\epsilon} g(\epsilon')d\epsilon'
\]

4.11.3 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.

4.11.4 Train the Model

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

```bash
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
```
4.11.5 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
0.00000000000000000000e+00 1.9631932649176465342e-03
0.00000000000000000000e+00 1.1784408367813137274e-03
0.00000000000000000000e+00 1.4412580717904077969e-04
0.00000000000000000000e+00 1.7872979331405817432e-03
0.00000000000000000000e+00 1.9016032802430249401e-03
0.00000000000000000000e+00 2.279848925571981155e-03
0.00000000000000000000e+00 2.1493585466856160736e-03
0.00000000000000000000e+00 1.8298485187260566073e-03
0.00000000000000000000e+00 1.9051565124197922255e-03
```
4.12 Type embedding approach

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.

4.12.1 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 $N_{e,(2,3)}$ are as follows chemical-species-wise:

\[
(G^i)_j = N_{e,2}^{\alpha_i,\alpha_j}(s(r_{ij})) \quad \text{or} \quad (G^i)_j = N_{e,2}^{\alpha_j}(s(r_{ij})),
\]

\[
(G^i)_{jk} = N_{e,3}^{\alpha_j,\alpha_k}((\theta_i)_{jk}).
\]

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 $N_e$ is divided into corresponding blocks, and each block is padded to $N_e^{\alpha_j}$ 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 $\alpha_i$ in advance. Take an example, the atomic energy $E_i$ is represented as follows:

\[
E_i = F_0^{\alpha_i}(D^i).
\]

To reduce the number of NN parameters and improve computing efficiency when there are large numbers of chemical species, the type embedding $A^i$ is introduced, represented as a NN function $N_t$ of the atomic type $\alpha$:

\[
A^i = N_t(\text{one hot} (\alpha_i)),
\]

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

\[
(G^i)_j = N_{e,2}(\{s(r_{ij}), A^i, A^j\}) \quad \text{or} \quad (G^i)_j = N_{e,2}(\{s(r_{ij}), A^j\}),
\]

\[
(G^i)_{jk} = N_{e,3}(\{((\theta_i)_{jk}, A^j, A^k\}).
\]

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

\[
E_i = F_0(\{D^i, A^i\}).
\]

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 Tao, Jiabing Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jianpeng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Pengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia Xian Zhu, Chenzhe Luo, Yuzhi Zhang, Rixys 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.
4.12.2 Instructions

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

```json
"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:

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

- **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.
4.13 Descriptor "se_a_mask"

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 residues coordinates. A sample input system that contains 2 frames is included in the directory.

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

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

```bash
$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

```json
"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 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.
- 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.

```json
"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 dimension 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.

```json
"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"
}
```

### 4.14 Deep potential long-range (DPLR)

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.
4.14.1 Theory

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

\[ E = E_{\text{DP}} + E_{\text{ele}}, \]

where \( E_{\text{DP}} \) is the short-range contribution constructed as the standard energy model that is fitted against \((E^* - E_{\text{ele}})\). \( E_{\text{ele}} \) 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

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

where \( \beta \) 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

\[
S(m) = \sum_i q_i e^{-2\pi i m r_i} + \sum_n q_n e^{-2\pi i m W_n},
\]

where \( i = \sqrt{-1} \) denotes the imaginary unit, \( r_i \) 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.\(^1\)

4.14.2 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 \texttt{wc.json} and a small dataset \texttt{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

\[ \frac{1}{2} \sum_i \| \mathbf{d}_i - \mathbf{WC}_i \|^2 \]

\(^1\) 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 Tao, Jiabin Yang, Ye Ding, Yifan Li, Davide Tsi, Qiyu Zeng, Han Bao, Yu Xia, Jianeng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Loland Bore, Chun Cai, Yinhian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhyes 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.
DeePMD-kit

"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
```

4.14.3 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 WC... 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
```

4.14.4 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 WC... are explicitly written in the configuration file. Moreover, a virtual bond is established between the oxygens and the WC... 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 1 2 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 WC s 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

```plaintext
# 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_style "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 virtualAtoms. The model file ener.pb stores both the DW and DPLR models, so the position of WC s 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.

```plaintext
# 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**

```plaintext
# "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:

```plaintext
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
```
DeePMD-kit

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.

```bash
# 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.

```bash
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])" u 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

```bash
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.

4.15 Deep Potential - Range Correction (DPRc)

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:

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

4.15.1 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:

$$ s_{DPRc}(r_{ij}) = \begin{cases} 
0, & \text{if } i \in \text{MM} \land j \in \text{MM}, \\
 s(r_{ij}), & \text{otherwise}, 
\end{cases} $$
where \( s_{\text{DPRc}}(r_{ij}) \) is the new switch function and \( s(r_{ij}) \) is the old one. This ensures the forces between MM atoms are zero, i.e.

\[
F_{ij} = -\frac{\partial E}{\partial r_{ij}} = 0, \quad i \in \text{MM} \land j \in \text{MM}.
\]

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

\[
E_i = \begin{cases} 
F_0(D^i), & \text{if } i \in \text{QM}, \\
F_0(D^i) - F_0(0), & \text{if } i \in \text{MM}, 
\end{cases}
\]

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.

### 4.15.2 Training data

Instead of 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 \( E_{\text{MM}} \) is eliminated.

### 4.15.3 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 \( E_{\text{QM}} \) and \( E_{\text{QM/MM}} \) within the cutoff, so we use a hybrid descriptor to describe them separately:

```
"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"
    }
  ]
}
```

(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, Jianeng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sibjørn Leland 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.
DeePMD-kit

```json
{
    "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"
}
```

exclude_types can be generated by the following Python script:

```python
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:

```json
"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.
4.15.4 Run MD simulations

The DPRc model has the best practices with the AMBER QM/MM module. An example is given by GitLab RutgersLBSR/AmbertDPRc. 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.

4.15.5 Pairwise DPRc

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 $l$:

$$E = E_{QM} + \sum_l E_{QM/MM,l}$$

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.

```json
{
  "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": 0.50,
        "rcut": 9.00,
        "attn_layer": 0,
        "neuron": [25, 50, 100],
        "resnet_dt": false,
        "axis_neuron": 12,
        "precision": "float32",
        "seed": 1
      }
    }
  }
}
```

(continues on next page)
"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
}
"qmmm_model": {
  "descriptor": {
    "type": "se_atten_v2",
    "sel": 27,
    "rcut_smth": 0.50,
    "rcut": 6.00,
    "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
  ]
}
}
4.16 Linear model

One can linearly combine existing models with arbitrary coefficients:

```json
"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`.

4.17 Interpolation or combination with a pairwise potential

4.17.1 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

\[ E_i = (1 - w_i)E_i^{DP} + w_i(E_i^0 + E_i^{pair}), \]

where \( w_i \) is the interpolation weight and the \( E_i^{pair} \) is the atomic contribution due to the pairwise potential \( u^{pair}(r) \), i.e.

\[ E_i^{pair} = \sum_{j \in n(i)} u^{pair}(r_{ij}). \]
The interpolation weight $w_i$ is defined by

$$w_i = \begin{cases} 
1, & \sigma_i r_a, \\
\sigma_i^3(-6u_i^2 + 15u_i - 10) + 1, & r_a \leq \sigma_i r_b, \\
0, & \sigma_i \geq r_b, 
\end{cases}$$

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 $\sigma_i$ is the softmin of the distance between atom $i$ and its neighbors,

$$\sigma_i = \frac{\sum_{j \in n(i)} r_{ij}e^{-r_{ij}/\alpha_s}}{\sum_{j \in n(i)} e^{-r_{ij}/\alpha_s}},$$

where the scale $\alpha_s$ is a tunable scale of the interatomic distance $r_{ij}$. The pairwise potential $u_{\text{pair}}(r)$ is defined by a user-defined table that provides the value of $u_{\text{pair}}$ on an evenly discretized grid from 0 to the cutoff distance.

DeePMD-kit also supports combination with a pairwise potential:

$$E_i = E_{\text{DP}}^i + E_{\text{pair}}^i,$$

### 4.17.2 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_0, ..., Type_1-Type_N, ..., and Type_N-Type_N.

The interaction should be smooth at the cut-off distance.

### 4.17.3 Interpolation with a short-range pairwise potential

```json
"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.

---

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, Jianeng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigiørn Leland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jia Yin Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenhuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Wei X. Jia, Darrin M. York, Weina A, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.
4.17.4 Combination with a pairwise potential

To combine with a pairwise potential, use the linear model:

```json
"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.
5.1 Train a model

Several examples of training can be found in the examples directory:

```bash
$ cd $deepmd_source_dir/examples/water/se_e2_a/
```

After switching to that directory, the training can be invoked by

```bash
$ 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.

<table>
<thead>
<tr>
<th>DEEPMD INFO</th>
<th>---Summary of DataSystem: training</th>
<th>----------------------------------------</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>found 3 system(s):</td>
<td></td>
</tr>
<tr>
<td>DEEPMD INFO</td>
<td>system</td>
<td>natoms</td>
</tr>
<tr>
<td>DEEPMD INFO</td>
<td>../data_water/data_0/</td>
<td>192</td>
</tr>
<tr>
<td>DEEPMD INFO</td>
<td>../data_water/data_1/</td>
<td>192</td>
</tr>
<tr>
<td>DEEPMD INFO</td>
<td>../data_water/data_2/</td>
<td>192</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEEPMD INFO</th>
<th>---Summary of DataSystem: validation</th>
<th>----------------------------------------</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>found 1 system(s):</td>
<td></td>
</tr>
<tr>
<td>DEEPMD INFO</td>
<td>system</td>
<td>natoms</td>
</tr>
<tr>
<td>DEEPMD INFO</td>
<td>../data_water/data_3</td>
<td>192</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th># step</th>
<th>rmse_val</th>
<th>rmse_trn</th>
<th>rmse_e_val</th>
<th>rmse_e_trn</th>
<th>rmse_f_val</th>
<th>rmse_f_trn</th>
<th>lr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.33e+01</td>
<td>3.41e+01</td>
<td>1.03e+01</td>
<td>1.03e+01</td>
<td>8.39e-01</td>
<td>8.72e-01</td>
<td>1.0e-03</td>
</tr>
<tr>
<td>100</td>
<td>2.57e+01</td>
<td>2.56e+01</td>
<td>1.87e+00</td>
<td>1.88e+00</td>
<td>8.03e-01</td>
<td>8.02e-01</td>
<td>1.0e-03</td>
</tr>
<tr>
<td>200</td>
<td>2.45e+01</td>
<td>2.46e+01</td>
<td>2.26e-01</td>
<td>2.21e-01</td>
<td>7.73e-01</td>
<td>8.10e-01</td>
<td>1.0e-03</td>
</tr>
<tr>
<td>300</td>
<td>1.62e+01</td>
<td>1.66e+01</td>
<td>5.01e-02</td>
<td>4.46e-02</td>
<td>5.11e-01</td>
<td>5.26e-01</td>
<td>1.0e-03</td>
</tr>
<tr>
<td>400</td>
<td>1.36e+01</td>
<td>1.32e+01</td>
<td>1.07e-02</td>
<td>2.07e-03</td>
<td>4.29e-01</td>
<td>4.19e-01</td>
<td>1.0e-03</td>
</tr>
<tr>
<td>500</td>
<td>1.07e+01</td>
<td>1.05e+01</td>
<td>2.45e-03</td>
<td>4.11e-03</td>
<td>3.38e-01</td>
<td>3.31e-01</td>
<td>1.0e-03</td>
</tr>
</tbody>
</table>

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:

```python
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.

### 5.2 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.

#### 5.2.1 Learning rate

**Theory**

The learning rate $\gamma$ 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

$$r = \left( \frac{\gamma_{\text{stop}}}{\gamma^0} \right)^{\frac{1}{s}},$$
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

```json
"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:

\[
lr(t) = \text{start}_\text{lr} \times \text{decay}_\text{rate} ^ \left(\frac{t}{\text{decay}_\text{steps}}\right)
\]

### 5.2.2 Training parameters

Other training parameters are given in the **training** section.

```json
"training": {
  "training_data": {
    "systems": ["../data_water/data_0/", ".../data_water/data_1/", ".../data_water/data_2/"]
  },
  "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,
}
```

---

\(^1\) This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Liang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiamei Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Loland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhyo 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.
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/",  
  "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/",  
  "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.

### 5.2.3 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

```plaintext
positional arguments:
  INPUT the input json database

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

  --init-model INIT_MODEL
  Initializes a model by the provided checkpoint

  --restart RESTART Restart the training from the provided checkpoint

  --init-frz-model INIT_FRZ_MODEL
  Initializes 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.
```

5.2. Advanced options
DeePMD-kit

`--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:

<table>
<thead>
<tr>
<th>Environment variables</th>
<th>Allowed value</th>
<th>Default value</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP_INTERFACE_PREC</td>
<td>high, low</td>
<td>high</td>
<td>Control high (double) or low (float) precision of training.</td>
</tr>
<tr>
<td>DP_AUTO_PARALLELIZATION</td>
<td></td>
<td></td>
<td>Enable auto parallelization for CPU operators.</td>
</tr>
<tr>
<td>DP_JIT</td>
<td>0, 1</td>
<td>0</td>
<td>Enable JIT. Note that this option may either improve or decrease the performance. Requires TensorFlow supports JIT.</td>
</tr>
</tbody>
</table>

5.2.4 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.
5.3 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 further 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_nbach:

    type: int, optional, default: 10
    argument path: model/data_stat_nbach

    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

    The table for the short-range pairwise interaction added on top of DP. The table is a text data file with \((N_t + 1) \times N_t / 2 + 1\) columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

smin_alpha:

    type: float, optional
    argument path: model/smin_alpha

    The short-range tabulated interaction will be switched 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`
- 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`
- 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:**
- type: `bool`, optional, default: `True`
- argument path: `model/srtab_add_bias`
- 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`
- 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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”. 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”, “float16”, “float32”, “float64”, “bfloat16”. 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: NoneType | int, optional, default: None
- argument path: model/type_embedding/seed

Random seed for parameter initialization

**modifier:**
- type: dict, optional
- argument path: model/modifier

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 Å⁻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 Å

**compress:**

- type: dict, optional
- argument path: model/compress

Model compression configurations

**spin:**

- type: dict, optional
- argument path: model/spin

The settings for systems with spin.

**use_spin:**

- type: typing.List[bool]
- argument path: model/spin/use_spin

Whether to use atomic spin model for each atom type

**spin_norm:**

- type: typing.List[float]
- argument path: model/spin/spin_norm

The magnitude of atomic spin for each atom type with spin

**virtual_len:**

- type: typing.List[float]
- argument path: model/spin/virtual_len

The distance between virtual atom representing spin and its corresponding real atom for each atom type with spin

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:

Standard 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.
DeePMD-kit

**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, se_a_ebd_v2, se_a_mask

The type of the descriptor. 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. 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). A parameter is 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**:

**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

---

**5.3. Training Parameters**

93
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 wrapped 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]
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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”. 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”, “float16”, “float32”, “float64”, “bfloat16”. 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: NoneType | int, 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.

**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):

**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 wrapped 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 layer 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 func-
tions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”,
“None”, “none”. 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”, “float16”, “float32”, “float64”, “bfloat16”. 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: NoneType | int, 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):

**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. \( \text{sel}[i] \) gives the selected number of type-\( i \) neighbors. \( \text{sel}[i] \) 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 \( \text{sel} \) 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 \( \text{sel} \). 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 wrapped 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 \( \text{rcut} \) to \( \text{rcut_smth} \)

**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: \( n\_\text{axis}\_\text{neuron} \)
- 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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”. 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”, “float16”, “float32”, “float64”, “bfloat16”. 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:** NoneType | int, 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.

**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 wrapped 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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”. 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”, “float16”, “float32”, “float64”, “bfloat16”. 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: NoneType | int, 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

5.3. Training Parameters
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`:

**sel:**

```
type: str | typing.List[int] | 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 within 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
```
DeePMD-kit

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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”. 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`

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`

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_atten]/precision`

The precision of the embedding net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

**trainable:**

- **type:** bool, optional, default: True
- **argument path:** `model[standard]/descriptor[se_atten]/trainable`

If the parameters in the embedding net is trainable

**seed:**

- **type:** NoneType | int, 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`

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.
DeePMD-kit

```
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 stripped_type_embedding is
  True

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, optional, default: False
  argument path:
  model[standard]/descriptor[se_atten]/stripped_type_embedding
  Whether to strip the type embedding into a separated embedding net-
  work. Setting it to False will fall back to the previous version of se_atten
  which is non-compressible.

smooth_type_embedding:
  type: bool, optional, default: False
  argument path:
  model[standard]/descriptor[se_atten]/smooth_type_embedding
  When using stripped type embedding, whether to dot smooth factor on
  the network output of type embedding to keep the network smooth, in-
  stead 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

When type is set to se_atten_v2:

sel:
  type: str | typing.List[int] | int, optional, default: auto
```
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_attn_v2]/rcut

The cut-off radius.

**rcut_smth:**

- type: float, optional, default: 0.5
- argument path: model[standard]/descriptor[se_attn_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_attn_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_attn_v2]/axis_neuron

Size of the submatrix of G (embedding matrix).

**activation_function:**

- type: str, optional, default: tanh
- argument path: model[standard]/descriptor[se_attn_v2]/activation_function

The activation function in the embedding net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”. 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
  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
  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_atten_v2]/precision
  The precision of the embedding net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

trainable:
  type: bool, optional, default: True
  argument path:
  model[standard]/descriptor[se_atten_v2]/trainable
  If the parameters in the embedding net is trainable

seed:
  type: NoneType | int, 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
  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 stripped_type_embedding is True.

**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 se_a_ebd_v2 (or its alias se_a_type_v2):

**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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”. 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”, “float16”, “float32”, “float64”, “bfloat16”. 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:** NoneType | int, 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.

**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**:

**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 func-
tions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”,
“None”, “none”. 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 net-
work 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”, “float16”, “float32”, “float64”, “bfloat16”. 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: `NoneType | int`, 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 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 polarizability 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 either 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 apams.

**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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”. 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”, “float16”, “float32”, “float64”, “bfloat16”. 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: 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 initial energy shift for each type of atoms. See rcond in `numpy.linalg.lstsq()` for more details.

**seed:**

- **type:** `NoneType | int`, 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:

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 fparsms.

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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”. 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”, “float16”, “float32”, “float64”, “bfloat16”. 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 an 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[dos]/rcond

The condition number used to determine the initial energy shift for each type of atoms. See rcond in numpy.linalg.lstsq() for more details.

**seed:**
- type: NoneType | int, 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
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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”. 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”, “float16”, “float32”, “float64”, “bfloat16”. 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 | NoneType | typing.List[int], 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.
**seed:**

type: `NoneType | int`, 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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”. 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”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

**sel_type:**

type: `int | NoneType | typing.List[int]`, 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.

**seed:**

type: `NoneType | int`, optional  
argument path: `model[standard]/fitting_net[dipole]/seed`

Random seed for parameter initialization of the fitting net

When `type` is set to `multi`:

Multiple-task model.
DeePMD-kit

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:

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 | typing.List[int] | 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 wrapped up to 4 divisible. The option “auto” is equivalent to “auto:1.1”.

5.3. Training Parameters
When `type` is set to `pairwise_dprc`:

```python
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`:

```python
models:
  type: list | dict
  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 definition 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.

```python
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`:

```python
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.
```
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.

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: float | int, 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: float | int, 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: float | int, 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.
\texttt{limit\_pref\_f}:

- type: \texttt{float | int}, optional, default: 1.0
- argument path: \texttt{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.

\texttt{start\_pref\_v}:

- type: \texttt{float | int}, optional, default: 0.0
- argument path: \texttt{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 \texttt{virial.npy} in each data system. If both \texttt{start\_pref\_v} and \texttt{limit\_pref\_v} are set to 0, then the virial will be ignored.

\texttt{limit\_pref\_v}:

- type: \texttt{float | int}, optional, default: 0.0
- argument path: \texttt{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.

\texttt{start\_pref\_ae}:

- type: \texttt{float | int}, optional, default: 0.0
- argument path: \texttt{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 \texttt{atom\_ener.npy} in each data system. If both \texttt{start\_pref\_ae} and \texttt{limit\_pref\_ae} are set to 0, then the atomic energy will be ignored.

\texttt{limit\_pref\_ae}:

- type: \texttt{float | int}, optional, default: 0.0
- argument path: \texttt{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.

\texttt{start\_pref\_pf}:

- type: \texttt{float | int}, optional, default: 0.0
- argument path: \texttt{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 \texttt{atom\_pref.npy} in each data system. If both \texttt{start\_pref\_pf} and \texttt{limit\_pref\_pf} are set to 0, then the atomic prefactor force will be ignored.

\texttt{limit\_pref\_pf}:

- type: \texttt{float | int}, optional, default: 0.0
- argument path: \texttt{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.

\texttt{relative\_f}:

- type: \texttt{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. \(DF_i/\left(\|F\| + \text{relative}_f\right)\) 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: float | int, 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: float | int, 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: float | int, 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: float | int, 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: float | int, 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: float | int, 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: float | int, 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: float | int, 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: float | int, 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:** float | int, 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:** float | int, 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:** float | int, 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. \( DF_i / ( \| F \| + \text{relative}_f ) \) 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 \( \sum_j c_j E_i.c_i \) should be provided by file atom_ener_coeff.npy in each data system, otherwise it's 1.

When **type** is set to **dos:**

**start_pref_dos:**

- **type:** float | int, 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:** float | int, 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:* float | int, optional, default: 0.0

*argument path:* loss[dos]/start_pref_cdf

The prefactor of Cumulative Distribution Function (cumulative integral 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 integral of DOS) label should be provided by file Cumulative Distribution Function (cumulative integral of DOS).npy in each data system. If both start_pref_Cumulative Distribution Function (cumulative integral of DOS) and limit_pref_Cumulative Distribution Function (cumulative integral of DOS) are set to 0, then the Cumulative Distribution Function (cumulative integral of DOS) will be ignored.

**limit**

**pref**

**cdf:**

*type:* float | int, optional, default: 0.0

*argument path:* loss[dos]/limit_pref_cdf

The prefactor of Cumulative Distribution Function (cumulative integral 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:* float | int, 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:* float | int, 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:* float | int, 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:* float | int, 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**:
pref:
  type: float | int
  argument path: loss[tensor]/pref
  The prefactor of the weight of global loss. It should be larger than or equal to 0. If it 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: float | int
  argument path: loss[tensor]/pref_atomic
  The prefactor of the weight of atomic loss. It should be larger than or equal to 0. If it 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: str | typing.List[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.

set_prefix:
  type: str, optional, default: set
  argument path: training/training_data/set_prefix
  The prefix of the sets in the systems.

batch_size:
  type: int | str | 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 divided 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**: `str` | `typing.List[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.

**set_prefix**:
- **type**: `str`, optional, default: `set`
- **argument path**: `training/validation_data/set_prefix`
The prefix of the sets in the systems.

**batch_size:**

- **type:** `int | str | 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 \( 1.0 / \text{get nsystems()} \)
- “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 divided 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: `NoneType | int`, 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.

**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 during training.

profiling:
  type: bool, optional, default: False
  argument path: training/profiling
Profiling during training.

profiling_file:
  type: str, optional, default: timeline.json
  argument path: training/profiling_file
Output file for profiling.

enable_profiler:
  type: bool, optional, default: False
  argument path: training(enable_profiler
Enable TensorFlow Profiler (available in TensorFlow 2.3) 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.
DeePMD-kit

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
5.4 Parallel training

Currently, parallel training 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.

5.4.1 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"
}
```

5.4.2 Scaling test

Testing examples/water/se_e2_a on an 8-GPU host, linear acceleration can be observed with the increasing number of cards.

<table>
<thead>
<tr>
<th>Num of GPU cards</th>
<th>Seconds every 100 samples</th>
<th>Samples per second</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.4515</td>
<td>68.89</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>1.5962</td>
<td>62.65*2</td>
<td>1.82</td>
</tr>
<tr>
<td>4</td>
<td>1.7635</td>
<td>56.71*4</td>
<td>3.29</td>
</tr>
<tr>
<td>8</td>
<td>1.7267</td>
<td>57.91*8</td>
<td>6.72</td>
</tr>
</tbody>
</table>
5.4.3 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 ----------------------------------------
```

5.4.4 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)
```
5.5 Multi-task training

5.5.1 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 $x^{(1)}, \ldots, x^{(n_t)}$. For each dataset, a training task is defined as

$$
\min_{\theta} L^{(t)}(x^{(t)}; \theta^{(t)}), \quad t = 1, \ldots, n_t.
$$

During the multi-task training process, all tasks share one descriptor with trainable parameters $\theta_d$, while each of them has its own fitting network with trainable parameters $\theta^{(t)}_f$, thus $\theta^{(t)} = \{\theta_d, \theta^{(t)}_f\}$. 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 $\theta^{(t)}$. If different fitting networks have the same architecture, they can share the parameters of some layers to improve training efficiency.\(^1\)

5.5.2 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` $\rightarrow$ `fitting_net_dict`, each key of which can be one individual fitting net.
- `training_data`, `validation_data` $\rightarrow$ `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` $\rightarrow$ `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:

- `se_a` (se_e2_a)
- `se_r` (se_e2_r)
- `se_at` (se_e3)
- `se_atten`

\(^1\) 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, Qiuyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yimian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenzxing 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.
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.

### 5.5.3 Initialization from pretrained 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:

```json
"model": {
  "type_map": ["O", "H"],
  "descriptor": {
    "type": "se_e2_a",
    "sel": [46, 92],
    "rcut_smth": 0.5,
    "rcut": 6.0,
    "neuron": [25, 50, 100],
  },
  "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:

```json
"model": {
  "type_map": ["O", "H"],
  "descriptor": {},
  "fitting_net_dict": {
    "water_ener": {},
    "water_dipole": {
      "type": "dipole",
      "neuron": [100, 100, 100],
    },
    "water_ener": {
      "neuron": [240, 240, 240],
      "resnet_dt": true,
    }
  }
}
```

(continues on next page)
"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
```

### 5.5.4 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": ["10", null, "12", null]
    },
    "wb97m": {
        "neuron": [240, 240, 240],
        "layer_name": ["10", null, "12", null]
    }
}
```
5.6 TensorBoard Usage

TensorBoard provides the visualization and tooling needed for machine learning experimentation. Full instructions for TensorBoard can be found here.

5.6.1 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

5.6.2 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.

```json
"training" : {
  "systems": ["../data/*"],
  "set_prefix": "set",
  "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”.

136 Chapter 5. Training
5.6.3 Examples

Tracking and visualizing loss metrics (red: train, blue: test)

tensorboard --logdir path/to/logs
Visualizing DeePMD-kit model graph
Viewing histograms of weights, biases, or other tensors as they change over time

5.6. TensorBoard Usage
DeePMD-kit

Chapter 5. Training

filter_type_0/bias_1_0_1/histogram

filter_type_0/bias_1_1_1/histogram

filter_type_0/bias_2_0_1/histogram

filter_type_0/bias_2_1_1/histogram

filter_type_0/bias_3_0_1/histogram

filter_type_0/bias_3_1_1/histogram

filter_type_0/matrix_1_0_1/histogram

filter_type_0/matrix_1_1_1/histogram

filter_type_0/matrix_2_0_1/histogram
Viewing summaries of trainable variables

5.6.4 Attention

Allowing the tensorboard analysis will take extra execution time. (e.g., 15% increasing at 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.

5.7 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.

### 5.8 Finetune the pretrained model

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 pretrained 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. DPA-1 can hopefully learn the common knowledge in the pretrained dataset (especially the force information) and thus reduce the computational cost in downstream training tasks. If you have a pretrained 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:

```bash
$ 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 pretrained 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:

```json
"model": {
   "type_map": ["O", "H"],
   "type_embedding": {"trainable": true},
   "descriptor": {},
   "fitting_net": {}
}
```
6.1 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

```
$ dp freeze -o graph.pb
```

in the folder where the model is trained. The output model is called graph.pb.

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.

6.2 Compress a model

6.2.1 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:

\[
a_m^l = \frac{1}{2\Delta x_l^2} \left[ 12h_{m,l} - 6(y_{m,l+1} + y_{m,l})\Delta x_l + (y''_{m,l+1} - y''_{m,l})\Delta x_l^2 \right].
\]
\[ b'_m = \frac{1}{2\Delta x_l} \left[ -30h_{m,l} + (14y'_{m,l+1} + 16y''_{m,l}) \Delta x_l + (-2y'_{m,l+1} + 3y''_{m,l}) \Delta x_l^2 \right], \]
\[ c'_m = \frac{1}{2\Delta x_l} \left[ 20h_{m,l} - (8y'_{m,l+1} + 12y''_{m,l}) \Delta x_l + (y''_{m,l+1} - 3y''_{m,l}) \Delta x_l^2 \right], \]
\[ d^l_m = \frac{1}{2} y''_{m,l}, \]
\[ e^l_m = y'_{m,l}, \]
\[ f^l_m = y_{m,l}, \]

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 \( (G^i)^T R \) requires the transfer of the tensor \( G^i \) 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 \( (G^i)^T R \) with the tabulated inference step. More specifically, once one column of the \( (G^i)^T \) 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 \( (G^i)^T R \). By the operator merging technique, the allocation of \( G^i \) 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 \( (R^i) \) and embedding \( (G^i) \) 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.\(^1\)

### 6.2.2 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]
```

\(^1\) 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, Xu Xia, Jianheng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Leland Bore, Chuan 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, Ji 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.
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
  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 its uniform stride, while the second sub-table takes 100 * stride as its uniform stride. The range of the first table is automati-
cally 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.

Available activation functions for descriptor:

- tanh
- gelu
- relu
- relu6
- softplus
- sigmoid
7.1 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)
```
7.2 Calculate Model Deviation

7.2.1 Theory

Model deviation $\epsilon_y$ is the standard deviation of properties $y$ inferred by an ensemble of models $M_1, \ldots, M_n$ that are trained by the same dataset(s) with the model parameters initialized independently. The DeePMD-kit supports $y$ to be the atomic force $F_i$ and the virial tensor $\Xi$. The model deviation is used to estimate the error of a model at a certain data frame, denoted by $x$, containing the coordinates and chemical species of all atoms. We present the model deviation of the atomic force and the virial tensor

$$
\epsilon_{F,i}(x) = \sqrt{\langle \|F_i(x; \theta_k) - \langle F_i(x; \theta_k) \rangle \|^2 \rangle},
$$

$$
\epsilon_{\Xi,\alpha\beta}(x) = \frac{1}{N} \sqrt{\langle \|\Xi_{\alpha\beta}(x; \theta_k) - \langle \Xi_{\alpha\beta}(x; \theta_k) \rangle \|^2 \rangle},
$$

where $\theta_k$ is the parameters of the model $M_k$, and the ensemble average $\langle \cdot \rangle$ is estimated by

$$
\langle y(x; \theta_k) \rangle = \frac{1}{n_m} \sum_{k=1}^{n_m} y(x; \theta_k).
$$

Small $\epsilon_{F,i}$ 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 $F_i$ or $\Xi$ is quite large, a relative model deviation $\epsilon_{F,i,rel}$ or $\epsilon_{\Xi,\alpha\beta,rel}$ can be used instead of the absolute model deviation:

$$
\epsilon_{F,i,rel}(x) = \frac{|\epsilon_{F,i}(x)|}{|\langle F_i(x; \theta_k) \rangle| + \nu},
$$

$$
\epsilon_{\Xi,\alpha\beta,rel}(x) = \frac{\epsilon_{\Xi,\alpha\beta}(x)}{|\langle \Xi(x; \theta_k) \rangle| + \nu},
$$

where $\nu$ is a small constant used to protect an atom where the magnitude of $F_i$ or $\Xi$ is small from having a large model deviation.

Statistics of $\epsilon_{F,i}$ and $\epsilon_{\Xi,\alpha\beta}$ can be provided, including the maximum, average, and minimal model deviation over the atom index $i$ and over the component index $\alpha, \beta$, respectively. The maximum model deviation of forces $\epsilon_{F,\text{max}}$ in a frame was found to be the best error indicator in a concurrent or active learning algorithm.\(^1\)

7.2.2 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 Tsi, Qiyu Zeng, Han Bao, Yu Xia, Jianfeng Huang, Kosuke Murakami, Yibo Wang, Junhan Chang, Fengbo Yuan, Siqijun Leland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jianyu Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhyse E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jieun 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.
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.

### 7.2.3 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 EIGHT

INFERENCENote that the model for inference is required to be compatible with the DeePMD-kit package. See Model compatibility for details.

8.1 Python interface

One may use the python interface of DeePMD-kit for model inference, an example is given as follows

```python
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.

```python
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 will never release the memory and this may lead to an out-of-memory (OOM) error.
8.1.1 External neighbor list algorithm

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.

```python
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.

8.2 C/C++ interface

8.2.1 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:

```cpp
#include "deepmd/DeepPot.h"

int main()
{
    deepmd::DeepPot dp ("graph.pb");
    std::vector<double> coord = {1., 0., 0., 0., 0., 1.0., 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:

```bash
```

and then run the program:

```bash
./infer_water
```
8.2.2 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, 0.0, 0.0, 1.5, 0.0, 0.3};
    double cell[] = {10.0, 0.0, 0.0, 10.0, 0.0, 0.0, 10.0};
    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);
    free(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:

```
```

and then run the program:

```
./infer_water
```
8.2.3 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`.

```cpp
#include "deepmd/deepmd.hpp"

int main()
{
    deepmd::hpp::DeepPot dp ("graph.pb");
    std::vector<double> coord = {1., 0., 0., 0., 1.5, 1.0, 3.0};
    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:

```bash
```

and then run the program:

```bash
./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:

```cpp
// 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.
8.3 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.

```javascript
const deepmd = require("deepmd-kit");
const dp = new deepmd.DeepPot("graph.pb");
const coord = [1., 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.
DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics

```
usage: dp [-h] [--version]
       {transfer, train, freeze, test, compress, doc-train-input, model-devi, convert-from, neighbor-stat, train-nvmd, gui}
       ...
```

### 9.1 Named Arguments

--version show program’s version number and exit

### 9.2 Valid subcommands

command Possible choices: transfer, train, freeze, test, compress, doc-train-input, model-devi, convert-from, neighbor-stat, train-nvmd, gui

### 9.3 Sub-commands

#### 9.3.1 transfer

pass parameters to another model

```
dp transfer [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-1 LOG_PATH]
            [-r RAW_MODEL] [-o OLD_MODEL] [-o OUTPUT]
```
DeePMD-kit

Named Arguments

- \texttt{-v, --log-level} \hspace{1cm} \text{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”

- \texttt{-l, --log-path} \hspace{1cm} \text{set log file to log messages to disk, if not specified, the logs will only be output to console}

- \texttt{-r, --raw-model} \hspace{1cm} \text{the model receiving parameters}
Default: “raw_frozen_model.pb”

- \texttt{-O, --old-model} \hspace{1cm} \text{the model providing parameters}
Default: “old_frozen_model.pb”

- \texttt{-o, --output} \hspace{1cm} \text{the model after passing parameters}
Default: “frozen_model.pb”

9.3.2 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]
INPUT

Positional Arguments

INPUT \hspace{1cm} \text{the input parameter file in json or yaml format}

Named Arguments

- \texttt{-v, --log-level} \hspace{1cm} \text{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”

- \texttt{-l, --log-path} \hspace{1cm} \text{set log file to log messages to disk, if not specified, the logs will only be output to console}

- \texttt{-m, --mpi-log} \hspace{1cm} \text{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”

- \texttt{-i, --init-model} \hspace{1cm} 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

examples:
  dp train input.json dp train input.json --restart model.ckpt dp train input.json --init-model model.ckpt

9.3.3 freeze

freeze the model

dp freeze [-n] [-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]

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  path to checkpoint folder
  Default: “.”
-o, --output  name of graph, will output to the checkpoint folder
  Default: “frozen_model.pb”
-n, --node-names  the frozen nodes, if not set, determined from the model type
-w, --nvnmd-weight  the name of weight file (.npy), if set, save the model’s weight into the file
--united-model  When in multi-task mode, freeze all nodes into one united model
  Default: False

examples:
  dp freeze dp freeze -o graph.pb
9.3.4 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]
```

**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 to import
  
  Default: “frozen_model.pb”

- `s, --system` The system dir. Recursively detect systems in this directory
  
  Default: “.”

- `f, --datafile` The path to file of test list.

- `S, --set-prefix` The set prefix
  
  Default: “set”

- `n, --numb-test` The number of data for test. 0 means all data.
  
  Default: 0

- `r, --rand-seed` 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` Test the accuracy of atomic label, i.e. energy/tensor (dipole, polar)
  
  Default: False

examples:
```
dp test -m graph.pb -s /path/to/system -n 30
```
9.3.5 compress

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”
-1, --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 ‘work-
     ers’ 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 sec-
     ond table will use 10*step as its 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 environ-
     ment matrix overflow the first or second table range). By default do not
     check the overflow
     Default: -1
DeePMD-kit

-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

9.3.6 doc-train-input

print the documentation (in rst format) of input training parameters.

\[
dp \text{ doc-train-input} \ [-h] \ [-v \{\text{DEBUG,3,INFO,2,WARNING,1,ERROR,0}\}] \ [-l \text{LOG\_PATH}] \n\quad \text{[--out-type \{rst,json\}]} \n\]

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”

9.3.7 model-devi

calculate model deviation

\[
dp \text{ model-devi} \ [-h] \ [-v \{\text{DEBUG,3,INFO,2,WARNING,1,ERROR,0}\}] \ [-l \text{LOG\_PATH}] \n\quad \text{[--models MODELS [MODELS ...]]} \ [-s \text{SYSTEM}] \ [-s \text{SET\_PREFIX}] \ [-o \text{OUTPUT}] \n\quad \text{[--f FREQUENCY]} \ [-v \text{--real\_error}] \ [-v \text{--atomic}] \ [-v \text{--relative RELATIVE}] \n\quad \text{[--relative\_v RELATIVE\_V]} \n\]

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 to import
   Default: ['graph.000.pb', 'graph.001.pb', 'graph.002.pb', 'graph.003.pb']
-s, --system
The system directory. Recursively detect systems in this directory.
Default: "."
-S, --set-prefix
The set prefix
Default: "set"
-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
```

### 9.3.8 convert-from

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

9.3.9 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 ...] [-one-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

--one-type  treat all types as a single type. Used with se_atten descriptor.
Default: False

tables:
dp neighbor-stat -s data -r 6.0 -t O H
9.3.10 train-nvnmd

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]
```

**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
```

9.3.11 gui

Serve DP-GUI.

```
dp gui [-h] [-v \{DEBUG,3,INFO,2,WARNING,1,ERROR,0\}] [-l LOG_PATH] [-p PORT]
[--bind_all]
```

9.3. Sub-commands
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
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.

10.1 Use deep potential with ASE

Deep potential can be set up as a calculator with ASE to obtain potential energies and forces.

```python
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:

```python
from ase.optimize import BFGS

dyn = BFGS(water)
dyn.run(fmax=1e-6)
print(water.get_positions())
```

10.2 Run MD with LAMMPS

10.2.1 units

All units in LAMMPS except 1j are supported. 1j 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 manually. The unit conversion is done automatically by LAMMPS.
DeePMD-kit

The only thing that one needs to take care is the unit of the output of `compute deeftensor/atom`. Working with metal units for `compute deeftensor/atom` is totally fine, since there is no unit conversion. For other unit styles, we currently assume that the output of the `compute deeftensor/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.

### 10.2.2 Enable DeePMD-kit plugin (plugin mode)

If you are using the plugin mode, enable DeePMD-kit package in LAMMPS with `plugin` command:

```bash
plugin load libdeepmd_lmp.so
```

After LAMMPS version patch_24Mar2022, another way to load plugins is to set the environmental variable LAMMPS_PLUGIN_PATH:

```bash
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.

### 10.2.3 pair_style deepmd

The DeePMD-kit package provides the `pair_style deepmd`

```bash
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**

```bash
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 ** 0 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 evaluates 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

$$E_{fi} = \frac{|D_{fi}|}{|f_i| + l}$$

where $D_{fi}$ 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:

$$E_{vi} = \frac{|D_{vi}|}{|v_i| + l}$$

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.
DeePMD-kit

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.

10.2.4 Compute tensorial properties

The DeePMD-kit package provides the compute deeptensor/atom for computing atomic tensorial properties.

\[\text{compute ID group-ID deeptensor/atom model_file}\]

• 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

\[\text{compute dipole all deeptensor/atom dipole.pb}\]

The result of the compute can be dumped to trajectory file by

\[\text{dump 1 all custom 100 water.dump id type c_dipole[1] c_dipole[2] c_dipole[3]}\]

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.

10.2.5 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

\[\text{pair_style deepmd graph.pb}\]
\[\text{pair_coeff *} \]
\[\text{kspace_style pppm 1.0e-5}\]
\[\text{kspace_modify gewald 0.45}\]

Please notice that the DeePM 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.
10.2.6 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:

\[ dv_{atom} = - \sum_m (r_n - r_m) \frac{de_m}{dr_n} \]

Where \( r_n \) is the atomic position of \( n\)th atom, \( v_n \) velocity of the atom and \( \frac{de_m}{dr_n} \) 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 (\( cv_{atom} \)) 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 \( xx, yy, zz, xy, xz, yz, yx, zx, zy \).


10.2.7 Computation of heat flux

Using a per-atom stress tensor one can, for example, compute the heat flux defined as:

\[ J = \sum_n e_n v_n + \sum_{n,m} (r_m - r_n) \frac{de_m}{dr_n} v_n \]

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 details.

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

### 10.3 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

```bash
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:

```json
{
    "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:

```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`. 
10.4 Running MD with GROMACS

10.4.1 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):

```plaintext
[ 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:

```plaintext
; 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

(continues on next page)
DeepMD-kit

(continued from previous page)

<table>
<thead>
<tr>
<th>atom</th>
<th>type</th>
<th>resi</th>
<th>atom</th>
<th>cgnr</th>
<th>charge</th>
<th>mass</th>
<th>qtot</th>
<th>bond_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>c3</td>
<td>c3</td>
<td>0.00000</td>
<td>0.00000</td>
<td>A</td>
<td>3.39771e-01</td>
<td>4.51035e-01</td>
<td>1.91</td>
<td>0.1078</td>
</tr>
<tr>
<td>hc</td>
<td>hc</td>
<td>0.00000</td>
<td>0.00000</td>
<td>A</td>
<td>2.60018e-01</td>
<td>8.70272e-02</td>
<td>1.46</td>
<td>0.0208</td>
</tr>
</tbody>
</table>

[moleculetype]

; name nrexcl
methane 3

[atoms]

<table>
<thead>
<tr>
<th>nr</th>
<th>type</th>
<th>resi</th>
<th>atom</th>
<th>cgnr</th>
<th>charge</th>
<th>mass</th>
<th>qtot</th>
<th>bond_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c3</td>
<td>1</td>
<td>MOL</td>
<td>C1</td>
<td>1</td>
<td>-0.106800</td>
<td>12.01000</td>
<td>qtot -0.107</td>
</tr>
<tr>
<td>2</td>
<td>hc</td>
<td>1</td>
<td>MOL</td>
<td>H1</td>
<td>2</td>
<td>0.026700</td>
<td>1.00800</td>
<td>qtot -0.080</td>
</tr>
<tr>
<td>3</td>
<td>hc</td>
<td>1</td>
<td>MOL</td>
<td>H2</td>
<td>3</td>
<td>0.026700</td>
<td>1.00800</td>
<td>qtot -0.053</td>
</tr>
<tr>
<td>4</td>
<td>hc</td>
<td>1</td>
<td>MOL</td>
<td>H3</td>
<td>4</td>
<td>0.026700</td>
<td>1.00800</td>
<td>qtot -0.027</td>
</tr>
<tr>
<td>5</td>
<td>hc</td>
<td>1</td>
<td>MOL</td>
<td>H4</td>
<td>5</td>
<td>0.026700</td>
<td>1.00800</td>
<td>qtot 0.000</td>
</tr>
</tbody>
</table>

[bonds]

<table>
<thead>
<tr>
<th>ai</th>
<th>aj</th>
<th>funct</th>
<th>r</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1.0970e-01</td>
<td>3.1455e+05</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1.0970e-01</td>
<td>3.1455e+05</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1.0970e-01</td>
<td>3.1455e+05</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1.0970e-01</td>
<td>3.1455e+05</td>
</tr>
</tbody>
</table>

[angles]

<table>
<thead>
<tr>
<th>ai</th>
<th>aj</th>
<th>ak</th>
<th>funct</th>
<th>theta</th>
<th>cth</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1.0758e+02</td>
<td>3.2635e+02</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1.0758e+02</td>
<td>3.2635e+02</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1.0758e+02</td>
<td>3.2635e+02</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1.0758e+02</td>
<td>3.2635e+02</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1.0758e+02</td>
<td>3.2635e+02</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1.0758e+02</td>
<td>3.2635e+02</td>
</tr>
</tbody>
</table>

DeepMD Settings

Before running simulations, we need to tell GROMACS to use DeepPotential by setting the environment variable GMX_DEEPMD_INPUT_JSON:

```bash
export GMX_DEEPMD_INPUT_JSON=input.json
```

Then, in your working directories, we have to write input.json file:

```json
{
    "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

Chapter 10. Integrate with third-party packages
• **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.

#### 10.4.2 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.

![NVT_500ps](image)

However, we still recommend you run an all-atom DP simulation using LAMMPS since it is more stable and efficient.
10.5 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.

10.5.1 dpdata

dpdata provides the predict method for System class:

```python
import dpdata
dsys = dpdata.LabeledSystem("OUTCAR")
dp_sys = dsys.predict("frozen_model_compressed.pb")
```

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.

10.5.2 OpenMM plugin for DeePMD-kit

An OpenMM plugin is provided from JingHuangLab/openmm_deepmd_plugin, written by the Huang Lab at Westlake University.

10.5.3 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.

10.5.4 CP2K interface to DeePMD-kit

CP2K v2024.2 adds an interface to the DeePMD-kit for molecular dynamics. Read the CP2K manual for details.
10.5.5 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.

10.5.6 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.

10.5.7 ABACUS

ABACUS can run molecular dynamics with a DP model. User is required to build ABACUS with DeePMD-kit.
11.1 Introduction

NVNMD stands for non-von Neumann molecular dynamics.

This is the training code we used to generate the results in our paper entitled “Accurate and Efficient Molecular Dynamics based on Machine Learning and non von Neumann Architecture”, which has been accepted by npj Computational Materials (DOI: 10.1038/s41524-022-00773-z).

Any user can follow two consecutive steps to run molecular dynamics (MD) on the proposed NVNMD computer, which has been released online: (i) to train a machine learning (ML) model that can decently reproduce the potential energy surface (PES); and (ii) to deploy the trained ML model on the proposed NVNMD computer, then run MD there to obtain the atomistic trajectories.

11.1.1 Training

Our training procedure consists of not only continuous neural network (CNN) training but also quantized neural network (QNN) training which uses the results of CNN as inputs. It is performed on CPU or GPU by using the training codes we open-sourced online.

To train an ML model that can decently reproduce the PES, a training and testing data set should be prepared first. This can be done by using either the state-of-the-art active learning tools or the outdated (i.e., less efficient) brute-force density functional theory (DFT)-based ab-initio molecular dynamics (AIMD) sampling.

If you just want to simply test the training function, you can use the example in the $deepmd_source_dir/examples/nvnmd directory. If you want to fully experience training and running MD functions, you can download the complete example from the website.

Then, copy the data set to the working directory

```bash
mkdir -p $workspace
cd $workspace
mkdir -p data
cp -r $dataset data
```

where $dataset is the path to the data set and $workspace is the path to the working directory.
11.1.2 Input script

Create and go to the training directory.

```
mkdir train
cd train
```

Then copy the input script `train_cnn.json` and `train_qnn.json` to the directory `train`

```
cp -r $deepmd_source_dir/examples/nvnmd/train/train_cnn.json train_cnn.json
cp -r $deepmd_source_dir/examples/nvnmd/train/train_qnn.json train_qnn.json
```

The structure of the input script is as follows

```
{
   "nvnmd" : {},
   "learning_rate" : {},
   "loss" : {},
   "training" : {}
}
```

**nvnmd**

The “nvnmd” section is defined as

```
{
   "version": 0,
   "max_nnei":128,
   "net_size":128,
   "sel":[60, 60],
   "rcut":6.0,
   "rcut_smth":0.5,
   "type_map": ["Ge", "Te"]
}
```

where items are defined as:

<table>
<thead>
<tr>
<th>Item</th>
<th>Mean</th>
<th>Optional Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>the version of network structure</td>
<td>0 or 1</td>
</tr>
<tr>
<td>max_nnei</td>
<td>the maximum number of neighbors that do not distinguish element types</td>
<td>128 or 256</td>
</tr>
<tr>
<td>net_size</td>
<td>the size of neural network</td>
<td>128</td>
</tr>
<tr>
<td>sel</td>
<td>the number of neighbors</td>
<td>version 0: integer list of lengths 1 to 4 are acceptable; version 1: integer</td>
</tr>
<tr>
<td>rcut</td>
<td>the cutoff radial</td>
<td>(0, 8.0)</td>
</tr>
<tr>
<td>rcut_smth</td>
<td>the smooth cutoff parameter</td>
<td>(0, 8.0)</td>
</tr>
<tr>
<td>type_map</td>
<td>mapping atom type to the name (str) of the type</td>
<td>string list, optional</td>
</tr>
</tbody>
</table>

Multiple versions of the nvnmd model correspond to different network structures. `nvnmd-v0` and `nvnmd-v1` differ in the following ways:

1. `nvnmd-v0` and `nvnmd-v1` use the se_a descriptor and se_atten descriptor, respectively
2. `nvnmd-v0` has 1 set of parameters for each element and supports up to 4 element types. `nvnmd-v1` shares 1 set of parameters for each element and supports up to 31 types.
3. *nvnmd-v0* distinguishes between neighboring atoms, so `sel` is a list of integers. *nvnmd-v1* does not distinguish between neighboring atoms, so `sel` is an integer.

**learning_rate**

The “learning_rate” section is defined as

```json
{
    "type": "exp",
    "start_lr": 1e-3,
    "stop_lr": 3e-8,
    "decay_steps": 5000
}
```

where items are defined as:

<table>
<thead>
<tr>
<th>Item</th>
<th>Mean</th>
<th>Optional Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>learning rate variant type</td>
<td>exp</td>
</tr>
<tr>
<td>start_lr</td>
<td>the learning rate at the beginning of the training</td>
<td>a positive real number</td>
</tr>
<tr>
<td>stop_lr</td>
<td>the desired learning rate at the end of the training</td>
<td>a positive real number</td>
</tr>
<tr>
<td>decay_steps</td>
<td>the learning rate is decaying every {decay_steps} training steps</td>
<td>a positive integer</td>
</tr>
</tbody>
</table>

**loss**

The “loss” section is defined as

```json
{
    "start_pref_e": 0.02,
    "limit_pref_e": 2,
    "start_pref_f": 1000,
    "limit_pref_f": 1,
    "start_pref_v": 0,
    "limit_pref_v": 0
}
```

where items are defined as:

<table>
<thead>
<tr>
<th>Item</th>
<th>Mean</th>
<th>Optional Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_pref_e</td>
<td>the loss factor of energy at the beginning of the training</td>
<td>zero or positive real number</td>
</tr>
<tr>
<td>limit_pref_e</td>
<td>the loss factor of energy at the end of the training</td>
<td>zero or positive real number</td>
</tr>
<tr>
<td>start_pref_f</td>
<td>the loss factor of force at the beginning of the training</td>
<td>zero or positive real number</td>
</tr>
<tr>
<td>limit_pref_f</td>
<td>the loss factor of force at the end of the training</td>
<td>zero or positive real number</td>
</tr>
<tr>
<td>start_pref_v</td>
<td>the loss factor of virial at the beginning of the training</td>
<td>zero or positive real number</td>
</tr>
<tr>
<td>limit_pref_v</td>
<td>the loss factor of virial at the end of the training</td>
<td>zero or positive real number</td>
</tr>
</tbody>
</table>
The “training” section is defined as

```json
{
    "seed": 1,
    "stop_batch": 1000000,
    "numb_test": 1,
    "disp_file": "lcurve.out",
    "disp_freq": 1000,
    "save_ckpt": "model.ckpt",
    "save_freq": 10000,
    "training_data": {
        "systems": ["system1_path", "system2_path", "..."],
        "set_prefix": "set",
        "batch_size": ["batch_size_of_system1", "batch_size_of_system2", "..."]
    }
}
```

where items are defined as:

<table>
<thead>
<tr>
<th>Item</th>
<th>Mean</th>
<th>Optional Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>seed</td>
<td>the random seed</td>
<td>an integer</td>
</tr>
<tr>
<td>stop_batch</td>
<td>the total training steps</td>
<td>a positive integer</td>
</tr>
<tr>
<td>numb_test</td>
<td>the accuracy is test by using (numb_test) sample</td>
<td>a positive integer</td>
</tr>
<tr>
<td>disp_file</td>
<td>the log file where the training message display</td>
<td>a string</td>
</tr>
<tr>
<td>disp_freq</td>
<td>display frequency</td>
<td>a positive integer</td>
</tr>
<tr>
<td>save_ckpt</td>
<td>path prefix of check point files</td>
<td>a string</td>
</tr>
<tr>
<td>save_freq</td>
<td>save frequency</td>
<td>a positive integer</td>
</tr>
<tr>
<td>systems</td>
<td>a list of data directory which contains the dataset</td>
<td>string list</td>
</tr>
<tr>
<td>set_prefix</td>
<td>the prefix of dataset</td>
<td>a string</td>
</tr>
<tr>
<td>batch_size</td>
<td>a list of batch size of corresponding dataset</td>
<td>a integer list</td>
</tr>
</tbody>
</table>

## 11.1.3 Training

Training can be invoked by

```
# step1: train CNN
dp train-nvnmd train_cnn.json -s s1
# step2: train QNN
dp train-nvnmd train_qnn.json -s s2
```

After the training process, you will get two folders: `nvnmdd_cnn` and `nvnmdd_qnn`. The `nvnmdd_cnn` contains the model after continuous neural network (CNN) training. The `nvnmdd_qnn` contains the model after quantized neural network (QNN) training. The binary file `nvnmdd_qnn/model.pb` is the model file that is used to perform NVNMD in the server [http://nvnmdd.picp.vip](http://nvnmdd.picp.vip).

You can also restart the CNN training from the path prefix of checkpoint files (`nvnmdd_cnn/model.ckpt`) by

```
dp train-nvnmd train_cnn.json -r nvnmdd_cnn/model.ckpt -s s1
```

You can also initialize the CNN model and train it by
11.2 Testing

The frozen model can be used in many ways. The most straightforward testing can be invoked by

```
mkdir test
dp test -m ./nvnmd_qnn/frozen_model.pb -s path/to/system -d .test/detail -n 99999 -l test/output. -log
```

where the frozen model file to import is given via the `-m` command line flag, the path to the testing data set is given via the `-s` command line flag, and the file containing details of energy, forces and virials accuracy is given via the `-d` command line flag, the amount of data for testing is given via the `-n` command line flag.

11.3 Running MD in Bohrium

After CNN and QNN training, you can upload the ML model to our online NVNMD system and run MD there through Bohrium (https://bohrium.dp.tech). Bohrium is a research platform designed for AI for Science Era. For more information, please refer to Bohrium Introduction.

11.3.1 Registration

Click here to register a Bohrium account. If you already have an account for other DP products, you can skip this step and log in directly.
### 11.3.2 Top-up and create a project

After entering the homepage, you can click on the User Center in the lower left corner to top-up by yourself.

After completing the top-up, click on the Projects, and then click New Project in the upper right corner of the page. Give the project a name that is easy for you to recognize and click OK. If the project has other collaborators, you can refer to Project Collaboration for more information.
11.3.3 Run job

We will use Utility to submit jobs, you can install it with the following command

```bash
pip install lbg
```

When using the Lebesgue Utility for the first time, you need to configure your account by

```bash
lbg config account
```

Enter your Bohrium account and the corresponding password.

Then you need prepare the configuration file `job.json`, the configuration file is as follows

```json
{
    "job_name": "test",
    "command": "/usr/bin/lmp_mpi < in.lmp;",
    "log_file": "OUTCAR",
    "machine_type": "c4_m16_cpu",
    "job_type": "container",
    "image_name": "lammps_dp:29Sep2021",
    "platform": "hnugba",
    "region": "default",
    "project_id": 0000
}
```

where items are defined as:

<table>
<thead>
<tr>
<th>Item</th>
<th>Mean</th>
<th>Optional Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_name</td>
<td>the name of computing job, which can be named freely</td>
<td>a string</td>
</tr>
<tr>
<td>command</td>
<td>the command to be executed on the computing node</td>
<td>a string</td>
</tr>
<tr>
<td>log_file</td>
<td>the log file that can be viewed at any time during the calculation process, which can be viewed on the Bohrium “Jobs” page</td>
<td>a string</td>
</tr>
<tr>
<td>machine_type</td>
<td>the machine type used for the job</td>
<td>“c1_m4_cpu”, “c4_m16_cpu”, “c8_m32_cpu”</td>
</tr>
<tr>
<td>job_type</td>
<td>the job type</td>
<td>“container”</td>
</tr>
<tr>
<td>image_name</td>
<td>the image name used for the job</td>
<td>“lammps_dp:29Sep2021”</td>
</tr>
<tr>
<td>platform</td>
<td>resource provider</td>
<td>“hnugba”</td>
</tr>
<tr>
<td>project_id</td>
<td>the project ID to which the job belongs, which can be viewed on the “Projects” page</td>
<td>a integer</td>
</tr>
</tbody>
</table>

Notice The task will use 4 CPU cores for computation, so do not repeatedly use the `mpirun` command, otherwise an error will be reported. All 0000 after “project_id” need to be replaced with your own project ID, which can be viewed on the “Projects” page. Also, the JSON file format requires that no commas be added after the last field within the {}, otherwise, there will be a syntax error. Please check the documentation for the latest hardware configuration information.

In addition, it is necessary to prepare input script of the MD simulation, the ML model named `model.pb` obtained by QNN training and data files containing information required for running an MD simulation (e.g., `coord.lmp` containing initial atom coordinates).

In the input script, one needs to specify the pair style as follows
DeePMD-kit

```bash
pair_style nvnmd model.pb
pair_coeff *
```

where `model.pb` is the path to model.

After preparing the configuration file and the required files for calculation, using Lebesgue Utility to submit the job

```bash
lbg job submit -i job.json -p ./
```

where the configuration file for the job is given via the `-i` command line flag, the directory where the input files are located is given via the `-p` command line flag. Bohrium will package and upload the specified directory, and after decompressing it on the computing node, it will switch the working directory to that directory.

After the job is submitted successfully, the JOB ID and JOB GROUP ID will be output.

### 11.3.4 Check job status

After successfully submitting the job, you can view the progress and related logs of the submitted jobs on the `Jobs` page.
11.3.5 Terminate and delete jobs

You can choose between terminate and delete operations.

- Terminate: To end running jobs/job groups in advance, save the generated result files, and the status of the terminated jobs will be changed to “completed”.
- Delete: To end running jobs/job groups, the status of the jobs will be changed to “failed”. Job result files will be deleted, and the jobs/job groups disappear from the list. The delete operation cannot be undone.

The Jobs page provides buttons to end jobs and job groups

You can also use the Lebesgue Utility tool to end jobs

```
lbg jobgroup terminate <JOB GROUP ID>
lbg job terminate <JOB ID>
lbg jobgroup rm <JOB GROUP ID>
lbg job rm <JOB ID>
```

11.3.6 Download Results

After the calculation is completed, you can download the results on the Jobs page, or save them to the data disk.
DeePMD-kit

You can also download it using the commands of Lebesgue Utility

```
lbg job download <JOB ID>
```

or

```
lbg jobgroup download <JOB GROUP ID>
```

11.4 Running MD in Nvnmd website

After CNN and QNN training, you can upload the ML model to our online NVNMD system and run MD there.

11.4.1 Account application

The server website of NVNMD is available at http://nvnmd.picp.vip. You can visit the URL and enter the login interface.

To obtain an account, please send your application to the email (jie_liu@hnu.edu.cn, liujie@uw.edu). The username and password will be sent to you by email.

11.4.2 Adding task

After successfully obtaining the account, enter the username and password in the login interface, and click “Login” to enter the homepage.
The homepage displays the remaining calculation time and all calculation records not deleted. Click **Add a new task** to enter the interface for adding a new task.

- **Task name:** name of the task
- **Upload mode:** two modes of uploading results to online data storage, including **Manual upload** and **Automatic upload**. Results need to be uploaded manually to online data storage with **Manual upload** mode and will be uploaded automatically with **Automatic upload** mode.
- **Input script:** input file of the MD simulation.

In the input script, one needs to specify the pair style as follows:

```plaintext
pair_style nvnmd model.pb
pair_coeff * *
```

- **Model file:** the ML model named **model.pb** obtained by QNN training.
- **Data files:** data files containing the information required for running an MD simulation (e.g., coord.lmp containing initial atom coordinates).

Next, you can click **Submit** to submit the task and then automatically return to the homepage.
Then, click **Refresh** to view the latest status of all calculation tasks.

### 11.4.3 Cancelling calculation

For the task whose calculation status is **Pending** and **Running**, you can click the corresponding **Cancel** on the homepage to stop the calculation.
11.4.4 Downloading results

For the task whose calculation status is Completed, Failed and Cancelled, you can click the corresponding Package or Separate files in the Download results bar on the homepage to download results.

Click Package to download a zipped package of all files including input files and output results.

Click Separate files to download the required separate files.

If Manual upload mode is selected or the file has expired, click Upload on the download interface to upload manually.

11.4.5 Deleting record

For the task no longer needed, you can click the corresponding Delete on the homepage to delete the record.
Records cannot be retrieved after deletion.
11.4.6 Clearing records

Click Clear calculation records on the homepage to clear all records.
Records cannot be retrieved after clearing.
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.

12.1 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.

12.1.1 Al2O3

Fitting net size tuning form on Al2O3: (embedding-net size: [25,50,100])

<table>
<thead>
<tr>
<th>Fitting-net size</th>
<th>Energy L2err(eV)</th>
<th>Energy L2err/Natoms(eV)</th>
<th>Force L2err(eV/Angstrom)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[240,240,240]</td>
<td>1.742252e-02</td>
<td>7.259383e-05</td>
<td>4.014115e-02</td>
</tr>
<tr>
<td>[80,80,80]</td>
<td>1.799349e-02</td>
<td>7.497287e-05</td>
<td>4.042977e-02</td>
</tr>
<tr>
<td>[40,40,40]</td>
<td>1.799036e-02</td>
<td>7.495984e-05</td>
<td>4.068806e-02</td>
</tr>
<tr>
<td>[20,20,20]</td>
<td>1.834032e-02</td>
<td>7.641801e-05</td>
<td>4.094784e-02</td>
</tr>
<tr>
<td>[10,10,10]</td>
<td>1.913058e-02</td>
<td>7.971073e-05</td>
<td>4.154775e-02</td>
</tr>
<tr>
<td>[5,5,5]</td>
<td>1.932914e-02</td>
<td>8.053808e-05</td>
<td>4.188052e-02</td>
</tr>
<tr>
<td>[4,4,4]</td>
<td>1.944832e-02</td>
<td>8.103467e-05</td>
<td>4.217826e-02</td>
</tr>
<tr>
<td>[3,3,3]</td>
<td>2.068631e-02</td>
<td>8.619296e-05</td>
<td>4.300497e-02</td>
</tr>
<tr>
<td>[2,2,2]</td>
<td>2.267962e-02</td>
<td>9.449840e-05</td>
<td>4.413609e-02</td>
</tr>
<tr>
<td>[1,1,1]</td>
<td>2.813596e-02</td>
<td>1.172332e-04</td>
<td>4.781115e-02</td>
</tr>
<tr>
<td>[]</td>
<td>3.135002e-02</td>
<td>1.306251e-04</td>
<td>5.373120e-02</td>
</tr>
</tbody>
</table>

[] 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])

<table>
<thead>
<tr>
<th>Embedding-net size</th>
<th>Energy L2err(eV)</th>
<th>Energy L2err/Natoms(eV)</th>
<th>Force L2err(eV/Angstrom)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[25,50,100]</td>
<td>1.742252e-02</td>
<td>7.259383e-05</td>
<td>4.014115e-02</td>
</tr>
<tr>
<td>[10,20,40]</td>
<td>2.909990e-02</td>
<td>1.212496e-04</td>
<td>4.734667e-02</td>
</tr>
<tr>
<td>[5,10,20]</td>
<td>3.357767e-02</td>
<td>1.399070e-04</td>
<td>5.706385e-02</td>
</tr>
<tr>
<td>[4,8,16]</td>
<td>6.060367e-02</td>
<td>2.525153e-04</td>
<td>7.333304e-02</td>
</tr>
<tr>
<td>[3,6,12]</td>
<td>5.656043e-02</td>
<td>2.356685e-04</td>
<td>7.793393e-02</td>
</tr>
<tr>
<td>[2,4,8]</td>
<td>5.277023e-02</td>
<td>2.198759e-04</td>
<td>7.459995e-02</td>
</tr>
<tr>
<td>[1,2,4]</td>
<td>1.302282e-01</td>
<td>5.426174e-04</td>
<td>9.672238e-02</td>
</tr>
</tbody>
</table>

### 12.1.2 Cu

### Fitting net size tuning form on Cu: (embedding-net size: [25,50,100])

<table>
<thead>
<tr>
<th>Fitting-net size</th>
<th>Energy L2err(eV)</th>
<th>Energy L2err/Natoms(eV)</th>
<th>Force L2err(eV/Angstrom)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10,10,10]</td>
<td>4.399364e-02</td>
<td>1.718502e-04</td>
<td>8.962891e-02</td>
</tr>
<tr>
<td>[5,5,5]</td>
<td>4.468104e-02</td>
<td>1.745470e-04</td>
<td>8.970111e-02</td>
</tr>
<tr>
<td>[4,4,4]</td>
<td>4.463562e-02</td>
<td>1.743586e-04</td>
<td>8.972011e-02</td>
</tr>
<tr>
<td>3,3,3</td>
<td>4.493575e-02</td>
<td>1.755374e-04</td>
<td>8.971303e-02</td>
</tr>
<tr>
<td>2,2,2</td>
<td>4.500736e-02</td>
<td>1.758100e-04</td>
<td>8.973878e-02</td>
</tr>
<tr>
<td>1,1,1</td>
<td>4.542073e-02</td>
<td>1.774247e-04</td>
<td>8.964761e-02</td>
</tr>
<tr>
<td>[]</td>
<td>4.545168e-02</td>
<td>1.775456e-04</td>
<td>8.983201e-02</td>
</tr>
</tbody>
</table>

### Embedding net size tuning form on Cu: (Fitting-net size: [240,240,240])

<table>
<thead>
<tr>
<th>Embedding-net size</th>
<th>Energy L2err(eV)</th>
<th>Energy L2err/Natoms(eV)</th>
<th>Force L2err(eV/Angstrom)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[25,50,100]</td>
<td>4.135548e-02</td>
<td>1.615449e-04</td>
<td>8.940946e-02</td>
</tr>
<tr>
<td>[20,40,50]</td>
<td>4.203562e-02</td>
<td>1.642016e-04</td>
<td>8.925881e-02</td>
</tr>
<tr>
<td>[15,30,60]</td>
<td>4.146672e-02</td>
<td>1.619794e-04</td>
<td>8.936911e-02</td>
</tr>
<tr>
<td>[10,20,40]</td>
<td>4.280060e-02</td>
<td>1.665258e-04</td>
<td>8.955818e-02</td>
</tr>
<tr>
<td>3,6,12</td>
<td>1.362098e-01</td>
<td>5.320695e-04</td>
<td>1.073860e-01</td>
</tr>
<tr>
<td>2,4,8</td>
<td>7.061800e-02</td>
<td>2.758515e-04</td>
<td>9.126418e-02</td>
</tr>
<tr>
<td>[1,2,4] &amp; seed = 1</td>
<td>9.843161e-02</td>
<td>3.844985e-04</td>
<td>9.348505e-02</td>
</tr>
<tr>
<td>[1,2,4] &amp; seed = 2</td>
<td>9.404335e-02</td>
<td>3.673568e-04</td>
<td>9.304089e-02</td>
</tr>
<tr>
<td>[1,2,4] &amp; seed = 3</td>
<td>1.508016e-01</td>
<td>5.890688e-04</td>
<td>1.382356e-01</td>
</tr>
<tr>
<td>[1,2,4] &amp; seed = 4</td>
<td>9.686949e-02</td>
<td>3.783965e-04</td>
<td>9.294820e-02</td>
</tr>
</tbody>
</table>
### 12.1.3 Water

Fitting net size tuning form on water: (embedding-net size: [25,50,100])

<table>
<thead>
<tr>
<th>Fitting-net size</th>
<th>Energy L2err/Natoms(eV)</th>
<th>Force L2err(eV/Angstrom)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[200,200,200]</td>
<td>9.3221E-04</td>
<td>5.2366E-02</td>
</tr>
<tr>
<td>[120,120,120]</td>
<td>9.5407E-04</td>
<td>5.3093E-02</td>
</tr>
<tr>
<td>[80,80,80]</td>
<td>9.4605E-04</td>
<td>5.3402E-02</td>
</tr>
<tr>
<td>[40,40,40]</td>
<td>9.8533E-04</td>
<td>5.5790E-02</td>
</tr>
<tr>
<td>[20,20,20]</td>
<td>1.0057E-03</td>
<td>5.8232E-02</td>
</tr>
<tr>
<td>[10,10,10]</td>
<td>1.0466E-03</td>
<td>6.2279E-02</td>
</tr>
<tr>
<td>[5,5,5]</td>
<td>1.1154E-03</td>
<td>6.7994E-02</td>
</tr>
<tr>
<td>[4,4,4]</td>
<td>1.1289E-03</td>
<td>6.9613E-02</td>
</tr>
<tr>
<td>[3,3,3]</td>
<td>1.2368E-03</td>
<td>7.9786E-02</td>
</tr>
<tr>
<td>[2,2,2]</td>
<td>1.3558E-03</td>
<td>9.7042E-02</td>
</tr>
<tr>
<td>[1,1,1]</td>
<td>1.4633E-03</td>
<td>1.1265E-01</td>
</tr>
<tr>
<td>[]</td>
<td>1.5193E-03</td>
<td>1.2136E-01</td>
</tr>
</tbody>
</table>

Embedding net size tuning form on water: (Fitting-net size: [240,240,240])

<table>
<thead>
<tr>
<th>Embedding-net size</th>
<th>Energy L2err/Natoms(eV)</th>
<th>Force L2err(eV/Angstrom)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[25,50,100]</td>
<td>9.1589E-04</td>
<td>5.1540E-02</td>
</tr>
<tr>
<td>[20,40,80]</td>
<td>9.5080E-04</td>
<td>5.3593E-02</td>
</tr>
<tr>
<td>[15,30,60]</td>
<td>9.7996E-04</td>
<td>5.6338E-02</td>
</tr>
<tr>
<td>[10,20,40]</td>
<td>1.0353E-03</td>
<td>6.2776E-02</td>
</tr>
<tr>
<td>[5,10,20]</td>
<td>1.1254E-03</td>
<td>7.3195E-02</td>
</tr>
<tr>
<td>[4,8,16]</td>
<td>1.2495E-03</td>
<td>8.0371E-02</td>
</tr>
<tr>
<td>[3,6,12]</td>
<td>1.3604E-03</td>
<td>9.9883E-02</td>
</tr>
<tr>
<td>[2,4,8]</td>
<td>1.4358E-03</td>
<td>9.7389E-02</td>
</tr>
<tr>
<td>[1,2,4]</td>
<td>2.1763E-03</td>
<td>1.7276E-01</td>
</tr>
</tbody>
</table>
12.1.4 Mg-Al

Fitting net size tuning form on Mg-Al: (embedding-net size: [25,50,100])

<table>
<thead>
<tr>
<th>Fitting-net size</th>
<th>Energy L2err/Natoms(eV)</th>
<th>Force L2err(eV/Angstrom)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[240,240,240]</td>
<td>3.9606e-03</td>
<td>1.6289e-02</td>
</tr>
<tr>
<td>[200,200,200]</td>
<td>3.9449e-03</td>
<td>1.6471e-02</td>
</tr>
<tr>
<td>[160,160,160]</td>
<td>4.0947e-03</td>
<td>1.6413e-02</td>
</tr>
<tr>
<td>[120,120,120]</td>
<td>3.9234e-03</td>
<td>1.6283e-02</td>
</tr>
<tr>
<td>[80,80,80]</td>
<td>3.9758e-03</td>
<td>1.6506e-02</td>
</tr>
<tr>
<td>[40,40,40]</td>
<td>3.9142e-03</td>
<td>1.6348e-02</td>
</tr>
<tr>
<td>[20,20,20]</td>
<td>4.1302e-03</td>
<td>1.7006e-02</td>
</tr>
<tr>
<td>[10,10,10]</td>
<td>4.3433e-03</td>
<td>1.7524e-02</td>
</tr>
<tr>
<td>[5,5,5]</td>
<td>5.3154e-03</td>
<td>1.9716e-02</td>
</tr>
<tr>
<td>[4,4,4]</td>
<td>5.4210e-03</td>
<td>1.9710e-02</td>
</tr>
<tr>
<td>[2,2,2]</td>
<td>6.2667e-03</td>
<td>2.2568e-02</td>
</tr>
<tr>
<td>[1,1,1]</td>
<td>7.3676e-03</td>
<td>2.6375e-02</td>
</tr>
<tr>
<td>[]</td>
<td>7.3999e-03</td>
<td>2.6097e-02</td>
</tr>
</tbody>
</table>

Embedding net size tuning form on Mg-Al: (Fitting-net size: [240,240,240])

<table>
<thead>
<tr>
<th>Embedding-net size</th>
<th>Energy L2err/Natoms(eV)</th>
<th>Force L2err(eV/Angstrom)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[25,50,100]</td>
<td>3.9606e-03</td>
<td>1.6289e-02</td>
</tr>
<tr>
<td>[20,40,80]</td>
<td>4.0292e-03</td>
<td>1.6555e-02</td>
</tr>
<tr>
<td>[15,30,60]</td>
<td>4.1743e-03</td>
<td>1.7026e-02</td>
</tr>
<tr>
<td>[10,20,40]</td>
<td>4.8138e-03</td>
<td>1.8516e-02</td>
</tr>
<tr>
<td>[5,10,20]</td>
<td>5.6052e-03</td>
<td>2.0709e-02</td>
</tr>
<tr>
<td>[4,8,16]</td>
<td>6.1335e-03</td>
<td>2.1450e-02</td>
</tr>
<tr>
<td>[3,6,12]</td>
<td>6.6469e-03</td>
<td>2.3003e-02</td>
</tr>
<tr>
<td>[2,4,8]</td>
<td>6.8222e-03</td>
<td>2.6318e-02</td>
</tr>
<tr>
<td>[1,2,4]</td>
<td>1.0678e-02</td>
<td>3.9559e-02</td>
</tr>
</tbody>
</table>

12.2 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.
12.2.1 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.

12.2.2 Parallelism between independent operators

For CPU devices, TensorFlow use multiple streams to run independent operators (OP).

```
export TF_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.

12.2.3 Parallelism within an individual operators

For CPU devices, TF_INTRA_OP_PARALLELISM_THREADS controls parallelism within TensorFlow native OPs when TensorFlow is built against Eigen.

```
export TF_INTRA_OP_PARALLELISM_THREADS=2
```

OMP_NUM_THREADS is threads for OpenMP parallelism. It controls parallelism within TensorFlow native OPs when TensorFlow is built by Intel OneDNN and DeePMD-kit custom CPU OPs. It may also control parallelsim 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. See Intel documentation for detailed information.

12.2. How to control the parallelism of a job?
12.2.4 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:

```bash
export OMP_NUM_THREADS=3
export TF_INTRA_OP_PARALLELISM_THREADS=3
export TF_INTER_OP_PARALLELISM_THREADS=2
dp train input.json
```

For a node with 128 cores, it is recommended to start with the following variables:

```bash
export OMP_NUM_THREADS=16
export TF_INTRA_OP_PARALLELISM_THREADS=16
export TF_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.

12.3 Do we need to set \(r_{cut} < \) half boxsize?

When seeking the neighbors of atom \(i\) under periodic boundary conditions, DeePMD-kit considers all \(j\) atoms within cutoff \(r_{cut}\) from atom \(i\) in all mirror cells.

So, there is no limitation on the setting of \(r_{cut}\).

PS: The reason why some software requires \(r_{cut} < \) half box size is that they only consider the nearest mirrors from the center cell. DeePMD-kit is different from them.

12.4 How to set \(sel\)?

\(sel\) is short for “selected number of atoms in \(r_{cut}\).

\(sel_a[i]\) 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 \(r_{cut}\). To ensure that the results are strictly accurate, \(sel_a[i]\) should be larger than the largest number of type \(i\) neighbors in the \(r_{cut}\).

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.
12.5 Installation

12.5.1 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++
```

12.5.2 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.

12.6 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.

12.7 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.
12.8 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.

<table>
<thead>
<tr>
<th>Model version</th>
<th>v0.12</th>
<th>v1.0</th>
<th>v1.1</th>
<th>v1.2</th>
<th>v1.3</th>
<th>v2.0</th>
<th>v2.1</th>
<th>v2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compatibility</td>
<td>😊</td>
<td>😊</td>
<td>😊</td>
<td>😊</td>
<td>😊</td>
<td>😄</td>
<td>😄</td>
<td>😄</td>
</tr>
</tbody>
</table>

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.

12.9 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.

12.9.1 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 evaluating 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.

12.9.2 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.

12.9.3 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.
After DeePMD-kit C/++ library is installed, one can find DeePMD-kit from CMake:

```bash
find_package(DeePMDB 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

```bash
target_link_libraries(some_library PRIVATE DeePMDB::deepmd_c)
```

To link against the C++ interface library, using

```bash
target_link_libraries(some_library PRIVATE DeePMDB::deepmd_cc)
```
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, trainer, 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.

### 14.1 Design a new component

When creating a new component, take descriptor as the example, you should inherit `deepmd.descriptor.descriptor.Descriptor` class and override several methods. Abstract methods such as `deepmd.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:

```python
from deepmd.descriptor import Descriptor

@Descriptor.register("some_descrpt")
class SomeDescriptor(Descriptor):
    def __init__(self, arg1: bool, arg2: float) -> None:
        pass
```

### 14.2 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

```python
from typing import List

from dargs import Argument
from deepmd.utils.argcheck import descrpt_args_plugin
```

(continues on next page)
allows one to use your new descriptor as below:

```python
"descriptor": {
    "type": "some_descrpt",
    "arg1": true,
    "arg2": 6.0
}
```

The arguments here should be consistent with the class arguments of your new component.

## 14.3 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`:

```python
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.
15.1 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 $R$. 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 $R$ can be embedded into matrix $G$. We can thus extract a descriptor vector (of $M_1 \times M_2$ dim) of the centric atom from the $G$ 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.

15.2 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
- $G_{ij}()$: Origin embedding net, take $s_{ij}$ as input and output embedding vector of $M_1$ dim
- $G()$: Shared embedding net
- Multi(): Matrix multiplication and flattening, output the descriptor vector of $M_1 \times M_2$ dim
- $F_i()$: Origin fitting net, take the descriptor vector as input and output energy
- $F()$: Shared fitting net
- $A()$: Atom type embedding net, input is atom type, the output is type embedding vector of dim nchan1

So, we can formulate the training process as follows. Vanilla DeePMD-kit algorithm:

$$E = F_i(\text{Multi}(G_{ij}(s_{ij})))$$

DeePMD-kit applying atom type embedding:

$$E = F(\text{Multi}(G([s_{ij}, A(i), A(j)]), A(j)))$$
DeePMD-kit

\[ E = F([\text{Multi}(G([s_{ij}, A(j)])), A(j)]) \]

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.

### 15.3 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
}
```

### 15.4 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.

#### 15.4.1 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.

#### 15.4.2 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 × nchan1] dimensions). We then save the type embedding vector into input_dict, so that they can be fetched later in embedding_net and fitting_net.
15.4.3 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

\[
\text{build} \to \_\text{pass}\_\text{filter} \to \_\text{filter} \to \_\text{filter}\_\text{lower}
\]

\text{\_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.

\text{\_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 Mult($\cdot$), and finally output the result of descriptor vector of $M_1 \times M_2$ dim.

\text{\_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.

15.4.4 fitting net (fit/ener.py)

In fitting net, it takes the descriptor vector as input, whose dimension is $[\text{natoms}, M_1 \times M_2]$. Because we need to involve information on the centric atom in this step, we need to generate a matrix named \text{atype\_embed} (of dim $[\text{natoms}, \text{nchanl}]$), 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 \text{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 \text{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 \text{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.
16.1 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.

16.2 Python

16.2.1 Rules

The code must be compatible with the oldest supported version of python which is 3.7

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

```python
f"something {'this' if x else 'that'}"
```  
• Use f-strings `s = f"{x:.2f}"` instead of old style formatting with "%.f" % x. string format method
```
{x:.2f}".format()
```  
maybe used sparsely where it is more convenient than f-strings.

### 16.2.2 Whitespace

Python is not C/C++ so whitespace should be used sparingly to maintain code readability

• Read the White space 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
```python
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.

### 16.2.3 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.

### 16.2.4 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](https://github.com/numpy/numpy/blob/master/doc/example.py)
16.3 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.

16.4 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.
17.1 CI

17.1.1 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.

17.2 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 EIGHTEEN

PYTHON API

18.1 backend package

18.1.1 Submodules

18.1.2 backend.dp_backend module

18.1.3 backend.dynamic_metadata module

backend.dynamic_metadata.dynamic_metadata(field: str, settings: Optional[Dict[str, object]] = None) → str

18.1.4 backend.find_tensorflow module

backend.find_tensorflow.find_tensorflow() → Tuple[Optional[str], List[str]]

Find TensorFlow library.

Tries to find TensorFlow in the order of:

1. Environment variable TENSORFLOW_ROOT if set
2. The current Python environment.
3. user site packages directory if enabled
4. system site packages directory (purelib)
5. add as a requirement (detect TENSORFLOW_VERSION or the latest) and let pip install it

Returns

str
TensorFlow library path if found.

list of str
TensorFlow requirement if not found. Empty if found.

backend.find_tensorflow.get_tf_requirement(tf_version: str = '') → dict

Get TensorFlow requirement (CPU) when TF is not installed.

If tf_version is not given and the environment variable TENSORFLOW_VERSION is set, use it as the requirement.
Parameters
tf_version
    [str, optional] TF version

Returns
dict
    TensorFlow requirement, including cpu and gpu.

backend.find_tensorflow.get_tf_version(tf_path: Union[str, Path]) → str
Get TF version from a TF Python library path.

Parameters
tf_path
    [str or Path] TF Python library path

Returns
str
    version

18.1.5 backend.read_env module

Read environment variables to configure the build.

backend.read_env.get_argument_from_env() → Tuple[str, list, list, dict, str]
Get the arguments from environment variables.

The environment variables are assumed to be not changed during the build.

Returns
str
    The minimum required CMake version.
list of str
    The CMake arguments.
list of str
    The requirements for the build.
dict
    The extra scripts to be installed.
str
    The TensorFlow version.

backend.read_env.set_scikit_build_env()
Set scikit-build environment variables before executing scikit-build.
18.2 deepmd package

Root of the deepmd package, exposes all public classes and submodules.

```python
class deepmd.DeepEval(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False,
auto_batch_size: Union[bool, int, AutoBatchSize] = False, input_map: Optional[dict] = None, neighbor_list=None)

Bases: object

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.

Attributes

model_type
    Get type of model.
model_version
    Get version of model.
sess
    Get TF session.

Methods

build_neighbor_list(coords, cell, atype, ...)  Make the mesh with neighbor list for a single
frame.

eval_typeebd()  Evaluate output of type embedding network by
using this model.

make_natoms_vec(atom_types[, mixed_type])  Make the natom vector used by deepmd-kit.

reverse_map(vec, imap)  Reverse mapping of a vector according to the in-
dex map.

sort_input(coord, atom_type[, sel_atoms, ...])  Sort atoms in the system according their types.
```
build_neighbor_list(coords: ndarray, cell: Optional[ndarray], atype: ndarray, imap: 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

coops
  [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

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]

eval_typeebd() → 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.utils.type_embed.TypeEmbedNet

The type embedding network.

Examples

Get the output of type embedding network of graph.pb:

```python
>>> from deepmd.infer import DeepPotential
>>> dp = DeepPotential('graph.pb')
>>> dp.eval_typeebd()
```

load_prefix: str

make_natoms_vec(atom_types: ndarray, mixed_type: bool = False) → ndarray

Make the natoms 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_attn.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[1]: 2 <= i < Ntypes+2, number of type i atoms

property model_type: str

Get type of model.

:type:str

property model_version: str

Get version of model.

Returns

- str
  - version of model

static reverse_map(vec: ndarray, imap: List[int]) → 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.
property sess: Session

Get TF session.

static sort_input (coord: ndarray, atom_type: ndarray, sel_atoms: Optional[List[int]] = 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.


Factory function that will inialize appropriate potential read from model_file.

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

Chapter 18. Python API
neighbor_list
    [ase.neighborlist.NeighborList, optional] The neighbor list object. If None,
    then build the native neighbor list.

Returns

    one of the available potentials

Raises

RuntimeError
    if model file does not correspond to any implemented potential

class deepmd.DipoleChargeModifier(model_name: str, model_charge_map: List[float],
    sys_charge_map: List[float], ewald_h: float = 1, ewald_beta: float = 1)

Bases: DeepDipole

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: Å

ewald_beta
    Splitting parameter of the Ewald sum. Unit: Å⁻¹

Attributes

model_type
    Get type of model.

model_version
    Get version of model.

sess
    Get TF session.
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build_fv_graph()</code></td>
<td>Build the computational graph for the force and virial inference.</td>
</tr>
<tr>
<td><code>build_neighbor_list(coords, cell, atype, ...)</code></td>
<td>Make the mesh with neighbor list for a single frame.</td>
</tr>
<tr>
<td><code>eval(coord, box, atype[, eval_fv])</code></td>
<td>Evaluate the modification.</td>
</tr>
<tr>
<td><code>eval_full(coords, cells, atom_types[, ...])</code></td>
<td>Evaluate the model with interface similar to the energy model.</td>
</tr>
<tr>
<td><code>eval_typeebd()</code></td>
<td>Evaluate output of type embedding network by using this model.</td>
</tr>
<tr>
<td><code>get_dim_aparam()</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_dim_fparam()</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Get the number of atom types of this model.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get the cut-off radius of this model.</td>
</tr>
<tr>
<td><code>get_sel_type()</code></td>
<td>Get the selected atom types of this model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map (element name of the atom types) of this model.</td>
</tr>
<tr>
<td><code>make_natoms_vec(atom_types[, mixed_type])</code></td>
<td>Make the natom vector used by deepmd-kit.</td>
</tr>
<tr>
<td><code>modify_data(data, data_sys)</code></td>
<td>Modify data.</td>
</tr>
<tr>
<td><code>reverse_map(vec, imap)</code></td>
<td>Reverse mapping of a vector according to the index map.</td>
</tr>
<tr>
<td><code>sort_input(coord, atom_type[, sel_atoms, ...])</code></td>
<td>Sort atoms in the system according their types.</td>
</tr>
</tbody>
</table>

**build_fv_graph() → Tensor**

Build the computational graph for the force and virial inference.

**eval(coord: ndarray, box: ndarray, atype: ndarray, eval_fv: bool = True) → Tuple[ndarray, ndarray, 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
modify_data(data: dict, data_sys: 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.

18.2.1 Subpackages
deepmd.cluster package
Module that reads node resources, auto detects if running local or on SLURM.

deepmd.cluster.get_resource() → Tuple[str, List[str], Optional[List[int]]]
Get local or slurm resources: nodename, nodelist, and gpus.

Returns

Tuple[str, List[str], Optional[List[int]]]

nodename, nodelist, and gpus

Submodules

deepmd.cluster.local module
Get local GPU resources.

deepmd.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.cluster.local.get_resource() → Tuple[str, List[str], Optional[List[int]]]
Get local resources: nodename, nodelist, and gpus.

Returns

Tuple[str, List[str], Optional[List[int]]]

nodename, nodelist, and gpus
deepmd.cluster.slurm module

Module to get resources on SLURM cluster.

References

https://github.com/deepsense-ai/tensorflow_on_slurm

deepmd.cluster.slurm.get_resource() → Tuple[str, List[str], Optional[List[int]]]

Get SLURM resources: nodename, nodelist, and gpus.

Returns

Tuple[str, List[str], Optional[List[int]]]

nodename, nodelist, and gpus

Raises

RuntimeError

if number of nodes could not be retrieved

ValueError

list of nodes is not of the same length as number of nodes

ValueError

if current nodename is not found in node list

deepmd.descriptor package

class deepmd.descriptor.Descriptor(*args, **kwargs)

Bases: PluginVariant

The abstract class for descriptors. All specific descriptors should be based on this class.

The descriptor $D$ 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

```python
>>> descriptor = Descriptor(type="se_e2_a", rcut=6., rcut_smth=0.5, sel=[50])

>>> type(descriptor)
<class 'deepmd.descriptor.se_a.DescrptSeA'>
```

Attributes

explicit_ntypes

Explicit ntypes with type embedding.
### Methods

- **build**(*coord_, atype_, natoms, box_, mesh,...) Build the computational graph for the descriptor.
- **build_type_exclude_mask**(exclude_types,...) Build the type exclude mask for the descriptor.
- **compute_input_stats**(data_coord, data_box, ...) Compute the statistics (avg and std) of the training data.
- **enable_compression**(min_nbor_dist, graph,...) Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
- **enable_mixed_precision**(mixed_prec) Receive the mixed precision setting.
- **get_dim_out()** Returns the output dimension of this descriptor.
- **get_dim_rot_mat_I()** Returns the first dimension of the rotation matrix.
- **get_nlist()** Returns neighbor information.
- **get_ntypes()** Returns the number of atom types.
- **get_rcut()** Returns the cut-off radius.
- **get_tensor_names**(suffix) Get names of tensors.
- **init_variables**(graph, graph_def[, suffix]) Init the embedding net variables with the given dict.
- **pass_tensors_from_frz_model**(tensors) Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz_graph_def.
- **prod_force_virial**(atom_ener, natoms) Compute force and virial.
- **register**(key) Register a descriptor plugin.
- **update_sel**(global_jdata, local_jdata) Update the selection and perform neighbor statistics.


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
DeePMD-kit

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.

build_type_exclude_mask(exclude_types: List[Tuple[int, int]], ntypes: int, sel: List[int], ndscrpt: int, atype: Tensor, shape0: Tensor) → 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.
  ndscrpt
    [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[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 neibro type with the fixed number of given sel and each neighbor has the same number of descriptors (for example 4).

References

[1]

abstract compute_input_stats(data_coord: List[ndarray], data_box: List[ndarray], data_atype: List[ndarray], natoms_vec: List[ndarray], mesh: List[ndarray], input_dict: Dict[str, List[ndarray]], **kwargs) → None

Compute the statisics (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data_coord

[1 list[np.ndarray]] The coordinates. Can be generated by deepmd.model.model_stat.make_stat_input()

data_box

[1 list[np.ndarray]] The box. Can be generated by deepmd.model.model_stat.make_stat_input()

data_atype

[1 list[np.ndarray]] The atom types. Can be generated by deepmd.model.model_stat.make_stat_input()

natoms_vec

[1 list[np.ndarray]] The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.model.model_stat.make_stat_input()

mesh

[1 list[np.ndarray]] The mesh for neighbor searching. Can be generated by deepmd.model.model_stat.make_stat_input()

input_dict

[1 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.

```python
def enable_compression(min_nbor_dist: float, graph: Graph, graph_def: 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 statisites (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.

```python
def enable_mixed_precision(mixed_prec: Optional[dict] = 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.

```python
@property
def explicit_ntypes: bool
```
Explicit ntypes with type embedding.

```python
classmethod get_class_by_input(input: dict)
```
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.

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

get_nlist() → Tuple[Tensor, 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 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_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.

**get_tensor_names**(suffix: \texttt{str} = \texttt{''}) \rightarrow \texttt{Tuple[str]}
Get names of tensors.

Parameters

suffix

\texttt{[str]} The suffix of the scope

Returns

\texttt{Tuple[str]}

Names of tensors

**init_variables**(graph: \texttt{Graph}, graph_def: \texttt{GraphDef}, suffix: \texttt{str} = \texttt{''}) \rightarrow \texttt{None}
Init the embedding net variables with the given dict.

Parameters

graph

\texttt{[tf.Graph]} The input frozen model graph

graph_def

\texttt{[tf.GraphDef]} The input frozen model graph_def

suffix

\texttt{[str, optional]} The suffix of the scope

Notes

This method is called by others when the descriptor supported initialization from the given variables.

**pass_tensors_from_frz_model**(\*tensors: \texttt{Tensor}) \rightarrow \texttt{None}
Pass the descrpt reshape tensor as well as descrpt deriv tensor from the frz graph_def.

Parameters

\*tensors

\texttt{[tf.Tensor]} passed tensors

Notes

The number of parameters in the method must be equal to the numbers of returns in \texttt{get_tensor_names}.  

**abstract prod_force_virial**(atom_ener: \texttt{Tensor}, natoms: \texttt{Tensor}) \rightarrow \texttt{Tuple[Tensor, Tensor, Tensor]}
Compute force and virial.

Parameters

atom_ener

\texttt{[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

static register(key: str) → Callable

Register a descriptor plugin.

Parameters

key

[ str ] the key of a descriptor

Returns

Descriptor

the registered descriptor

Examples

>>> @Descriptor.register("some_descrpt")
class SomeDescript(Descriptor):
    pass

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

class deepmd.descriptor.DescrptHybrid(*args, **kwargs)

Bases: Descriptor

Concate a list of descriptors to form a new descriptor.

Parameters

list

[ list ] Build a descriptor from the concatenation of the list of descriptors.

Attributes

explicit_ntypes

Explicit ntypes with type embedding.
### Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box_, mesh, ...)</code></td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><code>build_type_exclude_mask(exclude_types, ...)</code></td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><code>compute_input_stats(data_coord, data_box, ...)</code></td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><code>enable_compression(min_nbor_dist, graph, ...)</code></td>
<td>Reieve the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><code>enable_mixed_precision([mixed_prec])</code></td>
<td>Reieve the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_dim_out()</code></td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><code>get_dim_rot_mat_1()</code></td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><code>get_nlist()</code></td>
<td>Get the neighbor information of the descriptor, returns the nlist of the descriptor with the largest cut-off radius.</td>
</tr>
<tr>
<td><code>get_nlist_i(ii)</code></td>
<td>Get the neighbor information of the ii-th descriptor.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><code>get_tensor_names([suffix])</code></td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, suffix])</code></td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><code>merge_input_stats(stat_dict)</code></td>
<td>Merge the statisitics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td><code>pass_tensors_from_frz_model(*tensors)</code></td>
<td>Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz_graph_def.</td>
</tr>
<tr>
<td><code>prod_force_virial(atom_ener, natoms)</code></td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```python
build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: dict,
       reuse: Optional[bool] = None, suffix: str = '') → 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
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_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: Optional[list] = None, **kwargs) → None

Compute the statistics (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.model.make_stat_input
data_box
The box. Can be generated by deepmd.model.make_stat_input
data_atype
The atom types. Can be generated by deepmd.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.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.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: Graph, graph_def: 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 statistics (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: Optional[dict] = None) → None
    Retrieve the mixed precision setting.

Parameters

mixed_prec
    The mixed precision setting used in the embedding net

property explicit_ntypes: bool
    Explicit ntypes with type embedding.

def get_dim_out() → int
    Returns the output dimension of this descriptor.

def get_nlist() → Tuple[Tensor, 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

def get_nlist_i(ii: int) → Tuple[Tensor, 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

get_ntypes() → int
  Returns the number of atom types.

get_rcut() → float
  Returns the cut-off radius.

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

init_variables(graph: Graph, graph_def: 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

merge_input_stats(stat_dict)
  Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict
  The dict of statistics computed from compute_input_stats, including:

  sumr
    The sum of radial statistics.

  suma
    The sum of relative coord statistics.
DeePMD-kit

```
sumn
    The sum of neighbor numbers.
sumr2
    The sum of square of radial statsitcs.
suma2
    The sum of square of relative coord statsitcs.

pass_tensors_from_frz_model(*tensors: Tensor) → None
    Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.
    Parameters
        *tensors
            [tf.Tensor] passed tensors

prod_force_virial(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, 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

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.descriptor.DescrptLocFrame(*args, **kwargs)
    Bases: 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
```
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.

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.

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.

**Attributes**

**explicit_ntypes**

Explicit ntypes with type embedding.
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_ , atype_ , natoms , box_ , mesh , ...)</code></td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><code>build_type_exclude_mask(exclude_types , ...)</code></td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><code>compute_input_stats(data_coord , data_box , ...)</code></td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><code>enable_compression(min_nbor_dist , graph , ...)</code></td>
<td>Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><code>enable_mixed_precision()</code></td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_dim_out()</code></td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><code>get_dim_rot_mat_1()</code></td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><code>get_nlist()</code></td>
<td>Returns</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><code>get_rot_mat()</code></td>
<td>Get rotational matrix.</td>
</tr>
<tr>
<td><code>get_tensor_names([suffix])</code></td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><code>init_variables(graph , graph_def[, suffix])</code></td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><code>pass_tensors_from_frz_model(*tensors)</code></td>
<td>Pass the descrptreshape tensor as well as descrpt_deriv tensor from the frz graph_def.</td>
</tr>
<tr>
<td><code>prod_force_virial(atom_ener , natoms)</code></td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata , local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```python
build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → 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
    - 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_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list,
    input_dict: dict, **kwargs) → None

Compute the statistics (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.model.make_stat_input

    data_box
        The box. Can be generated by deepmd.model.make_stat_input

    data_atype
        The atom types. Can be generated by deepmd.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.model.make_stat_input

    mesh
        The mesh for neighbor searching. Can be generated by deepmd.model.make_stat_input

    input_dict
        Dictionary for additional input

    **kwargs
        Additional keyword arguments.

get_dim_out() → int

Returns the output dimension of this descriptor.

get_nlist() → Tuple[Tensor, Tensor, List[int], List[int]]

Returns

    nlist
        Neighbor list

    rj
        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_ntypes() → int
Returns the number of atom types.

getrcut() → float
Returns the cut-off radius.

gerotmat() → Tensor
Get rotational matrix.

init_variables(graph: Graph, graph_def: 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

prod_force_virial(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, 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[1]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
The total virial

atom_virial
The atomic virial

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.descriptor.DescrptSeA(*args, **kwargs)
Bases: 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 \( \mathcal{R}^i \in \mathcal{R}^{N \times 4} \) is the coordinate matrix, and each row of \( \mathcal{R}^i \) can be constructed as follows
\[
(\mathcal{R}^i)_j = \begin{bmatrix}
\frac{s(r_{ji})}{s(r_{ji})x_{ji}} \\
\frac{s(r_{ji})y_{ji}}{s(r_{ji})z_{ji}} \\
\frac{s(r_{ji})z_{ji}}{r_{ji}}
\end{bmatrix}
\]
where \( R_{ji} = R_j - R_i = (x_{ji}, y_{ji}, z_{ji}) \) is the relative coordinate and \( r_{ji} = ||R_{ji}|| \) is its norm. The switching function \( s(r) \) is defined as:
\[
s(r) = \begin{cases} 
\frac{1}{r}, & r < r_s \\
\frac{1}{r_s} \left( \left( \frac{r - r_s}{r_c - r_s} \right)^3 - 6 \left( \frac{r - r_s}{r_c - r_s} \right)^2 + 15 \frac{r - r_s}{r_c - r_s} - 10 \right) + 1, & r_s \leq r < r_c \\
0, & r \geq r_c
\end{cases}
\]
Each row of the embedding matrix \( \mathcal{G}^i \in \mathcal{R}^{N \times M_1} \) consists of outputs of a embedding network \( \mathcal{N} \) of \( s(r_{ji}) \):
\[
(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))
\]
\( \mathcal{G}^i_{<} \in \mathcal{R}^{N \times M_2} \) takes first \( M_2 \) columns of \( \mathcal{G}^i \). The equation of embedding network \( \mathcal{N} \) can be found at \texttt{deepmd.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[str]] sel[i] specifies the maximum 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: \( y = x + dt \ast \phi(Wx + b) \)
- **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.
DeePMD-kit

set_davg_zero
Set the shift of embedding net input to zero.

activation_function
The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.

precision
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

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.

References
[1]

Attributes

explicit_ntypes
Explicit ntypes with type embedding.

precision
Precision of filter network.
### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box_, mesh, ...)</code></td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><code>build_type_exclude_mask(exclude_types, ...)</code></td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><code>compute_input_stats(data_coord, data_box, ...)</code></td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><code>enable_compression(min_nbor_dist, graph, ...)</code></td>
<td>Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><code>enable_mixed_precision([mixed_prec])</code></td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_dim_out()</code></td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><code>get_dim_rot_mat_1()</code></td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><code>get_nlist()</code></td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><code>get_rot_mat()</code></td>
<td>Get rotational matrix.</td>
</tr>
<tr>
<td><code>get_tensor_names([suffix])</code></td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><code>merge_input_stats(stat_dict)</code></td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td><code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code></td>
<td>Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.</td>
</tr>
<tr>
<td><code>prod_force_virial(atom_ener, natoms)</code></td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```python
build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: dict, 
      reuse: Optional[bool] = None, suffix: str = '') → 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_`: The box of the system.
- `mesh`: For historical reasons, only the length of the Tensor matters. If size of mesh == 6,
DeePMD-kit

DeePMD-kit

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_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None

Compute the statistics (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.model.make_stat_input

data_box
The box. Can be generated by deepmd.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.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.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.model.make_stat_input

input_dict
Dictionary for additional input

**kwargs
Additional keyword arguments.

enable_compression(min_nbor_dist: float, graph: Graph, graph_def: GraphDef, table_extrapolate: float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None

Receive the statistics (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
DeePMD-kit

**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: Optional[dict] = None) → None*
Reveive the mixed precision setting.

**Parameters**

**mixed_prec**
The mixed precision setting used in the embedding net

**property explicit_ntypes**: **bool**
Explicit ntypes with type embedding.

**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[Tensor, 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

**get_ntypes() → int**
Returns the number of atom types.

**get_rcut() → float**
Returns the cut-off radius.

**get_rot_mat() → Tensor**
Get rotational matrix.

**init_variables** *(graph: Graph, graph_def: GraphDef, suffix: str = '') → None*
Init the embedding net variables with the given dict.

**Parameters**

18.2. deepmd package
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

merge_input_stats(stat_dict)
    Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters
    stat_dict
        The dict of statistics computed from compute_input_stats, including:

        sumr
            The sum of radial statistics.

        suma
            The sum of relative coord statistics.

        sumn
            The sum of neighbor numbers.

        sumr2
            The sum of square of radial statistics.

        suma2
            The sum of square of relative coord statistics.

prod_force_virial(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, 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.descriptor.DescrptSeAEbd(*args, **kwargs)
    Bases: 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
  \([\text{list[\text{str}]}]\) \(\text{sel}[\text{i}]\) specifies the maximum number of type \(\text{i}\) atoms in the cut-off radius
neuron
  \([\text{list[\text{int}]}]\) 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 \(\text{dt}\) in the resnet construction: \(y = x + \text{dt} \times \phi(Wx + b)\)
trainable
  If the weights of embedding net are trainable.
seed
  Random seed for initializing the network parameters.
type_one_side
  Try to build \(\text{N}\) types embedding nets. Otherwise, building \(\text{N}\) 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
  \([\text{List[\text{List[\text{int}]}}]]\) 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.

Attributes

  \textbf{explicit_ntypes}
  Explicit ntypes with type embedding.

  \textbf{precision}
  Precision of filter network.
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box_, mesh, ...)</code></td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><code>build_type_exclude_mask(exclude_types, ...)</code></td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><code>compute_input_stats(data_coord, data_box, ...)</code></td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><code>enable_compression(min_nbor_dist, graph, ...)</code></td>
<td>Receiving the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><code>enable_mixed_precision([mixed_prec])</code></td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_dim_out()</code></td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><code>get_dim_rot_mat_1()</code></td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><code>get_nlist()</code></td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><code>get_rot_mat()</code></td>
<td>Get rotational matrix.</td>
</tr>
<tr>
<td><code>get_tensor_names([suffix])</code></td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def, [suffix])</code></td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><code>merge_input_stats(stat_dict)</code></td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td><code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code></td>
<td>Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.</td>
</tr>
<tr>
<td><code>prod_force_virial(atom_ener, natoms)</code></td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```python
build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: dict, reuse: Optional[bool] = None, suffix: str = '') -> 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,
DeePMD-kit

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

class deepmd.descriptor.DescrptSeAEbdV2(*args, **kwargs)
Bases: DescrptSeA

A compressible se_a_ebd model.

This model is a warpper for DescriptorSeA, which set stripped_type_embedding=True.

Attributes

   explicit_ntypes
      Explicit ntypes with type embedding.

   precision
      Precision of filter network.
Methods

```python
build(coord_, atype_, natoms, box_, mesh, ...)  Build the computational graph for the descriptor.
build_type_exclude_mask(exclude_types, ...)   Build the type exclude mask for the descriptor.
compute_input_stats(data_coord, data_box, ...) Compute the statistics (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...) Reiveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec]) Reiveive the mixed precision setting.
get_dim_out() Returns the output dimension of this descriptor.
get_dim_rot_mat_1() Returns the first dimension of the rotation matrix.
get_nlist() Returns neighbor information.
get_ntypes() Returns the number of atom types.
get_rcut() Returns the cut-off radius.
get_rot_mat() Get rotational matrix.
get_tensor_names([suffix]) Get names of tensors.
init_variables(graph, graph_def[, suffix]) Init the embedding net variables with the given dict.
merge_input_stats(stat_dict) Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
pass_tensors_from_frz_model(descrpt_reshape, ...) Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.
prod_force_virial(atom_ener, natoms) Compute force and virial.
register(key) Register a descriptor plugin.
update_sel(global_jdata, local_jdata) Update the selection and perform neighbor statistics.
```

class deepmd.descriptor.DescrptSeAEf(*args, **kwargs)
Bases: DescrptSe
Smooth edition descriptor with Ef.

Parameters

cut
The cut-off radius

cut_smth
From where the environment matrix should be smoothed

sel
[list[str]] sel[i] specifies the maximum 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: \( y = x + dt \cdot \phi(Wx + b) \)

trainable
    If the weights of embedding net are trainable.

seed
    Random seed for initializing the network parameters.

type_one_side
    Try to build \( N \) embedding nets. Otherwise, building \( N^2 \) embedding nets

exclude_types
    \([\text{List}[	ext{List}[	ext{int}]]] \) 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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.

precision
    The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

uniform_seed
    Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

Attributes

explicit_ntypes
    Explicit ntypes with type embedding.

precision
    Precision of filter network.
### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box_, mesh, ...)</code></td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><code>build_type_exclude_mask(exclude_types, ...)</code></td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><code>compute_input_stats(data_coord, data_box, ...)</code></td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><code>enable_compression(min_nbor_dist, graph, ...)</code></td>
<td>Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><code>enable_mixed_precision([mixed_prec])</code></td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_dim_out()</code></td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><code>get_dim_rot_mat_1()</code></td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><code>get_nlist()</code></td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><code>get_rot_mat()</code></td>
<td>Get rotational matrix.</td>
</tr>
<tr>
<td><code>get_tensor_names([suffix])</code></td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, suffix])</code></td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><code>pass_tensors_from_frz_model(descript_reshape, ...)</code></td>
<td>Pass the descript_reshape tensor as well as descript_deriv tensor from the frz graph_def.</td>
</tr>
<tr>
<td><code>prod_force_virial(atom_ener, natoms)</code></td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

**build**

```python
build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: dict, 
        reuse: Optional[bool] = None, suffix: str = '') → 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[1]: 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

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None

Compute the statistics (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.model.make_stat_input

  data_box
    The box. Can be generated by deepmd.model.make_stat_input

  data_atype
    The atom types. Can be generated by deepmd.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.model.make_stat_input

  mesh
    The mesh for neighbor searching. Can be generated by deepmd.model.make_stat_input

  input_dict
    Dictionary for additional input

  **kwargs
    Additional keyword arguments.

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[Tensor, 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
get_ntypes() → int
Returns the number of atom types.

get_rcut() → float
Returns the cut-off radius.

get_rot_mat() → Tensor
Get rotational matrix.

prod_force_virial(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, 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[2]: 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.descriptor.DescrptSeAEfLower(*args, **kwargs)
Bases: DescriptSeA
Helper class for implementing DescriptSeAEf.

Attributes

explicit_ntypes
Explicit ntypes with type embedding.

precision
Precision of filter network.
### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build</code></td>
<td>Build the computational graph for the descriptor. Parameters: coord, atype, natoms, box, mesh, input_dict, suffix='', reuse=None.</td>
</tr>
<tr>
<td><code>build_type_exclude_mask</code></td>
<td>Build the type exclude mask for the descriptor. Parameters: exclude_types, ...</td>
</tr>
<tr>
<td><code>compute_input_stats</code></td>
<td>Compute the statistics (avg and std) of the training data. Parameters: data_coord, data_box, ...</td>
</tr>
<tr>
<td><code>enable_compression</code></td>
<td>Retrieve the statistics (distance, max_nbor_size and env_mat_range) of the training data. Parameters: min_nbor_dist, graph, ...</td>
</tr>
<tr>
<td><code>enable_mixed_precision</code></td>
<td>Retrieve the mixed precision setting. Parameters: [mixed_prec]</td>
</tr>
<tr>
<td><code>get_dim_out</code></td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><code>get_dim_rot_mat_1</code></td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><code>get_nlist</code></td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td><code>get_ntypes</code></td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><code>get_rcut</code></td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><code>get_rot_mat</code></td>
<td>Get rotational matrix.</td>
</tr>
<tr>
<td><code>get_tensor_names</code></td>
<td>Get names of tensors. Parameters: [suffix]</td>
</tr>
<tr>
<td><code>init_variables</code></td>
<td>Init the embedding net variables with the given dict. Parameters: graph, graph_def[, suffix].</td>
</tr>
<tr>
<td><code>merge_input_stats</code></td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd. Parameters: stat_dict.</td>
</tr>
<tr>
<td><code>pass_tensors_from_frz_model</code></td>
<td>Pass the descrpt reshape tensor as well as descrp_term tensor from the frz graph_def. Parameters: descrpt_reshape, ...</td>
</tr>
<tr>
<td><code>register</code></td>
<td>Register a descriptor plugin. Parameters: key.</td>
</tr>
<tr>
<td><code>update_sel</code></td>
<td>Update the selection and perform neighbor statistics. Parameters: global_jdata, local_jdata.</td>
</tr>
</tbody>
</table>

### 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

descrptor
    The output descriptor

compute_input_stats(data_coord, data_box, data_atype, natoms_vec, mesh, input_dict, **kwargs)
    Compute the statistics (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.model.make_stat_input
data_box
    The box. Can be generated by deepmd.model.make_stat_input
data_atype
    The atom types. Can be generated by deepmd.model.make_stat_input
natom_vec
    The vector for the number of atoms of the system and different types of atoms.
    Can be generated by deepmd.model.make_stat_input
mesh
    The mesh for neighbor searching. Can be generated by deepmd.model.make_stat_input
input_dict
    Dictionary for additional input
**kwargs
    Additional keyword arguments.

class deepmd.descriptor.DescrptSeAMask(*args, **kwargs)
Bases: 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 $D^i \in R^{M_1 \times M_2}$ is given by [1]

$$D^i = (G^i)^T R^i (G^i)^T G^i$$

where $R^i \in R^{N \times 4}$ is the coordinate matrix, and each row of $R^i$ can be constructed as follows

$$(R^i)_j = \begin{bmatrix}
    s(r_{ji}) \\
    s(r_{ji}) x_{ji} \\
    s(r_{ji}) y_{ji} \\
    s(r_{ji}) z_{ji} \\
\end{bmatrix}$$

where $s(r_{ji})$ is the smooth function.
where $R_{ji} = R_j - R_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \|R_{ji}\|$ is its norm. The switching function $s(r)$ is defined as:

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

Each row of the embedding matrix $G_i \in \mathbb{R}^{N \times M_1}$ consists of outputs of an embedding network $N$ of $s(r_{ji})$:

\[(G^i)_j = N(s(r_{ji}))\]

$G_i^j \in \mathbb{R}^{N \times M_2}$ takes first $M_2$ columns of $G^i$. The equation of embedding network $N$ can be found at `deepmd.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 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` \[\text{list[str]}\]\[\text{sel[i]}\] specifies the maximum number of type $i$ atoms in the neighbor list.
- `neuron` \[\text{list[int]}\] Number of neurons in each hidden layers of the embedding net $N$.
- `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: $y = x + dt \cdot \phi(Wx + b)$.
- `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` \[\text{List[List[int]]}\] 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]

Attributes

**explicit_ntypes**
Explicit ntypes with type embedding.

**precision**
Precision of filter network.

Methods

```python
build(coord_, atype_, natoms, box_, mesh, ...) Build the computational graph for the descriptor.
build_type_exclude_mask(exclude_types, ...) Build the type exclude mask for the descriptor.
compute_input_stats(data_coord, data_box, ...) Compute the statistics (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...) Reive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec]) Reive the mixed precision setting.
get_dim_out() Returns the output dimension of this descriptor.
get_dim_rot_mat_1() Returns the first dimension of the rotation matrix.
get_nlist() Returns neighbor information.
get_ntypes() Returns the number of atom types.
get_rcut() Returns the cutoff radius.
get_rot_mat() Get rotational matrix.
get_tensor_names([suffix]) Get names of tensors.
init_variables(graph, graph_def[, suffix]) Init the embedding net variables with the given dict.
merge_input_stats(stat_dict) Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
pass_tensors_from_frz_model(descrpt_reshape, ...) Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz_graph_def.
prod_force_virial(atom_ener, natoms) Compute force and virial.
register(key) Register a descriptor plugin.
update_sel(global_jdata, local_jdata) Update the selection and perform neighbor statistics.
```

```
build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: Dict[str, Any], reuse: Optional[bool] = None, suffix: str = '') → Tensor
Build the computational graph for the descriptor.

Parameters

- coord
  The coordinate of atoms
```
DeePMD-kit

\texttt{atype} \\
The type of atoms

\texttt{natoms} \\
The number of atoms. This tensor has the length of \texttt{Ntypes + 2 natoms[0]: number} \texttt{of local atoms natoms[1]: total number of atoms held by this processor natoms[1]: 2 <= i < Ntypes+2, number of type i atoms}

\texttt{box} \\
\texttt{[tf.Tensor]} The box of the system

\texttt{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.

\texttt{input_dict} \\
Dictionary for additional inputs

\texttt{reuse} \\
The weights in the networks should be reused when get the variable.

\texttt{suffix} \\
Name suffix to identify this descriptor

\textbf{Returns} \\
\texttt{descriptor} \\
The output descriptor

\textbf{compute_input_stats} \texttt{(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) \rightarrow None} \\
Compute the statistics (avg and std) of the training data. The input will be normalized by the statistics.

\textbf{Parameters} \\
\texttt{data_coord} \\
The coordinates. Can be generated by deepmd.model.make_stat_input

\texttt{data_box} \\
The box. Can be generated by deepmd.model.make_stat_input

\texttt{data_atype} \\
The atom types. Can be generated by deepmd.model.make_stat_input

\texttt{natoms_vec} \\
The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.model.make_stat_input

\texttt{mesh} \\
The mesh for neighbor searching. Can be generated by deepmd.model.make_stat_input

\texttt{input_dict} \\
Dictionary for additional input

\texttt{**kwargs} \\
Additional keyword arguments.

\textbf{get_rcut} \rightarrow \texttt{float} \\
Returns the cutoff radius.
**prod_force_virial** (atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, 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.descriptor.DescrptSeAtten**(*args, **kwargs)
Smooth version descriptor with attention.

Parameters

- **rcut**
  The cut-off radius $r_c$

- **rcut_smth**
  From where the environment matrix should be smoothed $r_s$

- **sel**
  [list[str]] sel[i] specifies the maximum number of type i atoms in the cut-off radius

- **neuron**
  [list[int]] Number of neurons in each hidden layers of the embedding net $N$

- **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
    \[[\text{List}[\text{List}[\text{int}]]]\] 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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.

precision
    The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

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.

stripped_type_embedding
    Whether to strip the type embedding into a separated embedding network. Default value will be True in se_atten_v2 descriptor.

smooth_type_embedding
    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.

Raises
    \[\text{ValueError}\]
    if ntypes is 0.

Attributes

explicit_ntypes
    Explicit ntypes with type embedding.

precision
    Precision of filter network.
## Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>build</strong>(coord_, atype_, natoms, box_, mesh, ...)</td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><strong>build_type_exclude_mask</strong>(exclude_types, ...)</td>
<td>Build the type exclude mask for the attention descriptor.</td>
</tr>
<tr>
<td><strong>compute_input_stats</strong>(data_coord, data_box, ...)</td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><strong>enable_compression</strong>(min_nbor_dist, graph, ...)</td>
<td>Retrieve the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><strong>enable_mixed_precision</strong>(mixed_prec)</td>
<td>Retrieve the mixed precision setting.</td>
</tr>
<tr>
<td><strong>get_dim_out</strong></td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><strong>get_dim_rot_mat_1</strong></td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><strong>get_nlist</strong></td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td><strong>get_ntypes</strong></td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><strong>get_rcut</strong></td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><strong>get_rot_mat</strong></td>
<td>Get rotational matrix.</td>
</tr>
<tr>
<td><strong>get_tensor_names</strong>(suffix)</td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><strong>init_variables</strong>(graph, graph_def[, suffix])</td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><strong>merge_input_stats</strong>(stat_dict)</td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td><strong>pass_tensors_from_frz_model</strong>(descrpt_reshape, ...)</td>
<td>Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.</td>
</tr>
<tr>
<td><strong>prod_force_virial</strong>(atom_ener, natoms)</td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><strong>register</strong>(key)</td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><strong>update_sel</strong>(global_jdata, local_jdata)</td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```python
get_class_by_input
```


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

build_type_exclude_mask(exclude_types: List[Tuple[int, int]], ntypes: int, sel: List[int], ndescript: int, atype: Tensor, shape0: Tensor, nei_type_vec: Tensor) → 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.

ndescript

[ 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 size of (shape0, ndescript), 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:

depem.descriptor.descriptor.Descriptor.build_type_exclude_mask
Notes

This method has the similar way to build the type exclude mask as `deepmd.descriptor.Descriptor.build_type_exclude_mask()`. The mathematical 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.

```python
def 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: Optional[list] = None, **kwargs) -> None
```

Compute the statistics (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.model.make_stat_input`

- **data_box**
  The box. Can be generated by `deepmd.model.make_stat_input`

- **data_atype**
  The atom types. Can be generated by `deepmd.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.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_attn.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.

```python
def enable_compression(min_nbor_dist: float, graph: Graph, graph_def: GraphDef, table_extrapolate: float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') -> None
```

Retrieve the statistics (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

**property explicit_ntypes:** bool
   Explicit ntypes with type embedding.

**init_variables**(graph: Graph, graph_def: 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.descriptor.DescrptSeAttenV2**(*args, **kwargs)

   Bases: 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
         [list[str]] sel[i] specifies the maximum number of type i atoms in the cut-off radius
      neuron
         [list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$
DeePMD-kit

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: \( y = x + dt \cdot \phi(Wx + b) \)

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
[\[\text{List[\text{List[int]}]}\]] The excluded pairs of types which have no interaction with each other. For example, \[\text{[\text{[0, 1]}]}\] 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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.

precision
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

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.

Attributes

\textit{explicit_ntypes}
Explicit ntypes with type embedding.

\textit{precision}
Precision of filter network.
Methods

```python
build(coord_, atype_, natoms, box_, mesh, ...)  Build the computational graph for the descriptor.
build_type_exclude_mask(exclude_types, ...)  Build the type exclude mask for the attention descriptor.
compute_input_stats(data_coord, data_box, ...)  Compute the statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)  Reiveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec])  Reiveive the mixed precision setting.
get_dim_out()  Returns the output dimension of this descriptor.
get_dim_rot_mat_1()  Returns the first dimension of the rotation matrix.
get_nlist()  Returns neighbor information.
get_ntypes()  Returns the number of atom types.
get_rcut()  Returns the cut-off radius.
get_rot_mat()  Get rotational matrix.
get_tensor_names([suffix])  Get names of tensors.
init_variables(graph, graph_def[, suffix])  Init the embedding net variables with the given dict.
merge_input_stats(stat_dict)  Merge the statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.
pass_tensors_from_frz_model(descrpt reshape, ...)  Pass the descrpt reshape tensor as well as descrpt deriv tensor from the frz graph_def.
prod_force_virial(atom_ener, natoms)  Compute force and virial.
register(key)  Register a descriptor plugin.
update_sel(global_jdata, local_jdata)  Update the selection and perform neighbor statistics.
```

```python
class deepmd.descriptor.DescrptSeR(*args, **kwargs)
Bases: 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[str]] sel[i] specifies the maximum 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 "relu", "relu6", "softplus", "sigmoid", "tanh", "gelu", "gelu_tf", "None", "none".

precision
  The precision of the embedding net parameters. Supported options are "default", "float16", "float32", "float64", "bfloat16".

uniform_seed
  Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

Attributes

explicit_ntypes
  Explicit ntypes with type embedding.

precision
  Precision of filter network.
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box_, mesh, ...)</code></td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><code>build_type_exclude_mask(exclude_types, ...)</code></td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><code>compute_input_stats(data_coord, data_box, ...)</code></td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><code>enable_compression(min_nbor_dist, graph, ...)</code></td>
<td>Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_dim_out()</code></td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><code>get_dim_rot_mat_1()</code></td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><code>get_nlist()</code></td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><code>get_tensor_names([suffix])</code></td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, suffix])</code></td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><code>merge_input_stats(stat_dict)</code></td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td><code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code></td>
<td>Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.</td>
</tr>
<tr>
<td><code>prod_force_virial(atom_ener, natoms)</code></td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```python
build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → 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

compute_input_stats(data_coord, data_box, data_atype, natoms_vec, mesh, input_dict, **kwargs)
    Compute the statistics (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.model.make_stat_input
    data_box
        The box. Can be generated by deepmd.model.make_stat_input
    data_atype
        The atom types. Can be generated by deepmd.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.model.make_stat_input
    mesh
        The mesh for neighbor searching. Can be generated by deepmd.model.make_stat_input
    input_dict
        Dictionary for additional input
    **kwargs
        Additional keyword arguments.

enable_compression(min_nbor_dist: float, graph: Graph, graph_def: GraphDef, table_extrapolate: float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None
    Receiv the statistics (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
DeePMD-kit

- `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

**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
    ```

**get_ntypes()**
- Returns the number of atom types.

**get_rcut()**
- Returns the cut-off radius.

**merge_input_stats**(stat_dict)
- Merge the statistics computed from `compute_input_stats` to obtain the self.davg and self.dstd.
  - Parameters
    ```
    stat_dict
    The dict of statistics computed from `compute_input_stats`, including:
    ```
    ```
    sumr
    The sum of radial statistics.
    sumn
    The sum of neighbor numbers.
    sumr2
    The sum of square of radial statistics.
    ```

**prod_force_virial**(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, 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.descriptor.DescrptSeT(*args, **kwargs)

Bases: 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[str]] sel[i] specifies the maximum 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 “relu”, “relu6”,
“softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.

precision
The precision of the embedding net parameters. Supported options are “default”,
“float16”, “float32”, “float64”, “bfloat16”.

uniform_seed
Only for the purpose of backward compatibility, retrieves the old behavior of using
the random seed.

Attributes
explicit_ntypes
Explicit ntypes with type embedding.

precision
Precision of filter network.

**Methods**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box_, mesh, ...)</code></td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><code>build_type_exclude_mask(exclude_types, ...)</code></td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><code>compute_input_stats(data_coord, data_box, ...)</code></td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><code>enable_compression(min_nbor_dist, graph, ...)</code></td>
<td>Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><code>enable_mixed_precision([mixed_prec])</code></td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_dim_out()</code></td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><code>get_dim_rot_mat_1()</code></td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><code>get_nlist()</code></td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><code>get_tensor_names([suffix])</code></td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, suffix])</code></td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><code>merge_input_stats(stat_dict)</code></td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td><code>pass_tensors_from_frz_model(descript_reshape, ...)</code></td>
<td>Pass the descript_reshape tensor as well as descript_deriv tensor from the frz graph_def.</td>
</tr>
<tr>
<td><code>prod_force_virial(atom_ener, natoms)</code></td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```
build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → 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

**18.2. deepmd package**
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_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, 
input_dict: dict, **kwargs) → None

Compute the statistics (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.model.make_stat_input

data_box  
The box. Can be generated by deepmd.model.make_stat_input

data_atype  
The atom types. Can be generated by deepmd.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.model.make_stat_input

mesh  
The mesh for neighbor searching. Can be generated by 
depemd.model.make_stat_input

input_dict  
Dictionary for additional input

**kwargs  
Additional keyword arguments.

enable_compression(min_nbor_dist: float, graph: Graph, graph_def: 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 statistics (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

get_dim_out() → int
    Returns the output dimension of this descriptor.

get_nlist() → Tuple[Tensor, 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

get_ntypes() → int
    Returns the number of atom types.

get_rcut() → float
    Returns the cut-off radius.

merge_input_stats(stat_dict)
    Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict
    The dict of statistics computed from compute_input_stats, including:

sumr
    The sum of radial statistics.

suma
    The sum of relative coord statistics.

sumn
    The sum of neighbor numbers.
sumr^2
The sum of square of radial statistics.
suma^2
The sum of square of relative coordinate statistics.

**prod_force_virial** *(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, 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[1]: 2 <= i < Ntypes+2, number of type i atoms

**Returns**

force
The force on atoms

virial
The total virial

atom_virial
The atomic virial

**Submodules**

**deepmd.descriptor.descriptor module**

class deepmd.descriptor.descriptor.Descriptor(*args, **kwargs)

Bases: PluginVariant

The abstract class for descriptors. All specific descriptors should be based on this class.

The descriptor \( D \) 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

```python
>>> descript = Descriptor(type="se_e2_a", rcut=6., rcut_smth=0.5, sel=[50])
>>> type(descript)
<class 'deepmd.descriptor.se_e2_a.DescrptSeA'>
```

Attributes

- `explicit_ntypes`
  
  Explicit ntypes with type embedding.

Methods

- `build(coord_, atype_, natoms, box_, mesh, ...)`
  
  Build the computational graph for the descriptor.

- `build_type_exlude_mask(exclude_types, ...)`
  
  Build the type exclude mask for the descriptor.

- `compute_input_stats(data_coord, data_box, ...)`
  
  Compute the statistics (avg and std) of the training data.

- `enable_compression(min_nbor_dist, graph, ...)`
  
  Reiveve the statistics (distance, max_nbor_size and env_mat_range) of the training data.

- `enable_mixed_precision([mixed_prec])`
  
  Reiveve the mixed precision setting.

- `get_dim_out()`
  
  Returns the output dimension of this descriptor.

- `get_dim_rot_mat_1()`
  
  Returns the first dimension of the rotation matrix.

- `get_nlist()`
  
  Returns neighbor information.

- `get_ntypes()`
  
  Returns the number of atom types.

- `get_rcut()`
  
  Returns the cut-off radius.

- `get_tensor_names([suffix])`
  
  Get names of tensors.

- `init_variables(graph, graph_def[, suffix])`
  
  Init the embedding net variables with the given dict.

- `pass_tensors_from_frz_model(*tensors)`
  
  Pass the descrpt reshape tensor as well as descrt deriv tensor from the frz graph_def.

- `prod_force_virial(atom_ener, natoms)`
  
  Compute force and virial.

- `register(key)`
  
  Register a descriptor plugin.

- `update_sel(global_jdata, local_jdata)`
  
  Update the selection and perform neighbor statistics.

```python
abstract build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: Dict[str, Any], reuse: Optional[bool] = None, suffix: str = '') → 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.

build_type_exclude_mask(exclude_types: List[Tuple[int, int]], ntypes: int, sel: List[int], ndescrpt: int, atype: Tensor, shape0: Tensor) → 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 nsam-
    ples * 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, 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 neighbor type with the fixed number of given sel and each neighbor has the same number of descriptors (for example 4).

References

[1]

```python
abstract compute_input_stats(data_coord: List[ndarray], data_box: List[ndarray], data_atype: List[ndarray], natoms_vec: List[ndarray], mesh: List[ndarray], input_dict: Dict[str, List[ndarray]], **kwargs) → None
```

Compute the statistics (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.model.model_stat.make_stat_input()`
- `data_box` [list[np.ndarray]] The box. Can be generated by `deepmd.model.model_stat.make_stat_input()`
- `data_atype` [list[np.ndarray]] The atom types. Can be generated by `deepmd.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.model.model_stat.make_stat_input()`
- `mesh` [list[np.ndarray]] The mesh for neighbor searching. Can be generated by `deepmd.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.

\texttt{enable\_compression} (min\_nbor\_dist: float, graph: Graph, graph\_def: 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 statisites (distance, max\_nbor\_size and env\_mat\_range) of the training data.

Parameters

- \texttt{min\_nbor\_dist} [float] The nearest distance between atoms
- \texttt{graph} [tf.Graph] The graph of the model
- \texttt{graph\_def} [tf.GraphDef] The graph definition of the model
- \texttt{table\_extrapolate} [float, default: 5.] The scale of model extrapolation
- \texttt{table\_stride\_1} [float, default: 0.01] The uniform stride of the first table
- \texttt{table\_stride\_2} [float, default: 0.1] The uniform stride of the second table
- \texttt{check\_frequency} [int, default: -1] The overflow check frequency
- \texttt{suffix} [str, optional] The suffix of the scope

Notes

This method is called by others when the descriptor supported compression.

\texttt{enable\_mixed\_precision} (mixed\_prec: Optional[dict] = None) → None

Reveive the mixed precision setting.

Parameters

- \texttt{mixed\_prec} The mixed precision setting used in the embedding net

Notes

This method is called by others when the descriptor supported compression.

\texttt{property explicit\_ntypes: bool}

Explicit ntypes with type embedding.

\texttt{classmethod get\_class\_by\_input} (input: dict)

Chapter 18. Python API
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.

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

get_nlist() → Tuple[Tensor, 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 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_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.

```python
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

```python
init_variables(graph: Graph, graph_def: 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.

```python
pass_tensors_from_frz_model(*tensors: 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()`.

```python
abstract prod_force_virial(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, 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

static register(key: str) → Callable
   Register a descriptor plugin.

Parameters

key
   [str] the key of a descriptor

Returns

Descriptor
   the registered descriptor

Examples

```python
>>> @Descriptor.register("some_descrpt")
    class SomeDescript(Descriptor):
        pass
```

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

deepmd.descriptor.hybrid module

class deepmd.descriptor.hybrid.DescrptHybrid(*args, **kwargs)
   Bases: Descriptor

   Concatenate a list of descriptors to form a new descriptor.

Parameters

   list
      [list] Build a descriptor from the concatenation of the list of descriptors.

Attributes
**explicit_ntypes**
Explicit ntypes with type embedding.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>build</strong>(coord_, atype_, natoms, box_, mesh, ...)</td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><strong>build_type_exclude_mask</strong>(exclude_types, ...)</td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><strong>compute_input_stats</strong>(data_coord, data_box, ...)</td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><strong>enable_compression</strong>(min_nbor_dist, graph, ...)</td>
<td>Reieve the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><strong>enable_mixed_precision</strong>([mixed_prec])</td>
<td>Reieve the mixed precision setting.</td>
</tr>
<tr>
<td><strong>get_dim_out</strong>()</td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><strong>get_dim_rot_mat_1</strong>()</td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><strong>get_nlist</strong>()</td>
<td>Get the neighbor information of the descriptor, returns the nlist of the descriptor with the largest cut-off radius.</td>
</tr>
<tr>
<td><strong>get_nlist_i</strong>(ii)</td>
<td>Get the neighbor information of the ii-th descriptor.</td>
</tr>
<tr>
<td><strong>get_ntypes</strong>()</td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><strong>get_rcut</strong>()</td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><strong>get_tensor_names</strong>([suffix])</td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><strong>init_variables</strong>(graph, graph_def[, suffix])</td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><strong>merge_input_stats</strong>(stat_dict)</td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td><strong>pass_tensors_from_frz_model</strong>(*tensors)</td>
<td>Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz_graph_def.</td>
</tr>
<tr>
<td><strong>prod_force_virial</strong>(atom_ener, natoms)</td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><strong>register</strong>(key)</td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><strong>update_sel</strong>(global_jdata, local_jdata)</td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

---


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_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: Optional[list] = None, **kwargs) → None

Compute the statistics (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.model.make_stat_input

data_box
  The box. Can be generated by deepmd.model.make_stat_input

data_atype
  The atom types. Can be generated by deepmd.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.model.make_stat_input

mesh
  The mesh for neighbor searching. Can be generated by deepmd.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_attn.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: Graph, graph_def: 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 statistics (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: Optional[dict] = None) → None

Reveive the mixed precision setting.

Parameters

- mixed_prec
  - The mixed precision setting used in the embedding net

property explicit_ntypes: bool

Explicit ntypes with type embedding.

get_dim_out() → int

Returns the output dimension of this descriptor.

get_nlist() → Tuple[Tensor, 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[Tensor, 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

**get_ntypes()** → int

Returns the number of atom types.

**get_rcut()** → float

Returns the cut-off radius.

**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

**init_variables**(graph: Graph, graph_def: 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

**merge_input_stats**(stat_dict)

Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict
The dict of statistics computed from compute_input_stats, including:
sumr
   The sum of radial statistics.
suma
   The sum of relative coord statistics.
sumn
   The sum of neighbor numbers.
sumr2
   The sum of square of radial statistics.
suma2
   The sum of square of relative coord statistics.

\texttt{pass\_tensors\_from\_frz\_model} (*tensors: Tensor) \rightarrow \text{None}

Pass the descript\_reshape tensor as well as descript\_deriv tensor from the frz graph_def.

Parameters
*tensors
   \text{[tf.Tensor]} passed tensors

\texttt{prod\_force\_virial} (atom\_ener: Tensor, natoms: Tensor) \rightarrow \text{Tuple[Tensor, Tensor, 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[1]:
   2 \leq i < Ntypes+2, number of type i atoms

Returns
force
   The force on atoms
virial
   The total virial
atom\_virial
   The atomic virial

\texttt{classmethod update\_sel} (global\_jdata: \text{dict}, local\_jdata: \text{dict})

Update the selection and perform neighbor statistics.

Parameters
global\_jdata
   \text{[dict]} The global data, containing the training section
local\_jdata
   \text{[dict]} The local data refer to the current class
**deepmd.descriptor.loc_frame module**

**class** `deepmd.descriptor.loc_frame.DescrptLocFrame(*args, **kwargs)`

**Bases:** `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[str]]` 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[str]]` 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.

**Attributes**

- `explicit_ntypes`
  Explicit ntypes with type embedding.
### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box_, mesh, ...)</code></td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><code>build_type_exclude_mask(exclude_types, ...)</code></td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><code>compute_input_stats(data_coord, data_box, ...)</code></td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><code>enable_compression(min_nbor_dist, graph, ...)</code></td>
<td>Retrieve the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><code>enable_mixed_precision([mixed_prec])</code></td>
<td>Retrieve the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_dim_out()</code></td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><code>get_dim_rot_mat_1()</code></td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><code>get_nlist()</code></td>
<td>Returns.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><code>get_rot_mat()</code></td>
<td>Get rotational matrix.</td>
</tr>
<tr>
<td><code>get_tensor_names([suffix])</code></td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, suffix])</code></td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><code>pass_tensors_from_frz_model(*tensors)</code></td>
<td>Pass the descpt_reshape tensor as well as descpt_deriv tensor from the frz graph_def.</td>
</tr>
<tr>
<td><code>prod_force_virial(atom_ener, natoms)</code></td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>


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_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None
Compute the statistics (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.model.make_stat_input
data_box
        The box. Can be generated by deepmd.model.make_stat_input
data_atype
        The atom types. Can be generated by deepmd.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.model.make_stat_input
mesh
        The mesh for neighbor searching. Can be generated by deepmd.model.make_stat_input
input_dict
        Dictionary for additional input
**kwargs
        Additional keyword arguments.

get_dim_out() → int
Returns the output dimension of this descriptor.

get_nlist() → Tuple[Tensor, 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
get_ntypes() → int
Returns the number of atom types.

get_rcut() → float
Returns the cut-off radius.

get_rot_mat() → Tensor
Get rotational matrix.

init_variables(graph: Graph, graph_def: 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

prod_force_virial(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, 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

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
the deepmd.descriptor.se module

class deepmd.descriptor.se.DescrptSe(*args, **kwargs)

    Bases: Descriptor

    A base class for smooth version of descriptors.

    Notes

    All of these descriptors have an environmental matrix and an embedding network (`deepmd.utils.
    network.embedding_net()`), so they can share some similar 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

    Methods

    build(coord_, atype_, natoms, box_, mesh,...)
        Build the computational graph for the descriptor.
    build_type_exclude_mask(exclude_types,...)
        Build the type exclude mask for the descriptor.
    compute_input_stats(data_coord, data_box, ...)
        Compute the statistics (avg and std) of the training data.
    enable_compression(min_nbor_dist, graph,...)
        Reieve the statistics (distance, max_nbor_size and env_mat_range) of the training data.
    enable_mixed_precision([mixed_prec])
        Reieve the mixed precision setting.
    get_dim_out()
        Returns the output dimension of this descriptor.
    get_dim_rot_mat_1()
        Returns the first dimension of the rotation matrix.
    get_nlist()
        Returns neighbor information.
    get_ntypes()
        Returns the number of atom types.
    get_rcut()
        Returns the cut-off radius.
    get_tensor_names([suffix])
        Get names of tensors.
    init_variables(graph, graph_def[, suffix])
        Init the embedding net variables with the given dict.
    pass_tensors_from_frz_model(descrpt_reshape, ...)
        Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.
    prod_force_virial(atom_ener, natoms)
        Compute force and virial.
    register(key)
        Register a descriptor plugin.
    update_sel(global_jdata, local_jdata)
        Update the selection and perform neighbor statistics.
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

init_variables(graph: Graph, graph_def: 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

pass_tensors_from_frz_model(descrpt_reshape: Tensor, descrpt_deriv: Tensor, rij: Tensor, nlist: 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

property precision: DType
Precision of filter network.

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.descriptor.se_a.DescrptSeA(*args, **kwargs)

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor $D^i \in \mathbb{R}^{M_1 \times M_2}$ is given by

$$D^i = (G^i)^T \mathcal{R}^i (r^i)^T G^i_{<}$$

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

$$\begin{pmatrix} s(r^i)_{xji} \\ s(r^i)_{yji} \\ s(r^i)_{zji} \\ r^i_{ji} \end{pmatrix}$$

where $R^i_{ji} = R_j - R_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r^i_{ji} = ||R^i_{ji}||$ is its norm. The switching function $s(r)$ is defined as:

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

Each row of the embedding matrix $G^i \in \mathbb{R}^{N \times M_1}$ consists of outputs of an embedding network $N$ of $s(r_{ji})$:

$$(G^i)^j = N(s(r_{ji}))$$

$G^i_{<} \in \mathbb{R}^{N \times M_2}$ takes first $M_2$ columns of $G^i$. The equation of embedding network $N$ can be found at `deepmd.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[str]] sel[i] specifies the maximum number of type i atoms in the cut-off radius
- `neuron`: [list[int]] Number of neurons in each hidden layers of the embedding net $N$
- `axis_neuron`: Number of the axis neuron $M_2$ (number of columns of the sub-matrix of the embedding matrix)$\text{resnet}_dt$:
  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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.

precision
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

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.

References

[1]

Attributes

explicit_ntypes
Explicit ntypes with type embedding.

precision
Precision of filter network.
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>build</strong> <em>(coord_, atype_, natoms, box_, mesh, ...)</em></td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><strong>build_type_exclude_mask</strong> <em>(exclude_types, ...)</em></td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><strong>compute_input_stats</strong> <em>(data Coord, data box, ...)</em></td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><strong>enable_compression</strong> <em>(min_nbor_dist, graph, ...)</em></td>
<td>Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><strong>enable_mixed_precision</strong> <em>(mixed_prec)</em></td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td><strong>get_dim_out</strong> <em>(</em>)</td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><strong>get_dim_rot_mat_i</strong> <em>(</em>)</td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><strong>get_nlist</strong> <em>(</em>)</td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td><strong>get_ntypes</strong> <em>(</em>)</td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><strong>get_rcut</strong> <em>(</em>)</td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><strong>get_rot_mat</strong> <em>(</em>)</td>
<td>Get rotational matrix.</td>
</tr>
<tr>
<td><strong>get_tensor_names</strong> <em>(suffix)</em></td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><strong>init_variables</strong> <em>(graph, graph_def[, suffix])</em></td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><strong>merge_input_stats</strong> <em>(stat_dict)</em></td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td><strong>pass_tensors_from_frz_model</strong> <em>(descrpt_reshape, ...)</em></td>
<td>Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.</td>
</tr>
<tr>
<td><strong>prod_force_virial</strong> <em>(atom_ener, natoms)</em></td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><strong>register</strong> <em>(key)</em></td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><strong>update_sel</strong> <em>(global_jdata, local_jdata)</em></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

### get_class_by_input

```python
build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → 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,
DeePMD-kit

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_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None

Compute the statistics (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.model.make_stat_input

data_box
The box. Can be generated by deepmd.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.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.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.model.make_stat_input

input_dict
Dictionary for additional input

**kwargs
Additional keyword arguments.

enable_compression(min_nbor_dist: float, graph: Graph, graph_def: GraphDef, table_extrapolate: float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None

Receive the statistics (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
DeePMD-kit

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: Optional[dict] = None) → None
Reveive the mixed precision setting.

Parameters
mixed_prec
The mixed precision setting used in the embedding net

property explicit_ntypes: bool
Explicit ntypes with type embedding.

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[Tensor, 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

get_ntypes() → int
Returns the number of atom types.

get_rcut() → float
Returns the cut-off radius.

get_rot_mat() → Tensor
Get rotational matrix.

init_variables(graph: Graph, graph_def: GraphDef, suffix: str = '') → None
Init the embedding net variables with the given dict.

Parameters
DeePMD-kit

```python
def merge_input_stats(stat_dict):
    merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.

    Parameters:
    stat_dict:
        The dict of statistics computed from compute_input_stats, including:

        sumr:
            The sum of radial statistics.

        suma:
            The sum of relative coord statistics.

        sumn:
            The sum of neighbor numbers.

        sumr2:
            The sum of square of radial statistics.

        suma2:
            The sum of square of relative coord statistics.

    @prod_force_virial(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, 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.
```

Chapter 18. Python API
deepmd.descriptor.se_a_ebd module

class deepmd.descriptor.se_a_ebd.DescrpSeAEbd(*args, **kwargs)
   Bases: DescrpSeA

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[str]] sel[i] specifies the maximum 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.

Attributes
explicit_ntypes
Explicit ntypes with type embedding.

precision
Precision of filter network.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td>build_type_exclude_mask</td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td>compute_input_stats</td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td>enable_compression</td>
<td>Reiveve the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td>enable_mixed_precision</td>
<td>Reiveve the mixed precision setting.</td>
</tr>
<tr>
<td>get_dim_out</td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td>get_dim_rot_mat_1</td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td>get_nlist</td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td>get_ntypes</td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td>get_rcut</td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td>get_rot_mat</td>
<td>Get rotational matrix.</td>
</tr>
<tr>
<td>get_tensor_names</td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td>init_variables</td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td>merge_input_stats</td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td>pass_tensors_from_frz_model</td>
<td>Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.</td>
</tr>
<tr>
<td>prod_force_virial</td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td>register</td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td>update_sel</td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

### build

```python
build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: dict, reuse: Optional[bool] = None, suffix: str = "") → 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

**deepmd.descriptor.se_a_ebd_v2 module**

class deepmd.descriptor.se_a_ebd_v2.DescrptSeAEbdV2(*args, **kwargs)

Bases: DescrptSeA

A compressible se_a_ebd model.

This model is a warpper for DescriptorSeA, which set stripped_type_embedding=True.

Attributes

explicit_ntypes  
Explicit ntypes with type embedding.

precision  
Precision of filter network.
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box_, mesh, ...)</code></td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><code>build_type_exclude_mask(exclude_types, ...)  </code></td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><code>compute_input_stats(data_coord, data_box, ...)</code></td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><code>enable_compression(min_nbor_dist, graph, ...)</code></td>
<td>Retrieve the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><code>enable_mixed_precision([mixed_prec])</code></td>
<td>Retrieve the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_dim_out()</code></td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><code>get_dim_rot_mat_1()</code></td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><code>get_nlist()</code></td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><code>get_rot_mat()</code></td>
<td>Get rotational matrix.</td>
</tr>
<tr>
<td><code>get_tensor_names([suffix])</code></td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, suffix])</code></td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><code>merge_input_stats(stat_dict)</code></td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td><code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code></td>
<td>Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.</td>
</tr>
<tr>
<td><code>prod_force_virial(atom_ener, natoms)</code></td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

### deepmd.descriptor.se_a_ef module

```python
class deepmd.descriptor.se_a_ef.DescrptSeAEf(*args, **kwargs)
```

Bases: `DescrptSe`

Smooth edition descriptor with Ef.

**Parameters**

- `rcut`  
  The cut-off radius

- `rcut_smth`  
  From where the environment matrix should be smoothed

- `sel`  
  `[list[str]]` sel[i] specifies the maximum 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: \( y = x + dt \cdot \phi(Wx + b) \)

trainable
If the weights of embedding net are trainable.

seed
Random seed for initializing the network parameters.

type_one_side
Try to build \( N_{\text{types}} \) embedding nets. Otherwise, building \( N_{\text{types}}^2 \) embedding nets

exclude_types
\[[\text{List}[\text{List}[\text{int}]]]\] 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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.

precision
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

uniform_seed
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

Attributes

**explicit_ntypes**
Explicit ntypes with type embedding.

**precision**
Precision of filter network.
## Methods

**build**

```python
def build(coord_, atype_, natoms, box_, mesh, ...) -> Tensor
```

Build the computational graph for the descriptor.

**build_type_exclude_mask**

```python
def build_type_exclude_mask(exclude_types, ...) -> Tensor
```

Build the type exclude mask for the descriptor.

**compute_input_stats**

```python
def compute_input_stats(data_coord, data_box, ...) -> Tensor
```

Compute the statistics (avg and std) of the training data.

**enable_compression**

```python
def enable_compression(min_nbor_dist, graph, ...) -> Tensor
```

Retrieve the statistics (distance, max_nbor_size and env_mat_range) of the training data.

**enable_mixed_precision**

```python
def enable_mixed_precision(mixed_prec) -> Tensor
```

Retrieve the mixed precision setting.

**get_dim_out**

```python
def get_dim_out() -> Tensor
```

Returns the output dimension of this descriptor.

**get_dim_rot_mat_1**

```python
def get_dim_rot_mat_1() -> Tensor
```

Returns the first dimension of the rotation matrix.

**get_nlist**

```python
def get_nlist() -> Tensor
```

Returns neighbor information.

**get_n types**

```python
def get_n types() -> Tensor
```

Returns the number of atom types.

**get_rcut**

```python
def get_rcut() -> Tensor
```

Returns the cut-off radius.

**get_rot_mat**

```python
def get_rot_mat() -> Tensor
```

Get rotational matrix.

**get_tensor_names**

```python
def get_tensor_names(suffix) -> Tensor
```

Get names of tensors.

**init_variables**

```python
def init_variables(graph, graph_def, suffix) -> Tensor
```

Init the embedding net variables with the given dict.

**pass_tensors_from_frz_model**

```python
def pass_tensors_from_frz_model(descrpt_reshape, ...) -> Tensor
```

Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.

**prod_force_virial**

```python
def prod_force_virial(atom_ener, natoms) -> Tensor
```

Compute force and virial.

**register**

```python
def register(key) -> Tensor
```

Register a descriptor plugin.

**update_sel**

```python
def update_sel(global_jdata, local_jdata) -> Tensor
```

Update the selection and perform neighbor statistics.

---

**build**

```python
def build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: dict, reuse: Optional[bool] = None, suffix: str = '') -> 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[1]: 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’.
DeePMD-kit

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_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None

Compute the statistics (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.model.make_stat_input

data_box
The box. Can be generated by deepmd.model.make_stat_input

data_atype
The atom types. Can be generated by deepmd.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.model.make_stat_input

mesh
The mesh for neighbor searching. Can be generated by deepmd.model.make_stat_input

input_dict
Dictionary for additional input

**kwargs
Additional keyword arguments.

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[Tensor, 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
DeePMD-kit

```python
get_notypes() → int
    Returns the number of atom types.

get_rcut() → float
    Returns the cut-off radius.

get_rot_mat() → Tensor
    Get rotational matrix.

prod_force_virial(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, 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.descriptor.se_a_ef.DescrptSeEfLower(*args, **kwargs)
Bases: DescrptSeA
    Helper class for implementing DescrptSeEf.

    Attributes

    explicit_ntypes
        Explicit ntypes with type embedding.

    precision
        Precision of filter network.
```
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box_, mesh, ...)</code></td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><code>build_type_exclude_mask(exclude_types, ...)</code></td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><code>compute_input_stats(data_coord, data_box, ...)</code></td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><code>enable_compression(min_nbor_dist, graph, ...)</code></td>
<td>Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><code>enable_mixed_precision([mixed_prec])</code></td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_dim_out()</code></td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><code>get_dim_rot_mat_1()</code></td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><code>get_nlist()</code></td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><code>get_rot_mat()</code></td>
<td>Get rotational matrix.</td>
</tr>
<tr>
<td><code>get_tensor_names([suffix])</code></td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, suffix])</code></td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><code>merge_input_stats(stat_dict)</code></td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td><code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code></td>
<td>Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz_graph_def.</td>
</tr>
<tr>
<td><code>prod_force_virial(atom_ener, natoms)</code></td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

### get_class_by_input

```python
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_input_stats(data_coord, data_box, data_atype, natoms_vec, mesh, input_dict, **kwargs)
Compute the statistics (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.model.make_stat_input
data_box
    The box. Can be generated by deepmd.model.make_stat_input
data_atype
    The atom types. Can be generated by deepmd.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.model.make_stat_input
mesh
    The mesh for neighbor searching. Can be generated by
depemd.model.make_stat_input
input_dict
    Dictionary for additional input
**kwargs
    Additional keyword arguments.

depemd.descriptor.se_a_mask module

class deepemd.descriptor.se_a_mask.DescrptSeAMask(*args, **kwargs)
Bases: 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 $\mathcal{D}^i \in \mathbb{R}^{M_1 \times M_2}$ is given by \[1\]

$$
\mathcal{D}^i = (G^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}_<^i
$$

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

$$
(\mathcal{R}^i)_j = \begin{bmatrix}
\frac{s(r_{ji})}{s(r_{ji})x_{ji}} \\
\frac{r_{ji}}{s(r_{ji})y_{ji}} \\
\frac{r_{ji}}{s(r_{ji})z_{ji}} \\
\frac{r_{ji}}{r_{ji}}
\end{bmatrix}
$$
where \( R_{ji} = R_j - R_i = (x_{ji}, y_{ji}, z_{ji}) \) is the relative coordinate and \( r_{ji} = \| R_{ji} \| \) is its norm. The switching function \( s(r) \) is defined as:

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

Each row of the embedding matrix \( G^i \in \mathbb{R}^{N \times M_1} \) consists of outputs of a embedding network \( \mathcal{N} \) of \( s(r_{ji}) \):

\[
(G^i)_j = \mathcal{N}(s(r_{ji}))
\]

\( G^i_\prec \in \mathbb{R}^{N \times M_2} \) takes first \( M_2 \) columns of \( G^i \). The equation of embedding network \( \mathcal{N} \) can be found at `deepmd.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[str]] `sel[i]` specifies the maximum 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 \( N \) \( N \) 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]

**Attributes**

- `explicit_ntypes`
  - Explicit ntypes with type embedding.

- `precision`
  - Precision of filter network.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box_, mesh, ...)</code></td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><code>build_type_exclude_mask(exclude_types, ...)</code></td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><code>compute_input_stats(data_coord, data_box, ...)</code></td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><code>enable_compression(min_nbor_dist, graph, ...)</code></td>
<td>Re reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><code>enable_mixed_precision([mixed_prec])</code></td>
<td>Re reveive the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_dim_out()</code></td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><code>get_dim_rot_mat_1()</code></td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><code>get_nlist()</code></td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Returns the cutoff radius.</td>
</tr>
<tr>
<td><code>get_rot_mat()</code></td>
<td>Get rotational matrix.</td>
</tr>
<tr>
<td><code>get_tensor_names([suffix])</code></td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, suffix])</code></td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><code>merge_input_stats(stat_dict)</code></td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td><code>pass_tensors_from_frz_model(descript_reshape, ...)</code></td>
<td>Pass the descript_reshape tensor as well as descript_deriv tensor from the frz_graph_def.</td>
</tr>
<tr>
<td><code>prod_force_virial(atom_ener, natoms)</code></td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```python
build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: Dict[str, Any], reuse: Optional[bool] = None, suffix: str = '') → 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[1]: 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_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict, **kwargs) → None
    Compute the statistics (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.model.make_stat_input

data_box
    The box. Can be generated by deepmd.model.make_stat_input

data_atype
    The atom types. Can be generated by deepmd.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.model.make_stat_input

mesh
    The mesh for neighbor searching. Can be generated by deepmd.model.make_stat_input

input_dict
    Dictionary for additional input

**kwargs
    Additional keyword arguments.

get_rcut() → float
    Returns the cutoff radius.
prod_force_virial (atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, 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

depdm.descriptor.se_atten module

class deepmd.descriptor.se_atten.DescrptSeAtten (*args, **kwargs)
Bases: DescrptSeA
Smooth version descriptor with attention.

Parameters

rcut
The cut-off radius \( r_c \)

rcut_smth
From where the environment matrix should be smoothed \( r_s \)

sel
[list[str]] sel[i] specifies the maximum 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: \( y = x + dt \cdot\phi(Wx + b) \)
trainable
   If the weights of embedding net are trainable.

seed
   Random seed for initializing the network parameters.

type_one_side
   Try to build $N_{\text{types}}$ embedding nets. Otherwise, building $N_{\text{types}}^2$ embedding nets

exclude_types
   \[[\text{List}[\text{List}[\text{int}]]]\] 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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.

precision
   The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

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.

stripped_type_embedding
   Whether to strip the type embedding into a separated embedding network. Default value will be True in se_attn_v2 descriptor.

smooth_type_embedding
   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_attn_v2 descriptor.

Raises

ValueError
   if ntypes is 0.

Attributes

explicit_ntypes
   Explicit ntypes with type embedding.
**precision**

Precision of filter network.

**Methods**

```python
build(coord_, atype_, natoms, box_, mesh, ...)  # Build the computational graph for the descriptor.
build_type_exclude_mask(exclude_types, ...)  # Build the type exclude mask for the attention descriptor.
compute_input_stats(data_coord, data_box, ...)  # Compute the statistics (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)  # Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec])  # Receive the mixed precision setting.
get_dim_out()  # Returns the output dimension of this descriptor.
get_dim_rot_mat_1()  # Returns the first dimension of the rotation matrix.
get_nlist()  # Returns neighbor information.
get_ntypes()  # Returns the number of atom types.
get_rcut()  # Returns the cut-off radius.
get_rot_mat()  # Get rotational matrix.
get_tensor_names([suffix])  # Get names of tensors.
init_variables(graph, graph_def[, suffix])  # Init the embedding net variables with the given dict.
merge_input_stats(stat_dict)  # Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
pass_tensors_from_frz_model(descrpt_reshape, ...)  # Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz_graph_def.
prod_force_virial(atom_ener, natoms)  # Compute force and virial.
register(key)  # Register a descriptor plugin.
update_sel(global_jdata, local_jdata)  # Update the selection and perform neighbor statistics.
```

**build**

```python
build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → 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

build_type_exclude_mask(exclude_types: List[Tuple[int, int]], ntypes: int, sel: List[int], ndescript: int, atype: Tensor, shape0: Tensor, nei_type_vec: Tensor) \(\rightarrow\) 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.

    ndescript
        [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, ndescript), 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.descriptor.descriptor.Descriptor.build_type_exclude_mask
Notes

This method has the similar way to build the type exclude mask as \texttt{deepmd.descriptor.Descriptor.build_type_exclude_mask()}. The mathematical 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.

\texttt{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: Optional[list] = None, **kwargs) \rightarrow None}

Compute the statistics (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

\begin{itemize}
\item \texttt{data Coord}  
The coordinates. Can be generated by \texttt{deepmd.model.make_stat_input}
\item \texttt{data Box}  
The box. Can be generated by \texttt{deepmd.model.make_stat_input}
\item \texttt{data Atype}  
The atom types. Can be generated by \texttt{deepmd.model.make_stat_input}
\item \texttt{natomvecs}  
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 \texttt{real_natoms_vec}.
\item \texttt{mesh}  
The mesh for neighbor searching. Can be generated by \texttt{deepmd.model.make_stat_input}
\item \texttt{input Dict}  
Dictionary for additional input
\item \texttt{mixed_type}  
Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see doc/model/train_se_attn.md), in which frames in a system may have different natoms Vecs, with the same nloc.
\item \texttt{real_natoms Vec}  
If mixed\_type is True, it takes in the real natoms Vec for each frame.
\end{itemize}

\texttt{**kwargs}

Additional keyword arguments.

\texttt{enable_compression(min_nbor_dist: float, graph: Graph, graph def: GraphDef, table extrapolate: float = 5, table_stride 1: float = 0.01, table stride 2: float = 0.1, check frequency: int = -1, suffix: str = ‘’) \rightarrow None}

Receieve the statistics (distance, max\_nbor\_size and env\_mat\_range) of the training data.

Parameters

\begin{itemize}
\item \texttt{min_nbor_dist}  
The nearest distance between atoms
\item \texttt{graph}  
[tf.Graph] The graph of the model
\item \texttt{graph def}  
[tf.GraphDef] The graph\_def of the model
\end{itemize}
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

table_stride_2
   The uniform stride of the second table

table_stride_2
   The uniform stride of the second 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

property explicit_ntypes: bool
   Explicit ntypes with type embedding.

init_variables(graph: Graph, graph_def: GraphDef, suffix: str = '') → None
   Init the embedding net variables with the given dict.

   Parameters
   
   graph
      [tf.Graph] The input frozen model graph

t_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.descriptor.se_atten_v2 module

class deepmd.descriptor.se_atten_v2.DescrptSeAttenV2(*args, **kwargs)
   Bases: 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
   [list[str]] sel[i] specifies the maximum number of type \( i \) atoms in the cut-off radius
neuron
   [list[int]] Number of neurons in each hidden layers of the embedding net N
axis_neuron
   Number of the axis neuron M (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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.
precision
   The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.
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.
Attributes
explicit_ntypes
   Explicit ntypes with type embedding.
precision
   Precision of filter network.
**Methods**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build(coord_, atype_, natoms, box_, mesh, ...)</td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td>build_type_exclude_mask(exclude_types, ...)</td>
<td>Build the type exclude mask for the attention descriptor.</td>
</tr>
<tr>
<td>compute_input_stats(data_coord, data_box, ...)</td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td>enable_compression(min_nbor_dist, graph, ...)</td>
<td>Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td>enable_mixed_precision([mixed_prec])</td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td>get_dim_out()</td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td>get_dim_rot_mat_1()</td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td>get_nlist()</td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td>get_ntypes()</td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td>get_rcut()</td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td>get_rot_mat()</td>
<td>Get rotational matrix.</td>
</tr>
<tr>
<td>get_tensor_names([suffix])</td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td>init_variables(graph, graph_def[, suffix])</td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td>merge_input_stats(stat_dict)</td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td>pass_tensors_from_frz_model(descrpt_reshape, ...)</td>
<td>Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz_graph_def.</td>
</tr>
<tr>
<td>prod_force_virial(atom_ener, natoms)</td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td>register(key)</td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td>update_sel(global_jdata, local_jdata)</td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```python
get_class_by_input
```

**deepmd.descriptor.se_r module**

```python
class deepmd.descriptor.se_r.DescrptSeR(*args, **kwargs)
```

Bases: `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[str]]` sel[i] specifies the maximum number of type i atoms in the cut-off radius
Neuron

\[[\text{List}[\text{int}]]\] Number of neurons in each hidden layers of the embedding net

Resnet Dt

Time-step dt in the resnet construction: \( y = x + dt \cdot \phi(Wx + b) \)

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 \( N^2 \) embedding nets

Exclude Types

\[[\text{List}[\text{List}[\text{int}]]]\] 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 "relu", "relu6", "softplus", "sigmoid", "tanh", "gelu", "gelu_tf", "None", "none".

Precision

The precision of the embedding net parameters. Supported options are "default", "float16", "float32", "float64", "bfloat16".

Uniform Seed

Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

Attributes

**explicit_ntypes**

Explicit ntypes with type embedding.

**precision**

Precision of filter network.
### Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box_, mesh, ...)</code></td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><code>build_type_exclude_mask(exclude_types, ...)</code></td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><code>compute_input_stats(data_coord, data_box, ...)</code></td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><code>enable_compression(min_nbor_dist, graph, ...)</code></td>
<td>Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_dim_out()</code></td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><code>get_dim_rot_mat_1()</code></td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><code>get_nlist()</code></td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><code>get_tensor_names([suffix])</code></td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, suffix])</code></td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><code>merge_input_stats(stat_dict)</code></td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td><code>pass_tensors_from_frz_model(descript_reshape, ...)</code></td>
<td>Pass the descript_reshape tensor as well as descript_deriv tensor from the frz graph_def.</td>
</tr>
<tr>
<td><code>prod_force_virial(atom_ener, natoms)</code></td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```python
build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → 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 sufffix to identify this descriptor

Returns

**descriptor**
    The output descriptor

**compute_input_stats**(data_coord, data_box, data_atype, natoms_vec, mesh, input_dict, **kwargs)
Compute the statistics (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.model.make_stat_input
- **data_box**
  The box. Can be generated by deepmd.model.make_stat_input
- **data_atype**
  The atom types. Can be generated by deepmd.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.model.make_stat_input
- **mesh**
  The mesh for neighbor searching. Can be generated by deepmd.model.make_stat_input
- **input_dict**
  Dictionary for additional input
- **kwargs**
  Additional keyword arguments.

**enable_compression**(min_nbor_dist: float, graph: Graph, graph_def: 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 statistics (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

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

get_ntypes()
  Returns the number of atom types.

get_rcut()
  Returns the cut-off radius.

merge_input_stats(stat_dict)
  Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.

    Parameters

    stat_dict
      The dict of statistics computed from compute_input_stats, including:

      sumr
        The sum of radial statistics.

      sumn
        The sum of neighbor numbers.

      sumr2
        The sum of square of radial statistics.

prod_force_virial(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, Tensor]
  Compute force and virial.

    Parameters

    atom_ener
      The atomic energy
DeePMD-kit

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[1]: 2 <= i < Ntypes+2, number of type i atoms

Returns

force
The force on atoms

virial
The total virial

atom_virial
The atomic virial

depdmdescriptor.se_t module

class deepmd.descriptor.se_t.DescrptSeT(*args, **kwargs)
Bases: 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[str]] sel[i] specifies the maximum 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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tff”, “None”, “none”.

precision
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

uniform_seed
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed
Attributes

**explicit_ntypes**
Explicit ntypes with type embedding.

**precision**
Precision of filter network.

Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>build</strong>(coord_, atype_, natoms, box_, mesh, ...)</td>
<td>Build the computational graph for the descriptor.</td>
</tr>
<tr>
<td><strong>build_type_exclude_mask</strong>(exclude_types, ...)</td>
<td>Build the type exclude mask for the descriptor.</td>
</tr>
<tr>
<td><strong>compute_input_stats</strong>(data_coord, data_box, ...)</td>
<td>Compute the statistics (avg and std) of the training data.</td>
</tr>
<tr>
<td><strong>enable_compression</strong>(min_nbor_dist, graph, ...)</td>
<td>Retrieve the statistics (distance, max_nbor_size and env_mat_range) of the training data.</td>
</tr>
<tr>
<td><strong>enable_mixed_precision</strong>([mixed_prec])</td>
<td>Retrieve the mixed precision setting.</td>
</tr>
<tr>
<td><strong>get_dim_out</strong>()</td>
<td>Returns the output dimension of this descriptor.</td>
</tr>
<tr>
<td><strong>get_dim_rot_mat_1</strong>()</td>
<td>Returns the first dimension of the rotation matrix.</td>
</tr>
<tr>
<td><strong>get_dim_rot_mat_2</strong>()</td>
<td>Returns the second dimension of the rotation matrix.</td>
</tr>
<tr>
<td><strong>get_dim_rotation_matrix</strong>()</td>
<td>Returns the rotation matrix.</td>
</tr>
<tr>
<td><strong>get_nlist</strong>()</td>
<td>Returns neighbor information.</td>
</tr>
<tr>
<td><strong>get_ntypes</strong>()</td>
<td>Returns the number of atom types.</td>
</tr>
<tr>
<td><strong>get_rcut</strong>()</td>
<td>Returns the cut-off radius.</td>
</tr>
<tr>
<td><strong>get_tensor_names</strong>(suffix)</td>
<td>Get names of tensors.</td>
</tr>
<tr>
<td><strong>init_variables</strong>(graph, graph_def[, suffix])</td>
<td>Init the embedding net variables with the given dict.</td>
</tr>
<tr>
<td><strong>merge_input_stats</strong>(stat_dict)</td>
<td>Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.</td>
</tr>
<tr>
<td><strong>pass_tensors_from_frz_model</strong>(descrpt_reshape, ...)</td>
<td>Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz_graph_def.</td>
</tr>
<tr>
<td><strong>prod_force_virial</strong>(atom_ener, natoms)</td>
<td>Compute force and virial.</td>
</tr>
<tr>
<td><strong>register</strong>(key)</td>
<td>Register a descriptor plugin.</td>
</tr>
<tr>
<td><strong>update_sel</strong>(global_jdata, local_jdata)</td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

---

**get_class_by_input**

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input_dict: dict, reuse: Optional[bool] = None, suffix: str = ")") \rightarrow 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

compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list,
  input_dict: dict, **kwargs) → None

Compute the statistics (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.model.make_stat_input

data_box
  The box. Can be generated by deepmd.model.make_stat_input

data_atype
  The atom types. Can be generated by deepmd.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.model.make_stat_input

mesh
  The mesh for neighbor searching. Can be generated by
  deepmd.model.make_stat_input

input_dict
  Dictionary for additional input

**kwargs
  Additional keyword arguments.

enable_compression(min_nbor_dist: float, graph: Graph, graph_def: GraphDef, table_extrapolate:
  float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1,
  check_frequency: int = -1, suffix: str = '') → None

Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
  The nearest distance between atoms
DeePMD-kit

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

get_dim_out() → int
    Returns the output dimension of this descriptor.

get_nlist() → Tuple[Tensor, 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

get_ntypes() → int
    Returns the number of atom types.

get_rcut() → float
    Returns the cut-off radius.

merge_input_stats(stat_dict)
    Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.

    Parameters
        stat_dict
            The dict of statistics computed from compute_input_stats, including:

            sumr
                The sum of radial statistics.

            suma
                The sum of relative coord statistics.
sumn
The sum of neighbor numbers.

sumr2
The sum of square of radial statsitics.

suma2
The sum of square of relative coord statsitics.

**prod_force_virial**(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, 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[1]: 2 <= i < Ntypes+2, number of type i atoms

**Returns**

- **force**
  The force on atoms

- **virial**
  The total virial

- **atom_virial**
  The atomic virial

**deepmd.entrypoints package**

Submodule that contains all the DeePMD-Kit entry point scripts.


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] training 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 specified log will be written to this file

log_level
  [int] logging level

**kwargs
  additional arguments

deepmd.entrypoints.convert(*, FROM: str, input_model: str, output_model: str, **kwargs)

deepmd.entrypoints.doc_train_input(*, out_type: str = 'rst', **kwargs)

  Print out training input arguments to console.

deepmd.entrypoints.freeze(*, checkpoint_folder: str, output: str, node_names: Optional[str] = None,
  nvnmd_weight: Optional[str] = None, united_model: bool = False, **kwargs)

  Freeze the graph in supplied folder.

Parameters

  checkpoint_folder
    [str] location of the folder with model

  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


  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
set_prefix
  [str] The set prefix of the system
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 parameter 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_entrypoints.neighbor_stat(*, system: str, rcut: float, type_map: List[str], one_type: bool = False, **kwargs)

Calculate neighbor statistics.

Parameters

system
  [str] system to stat
rcut
  [float] cutoff radius
type_map
  [list[str]] type map
one_type
  [bool, optional, default=False] treat all types as a single type
**kwargs
  additional arguments
**Examples**

```python
>>> neighbor_stat(system='.*', rcut=6., type_map=['C', 'H', 'O', 'N', 'P', 'S', 'Na', 'H2O', 'mNa', 'mCl', 'mC', 'mH', 'mMg', 'mN', 'mO', 'mP', '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]
```

```python
depdep.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

```python
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
- **set_prefix**
  - [str] string prefix of set
- **numb_test**
  - [int] number 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
**kwargs
    additional arguments

Raise

    RuntimeError
    if no valid system was found


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

Raise

    RuntimeError
    if distributed training job name is wrong

deepmd.entrypoints.transfer(*, old_model: str, raw_model: str, output: str, **kwargs)

Transfer operation from old from 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 transferred parameters will be saved to this location
**kwargs
    additional arguments

Submodules

deeppmd.entrypoints.compress module

Compress a model, which including tabulating the embedding-net.
deeppmd.entrypoints.compress.compress(*, input: str, output: str, extrapolate: int, step: float,
    frequency: str, checkpoint_folder: str, training_script: str,
    mpi_log: str, log_path: Optional[str], 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 specified log will be written to this file
    log_level
        [int] logging level
    **kwargs
        additional arguments
**deepmd.entrypoints.convert module**

`deepmd.entrypoints.convert.convert(*, FROM: str, input_model: str, output_model: str, **kwargs)`

**deepmd.entrypoints.doc module**

`deepmd.entrypoints.doc.doc_train_input(*, out_type: str = 'rst', **kwargs)`

Print out training input arguments to console.

**deepmd.entrypoints.freeze module**

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

`deepmd.entrypoints.freeze.freeze(*, checkpoint_folder: str, output: str, node_names: Optional[str] = None, nvnmnd_weight: Optional[str] = None, united_model: bool = False, **kwargs)`

Freeze the graph in supplied folder.

**Parameters**

- checkpoint_folder
  - [str] location of the folder with model
- output
  - [str] output file name
- node_names
  - [Optional[str], optional] names of nodes to output, by default None
- nvnmnd_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.entrypoints.gui module**

`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.ipi module

Use dp_ipi inside the Python package.

deepmd.entrypoints.ipi.dp_ipi()
    dp_ipi.

deepmd.entrypoints.main module

DeePMD-Kit entry point module.

deepmd.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.entrypoints.main.main(args: Optional[Union[List[str], Namespace]] = 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.entrypoints.main.main_parser() → ArgumentParser
    DeePMD-Kit commandline options argument parser.

    Returns
        argparse.ArgumentParser
            main parser of DeePMD-kit
**deepmd.entrypoints.main.parse_args**

```python
def main.parse_args(args: Optional[List[str]] = None) -> Namespace:
    Parse arguments and convert argument strings to objects.

    Parameters
    ----------
    args : Optional[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.entrypoints.neighbor_stat**

```python
from deepmd.entrypoints import neighbor_stat

def neighbor_stat(*, system: str, rcut: float, type_map: List[str],
                   one_type: bool = False, **kwargs):
    Calculate neighbor statistics.

    Parameters
    ----------
    system : str
             system to stat
    rcut : float
           cutoff radius
    type_map : list[str]
               type map
    one_type : bool, optional, default=False
               treat all types as a single type
    **kwargs
             additional arguments

    Examples
    --------
    >>> neighbor_stat(system='.', rcut=6., type_map=['C', 'H', 'O', 'N', 'P', 'S', '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.entrypoints.test**

```python
from deepmd.entrypoints import test

def test(*, model: str, system: str, datafile: str, set_prefix: str, numb_test: int,
          rand_seed: Optional[int], shuffle_test: bool, detail_file: str, atomic: bool,
          **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
set_prefix
  [str] string prefix of set
numb_test
  [int] number 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
**kwargs
  additional arguments
Raises
  RuntimeError
  if no valid system was found

deepmd.entrypoints.train module

DeePMD training entrypoint script.
Can handle local or distributed training.

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
DeePMD-kit

```
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.entrypoints.transfer module

Module used for transfering parameters between models.

```
deepmd.entrypoints.transfer.transfer(*, old_model: str, raw_model: str, output: str, **kwargs)
 Transfer operation from old from 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
```
class `deepmd.fit.DOSFitting`(*args, **kwargs)

    Bases: `Fitting`

    Fitting the density of states (DOS) of the system. The energy should be shifted by the fermi level.

    Parameters
    ----------
    descrpt : str
        The descriptor $\mathcal{D}$
    neuron : int
        Number of neurons $N$ in each hidden layer of the fitting net
    resnet_dt : float
        Time-step $dt$ in the resnet construction: $y = x + dt \cdot \phi(Wx + b)$
    numb_fparam : int
        Number of frame parameter
    numb_aparam : int
        Number of atomic parameter
    ! numb_dos (added)
        Number of gridpoints on which the DOS is evaluated (NEDOS in VASP)
    rcond : float
        The condition number for the regression of atomic energy.
    trainable : list of bool, optional
        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 : int
        Random seed for initializing the network parameters.
    activation_function : str
        The activation function $\phi$ in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.
    precision : str
        The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.
    uniform_seed : bool, optional
        Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed.
    layer_name : list of 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.

Attributes
----------
DeePMD-kit

**precision**

Precision of fitting network.

**Methods**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(inputs, natoms[, input_dict, reuse, ...])</code></td>
<td>Build the computational graph for fitting net.</td>
</tr>
<tr>
<td><code>compute_input_stats(all_stat[, protection])</code></td>
<td>Compute the input statistics.</td>
</tr>
<tr>
<td><code>compute_output_stats(all_stat[, mixed_type])</code></td>
<td>Compute the output statistics.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Receiving the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function.</td>
</tr>
<tr>
<td><code>get_numb_aparam()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get_numb_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get_numb_fpnum()</code></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, suffix])</code></td>
<td>Init the fitting net variables with the given dict.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a Fitting plugin.</td>
</tr>
</tbody>
</table>

**build**

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

**compute_input_stats**

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_output_stats(all_stat: dict, mixed_type: bool = False) → None

Compute the output 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_attn.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

enable_mixed_precision(mixed_prec: Optional[dict] = None) → None

Receive the mixed precision setting.

Parameters

- mixed_prec
  - The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → Loss

Get the loss function.

Parameters

- loss
  - [dict] the loss dict

- lr
  - [LearningRateExp] the learning rate

Returns

Loss

the loss function

get_numb_aparam() → int

Get the number of atomic parameters.

get_numb_dos() → int

Get the number of gridpoints in energy space.

get_numb_fparam() → int

Get the number of frame parameters.

init_variables(graph: Graph, graph_def: 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
class deepmd.fit.DipoleFittingSeA(*args, **kwargs)

Bases: Fitting

Fit the atomic dipole with descriptor se_a.

Parameters

- **descrpt**
  - `tf.Tensor` The descriptor

- **neuron**
  - `[List[int]]` Number of neurons in each hidden layer of the fitting net

- **resnet_dt**
  - `[bool]` Time-step dt in the resnet construction: $y = x + dt \times \phi(Wx + b)$

- **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**

- **precision**
  - `[str]` The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

- **uniform_seed**
  - Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

Attributes

- **precision**
  - Precision of fitting network.

Methods

- **build**(input_d: Tensor, rot_mat: Tensor, natoms: Tensor[, ...]) → Tensor 
  Build the computational graph for fitting net.

- **enable_mixed_precision**(mixed_prec: Tensor) → Tensor 
  Retrieve the mixed precision setting.

- **get_loss**(loss, lr) → Tensor 
  Get the loss function.

- **get_out_size**() → Tensor 
  Get the output size.

- **get_sel_type**() → Tensor 
  Get selected type.

- **init_variables**(graph, graph_def[, suffix]) → Tensor 
  Init the fitting net variables with the given dict.

- **register**(key) 
  Register a Fitting plugin.


Build the computational graph for fitting net.

Parameters

- **input_d**
  - The input descriptor
rot_mat
The rotation matrix from the descriptor.

natom
The number of atoms. This tensor has the length of Ntypes + 2
natom[0]: number of local atoms natoms[1]: total number of atoms held by this processor
natom[1]: 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.

enable_mixed_precision(mixed_prec: Optional[dict] = None) → None
Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → Loss
Get the loss function.

Parameters

loss
[dict] the loss dict

lr
[LearningRateExp] the learning rate

Returns

Loss
the loss function

get_out_size() → int
Get the output size. Should be 3.

get_sel_type() → int
Get selected type.

init_variables(graph: Graph, graph_def: 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

class deepmd.fit.EnerFitting(*args, **kwargs)

Bases: 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 $D$:

$$E(D) = L^{(n)} \circ L^{(n-1)} \circ \cdots \circ L^{(1)} \circ L^{(0)}$$

The first $n$ hidden layers $L^{(0)}, \ldots, L^{(n-1)}$ are given by

$$y = L(x; w, b) = \phi(x^T w + b)$$

where $x \in \mathbb{R}^{N_1}$ is the input vector and $y \in \mathbb{R}^{N_2}$ is the output vector. $w \in \mathbb{R}^{N_1 \times N_2}$ and $b \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable[$i$] is True. $\phi$ is the activation function.

The output layer $L^{(n)}$ is given by

$$y = L^{(n)}(x; w, b) = x^T w + b$$

where $x \in \mathbb{R}^{N_{n-1}}$ is the input vector and $y \in \mathbb{R}$ is the output scalar. $w \in \mathbb{R}^{N_{n-1}}$ and $b \in \mathbb{R}$ are weights and bias, respectively, both of which are trainable if trainable[$n$] is True.

Parameters

- **descrpt**
  The descriptor $D$

- **neuron**
  Number of neurons $N$ in each hidden layer of the fitting net

- **resnet_dt**
  Time-step $dt$ in the resnet construction: $y = x + dt \times \phi(W x + b)$

- **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 $\phi$ in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.

precision
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

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.

Attributes

**precision**
Precision of fitting network.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>Build the computational graph for fitting net.</td>
</tr>
<tr>
<td>change_energy_bias</td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td>compute_input_stats</td>
<td>Compute the input statistics.</td>
</tr>
<tr>
<td>compute_output_stats</td>
<td>Compute the output statistics.</td>
</tr>
<tr>
<td>enable_mixed_precision</td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td>get_loss</td>
<td>Get the loss function.</td>
</tr>
<tr>
<td>get_numb_aparam</td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td>get_numb_fparam</td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td>init_variables</td>
<td>Init the fitting net variables with the given dict.</td>
</tr>
<tr>
<td>register</td>
<td>Register a Fitting plugin.</td>
</tr>
</tbody>
</table>

**build**
inputs: Tensor, natoms: Tensor, input_dict: Optional[dict] = None, reuse: Optional[bool] = None, suffix: str = "") → 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

change_energy_bias(data, frozen_model, origin_type_map, full_type_map, bias_shift='delta',
nest=10) → 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_shift
[str] The mode for changing energy bias : ['delta', 'statistic'] 'delta' : perform predictions on energies of target dataset,
and do least square on the errors to obtain the target shift as bias.
'statistic': directly use the statistic energy bias in the target dataset.

nest
[int] The number of test samples in a system to change the energy bias.

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_output_stats(all_stat: dict, mixed_type: bool = False) → None
Compute the output 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.

enable_mixed_precision(mixed_prec: Optional[dict] = None) → None
Reveive the mixed precision setting.

Parameters
mixed_prec
The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → Loss
Get the loss function.

Parameters
loss
[d] The loss function parameters.
lr
[LearningRateExp] The learning rate.

Returns
Loss
The loss function.

gt_numb_aparam() → int
Get the number of atomic parameters.

get_numb_fparam() → int
Get the number of frame parameters.

init_variables(graph: Graph, graph_def: 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

class deepmd.fit.Fitting(*args, **kwargs)
Bases: PluginVariant
Attributes
precision
Precision of fitting network.
Methods

```python
get_loss(loss, lr) Get the loss function.
init_variables(graph, graph_def[, suffix]) Init the fitting net variables with the given dict.
register(key) Register a Fitting plugin.
```

abstract get_loss(loss: dict, lr) → Loss
Get the loss function.
Parameters
  loss
    [dict] the loss dict
  lr
    [LearningRateExp] the learning rate
Returns
  Loss
    the loss function

init_variables(graph: Graph, graph_def: 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.

property precision: DType
Precision of fitting network.

static register(key: str) → Callable
Register a Fitting plugin.
Parameters
  key
    [str] the key of a Fitting
Returns
  Fitting
    the registered Fitting
Examples

```python
>>> @Fitting.register("some_fitting")
    class SomeFitting(Fitting):
        pass
```

class deepmd.fit.GlobalPolarFittingSeA(descrpt: Tensor, neuron: List[int] = [120, 120, 120],
            resnet_dt: bool = True, sel_type: Optional[List[int]] = None,
            fit_diag: bool = True, scale: Optional[List[float]] = None,
            diag_shift: Optional[List[float]] = None, seed: Optional[int] = None,
            activation_function: str = 'tanh', precision: str = 'default')

Bases: object

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 “relu”,

precision
    [str] The precision of the embedding net parameters. Supported options are “de-
          fault”, “float16”, “float32”, “float64”, “bfloat16”.

18.2. deepmd package
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(input_d, rot_mat, natoms[, ...])</code></td>
<td>Build the computational graph for fitting net.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function.</td>
</tr>
<tr>
<td><code>get_out_size()</code></td>
<td>Get the output size.</td>
</tr>
<tr>
<td><code>get_sel_type()</code></td>
<td>Get selected atom types.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, suffix])</code></td>
<td>Init the fitting net variables with the given dict.</td>
</tr>
</tbody>
</table>

### build

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 $N_{types} + 2$: number of local atoms $n_{atoms}[1]$: total number of atoms held by this processor $n_{atoms}[i]$: $2 \leq i < N_{types} + 2$, 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)

### enable_mixed_precision

Receive the mixed precision setting.

**Parameters**

- `mixed_prec` (The mixed precision setting used in the embedding net)

### get_loss

Get the loss function.

**Parameters**

- `loss` ([dict] the loss dict)
- `lr` ([LearningRateExp] the learning rate)

**Returns**
Loss
the loss function

get_out_size() → int
Get the output size. Should be 9.

get_sel_type() → int
Get selected atom types.

init_variables(graph: Graph, graph_def: 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

class deepmd.fit.PolarFittingSea(*args, **kwargs)
Bases: Fitting
Fit the atomic polarizability with descriptor se_a.

Parameters

- descrpt
  [tf.Tensor] The descriptor
- 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 “relu”,
precision

[**str**] The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

uniform_seed

Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

Attributes

**precision**

Precision of fitting network.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>build</strong></td>
<td>Build the computational graph for fitting net.</td>
</tr>
<tr>
<td><strong>compute_input_stats</strong></td>
<td>Compute the input statistics.</td>
</tr>
<tr>
<td><strong>enable_mixed_precision</strong></td>
<td>Retrieve the mixed precision setting.</td>
</tr>
<tr>
<td><strong>get_loss</strong></td>
<td>Get the loss function.</td>
</tr>
<tr>
<td><strong>get_out_size</strong></td>
<td>Get the output size.</td>
</tr>
<tr>
<td><strong>get_sel_type</strong></td>
<td>Get selected atom types.</td>
</tr>
<tr>
<td><strong>init_variables</strong></td>
<td>Init the fitting net variables with the given dict.</td>
</tr>
<tr>
<td><strong>register</strong></td>
<td>Register a Fitting plugin.</td>
</tr>
</tbody>
</table>


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

**compute_input_stats** (all_stat, protection=0.01)

Compute the input statistics.
Parameters

    all_stat
    Dictionary of inputs. Can be prepared by model.make_stat_input

    protection
    Divided-by-zero protection

**enable_mixed_precision**(mixed_prec: Optional[dict] = None) → None

Reveive the mixed precision setting.

Parameters

    mixed_prec
    The mixed precision setting used in the embedding net

**get_loss**(loss: dict, lr) → Loss

Get the loss function.

**get_out_size**() → int

Get the output size. Should be 9.

**get_sel_type**() → List[int]

Get selected atom types.

**init_variables**(graph: Graph, graph_def: 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

Submodules

depmd.fit.dipole module

class depmd.fit.dipole.DipoleFittingSeA(*args, **kwargs)

Bases: Fitting

Fit the atomic dipole with descriptor se_a.

Parameters

    descrpt
    [tf.Tensor] The descriptor

    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 “relu”,

precision
   [str] The precision of the embedding net parameters. Supported options are “de-
   fault”, “float16”, “float32”, “float64”, “bfloat16”.

uniform_seed
   Only for the purpose of backward compatibility, retrieves the old behavior of using
   the random seed

Attributes

precision
   Precision of fitting network.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build(input_d, rot_mat, natoms[, ...])</td>
<td>Build the computational graph for fitting net.</td>
</tr>
<tr>
<td>enable_mixed_precision([mixed_prec])</td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td>get_loss(loss, lr)</td>
<td>Get the loss function.</td>
</tr>
<tr>
<td>get_out_size()</td>
<td>Get the output size.</td>
</tr>
<tr>
<td>get_sel_type()</td>
<td>Get selected type.</td>
</tr>
<tr>
<td>init_variables(graph, graph_def[, suffix])</td>
<td>Init the fitting net variables with the given dict.</td>
</tr>
<tr>
<td>register(key)</td>
<td>Register a Fitting plugin.</td>
</tr>
</tbody>
</table>


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[1]:
   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.

enable_mixed_precision(mixed_prec: Optional[dict] = None) → None

    Reiveive the mixed precision setting.

    Parameters

        mixed_prec

            The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → Loss

    Get the loss function.

    Parameters

        loss

            [dict] the loss dict

        lr

            [LearningRateExp] the learning rate

Returns

Loss

    the loss function

get_out_size() → int

    Get the output size. Should be 3.

get_sel_type() → int

    Get selected type.

init_variables(graph: Graph, graph_def: 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
`deepmd.fit.dos module`

`class deepmd.fit.dos.DOSFitting(*args, **kwargs)`

Bases: `Fitting`

Fitting the density of states (DOS) of the system. The energy should be shifted by the fermi level.

Parameters

- **descrpt**
  The descriptor $D$

- **neuron**
  Number of neurons $N$ in each hidden layer of the fitting net

- **resnet_dt**
  Time-step $dt$ in the resnet construction: $y = x + dt \times \phi(Wx + b)$

- **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 $\phi$ in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.

- **precision**
  The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

- **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.

Attributes
Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(inputs, natoms[, input_dict, reuse,...])</code></td>
<td>Build the computational graph for fitting net.</td>
</tr>
<tr>
<td><code>compute_input_stats(all_stat[, protection])</code></td>
<td>Compute the input statistics.</td>
</tr>
<tr>
<td><code>compute_output_stats(all_stat[, mixed_type])</code></td>
<td>Compute the output statistics.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Retrieve the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function.</td>
</tr>
<tr>
<td><code>get_numb_aparam()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get_numb_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get_numb_fpamam()</code></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, suffix])</code></td>
<td>Init the fitting net variables with the given dict.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a Fitting plugin.</td>
</tr>
</tbody>
</table>

`build(inputs: Tensor, natoms: Tensor, input_dict: Optional[dict] = None, reuse: Optional[bool] = None, suffix: str = '') → Tensor`

Build the computational graph for fitting net.

Parameters

- **inputs**
  - The input descriptor

- **input_dict**
  - Additional dict for inputs. if numb_fpamam > 0, should have input_dict['fpamam']
  - if numb_apamam > 0, should have input_dict['apamam']

- **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

`compute_input_stats(all_stat: dict, protection: float = 0.01) → None`

Compute the input statistics.

Parameters

- **all_stat**
  - if numb_fpamam > 0 must have all_stat['fpamam'] if numb_apamam > 0 must have all_stat['apamam'] can be prepared by model.make_stat_input

- **protection**
  - Divided-by-zero protection

18.2. deepmd package
compute_output_stats(all_stat: dict, mixed_type: bool = False) → None
Compute the output 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_attn.md), in which frames in a sys-
    tem may have different natoms_vec(s), with the same nloc.

enable_mixed_precision(mixed_prec: Optional[dict] = None) → None
  Receiv the mixed precision setting.
  Parameters
  mixed_prec
    The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → Loss
Get the loss function.
  Parameters
  loss
    [dict] the loss dict
  lr
    [LearningRateExp] the learning rate
  Returns
  Loss
    the loss function

get_numb_aparam() → int
  Get the number of atomic parameters.

get_numb_dos() → int
  Get the number of gridpoints in energy space.

get_numb_fparam() → int
  Get the number of frame parameters.

init_variables(graph: Graph, graph_def: 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
deepmd.fit.ener module

class deepmd.fit.ener.EnerFitting(*args, **kwargs)
    Bases: 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 $D$:

$$E(D) = L^{(n)} \circ L^{(n-1)} \circ \cdots \circ L^{(1)} \circ L^{(0)}$$

The first $n$ hidden layers $L^{(0)}, \cdots, L^{(n-1)}$ are given by

$$y = L(x; w, b) = \phi(x^T w + b)$$

where $x \in \mathbb{R}^{N_1}$ is the input vector and $y \in \mathbb{R}^{N_2}$ is the output vector. $w \in \mathbb{R}^{N_1 \times N_2}$ and $b \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable[i] is True. $\phi$ is the activation function.

The output layer $L^{(n)}$ is given by

$$y = L^{(n)}(x; w, b) = x^T w + b$$

where $x \in \mathbb{R}^{N_{n-1}}$ is the input vector and $y \in \mathbb{R}$ is the output scalar. $w \in \mathbb{R}^{N_{n-1}}$ and $b \in \mathbb{R}$ are weights and bias, respectively, both of which are trainable if trainable[n] is True.

Parameters

descriptor
    The descriptor $D$

neuron
    Number of neurons $N$ in each hidden layer of the fitting net

resnet_dt
    Time-step dt in the resnet construction: $y = x + dt \ast \phi(Wx + b)$

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 descriptor should be set.
activation_function
The activation function $\phi$ in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.

precision
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

uniform_seed
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed.

layer_name
A list of 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.

Attributes

precision
Precision of fitting network.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build(inputs, natoms[, input_dict, reuse, ...])</td>
<td>Build the computational graph for fitting net.</td>
</tr>
<tr>
<td>change_energy_bias(data, frozen_model, ...)</td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td>compute_input_stats(all_stat[, protection])</td>
<td>Compute the input statistics.</td>
</tr>
<tr>
<td>compute_output_stats(all_stat[, mixed_type])</td>
<td>Compute the output statistics.</td>
</tr>
<tr>
<td>enable_mixed_precision([mixed_prec])</td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td>get_loss(loss, lr)</td>
<td>Get the loss function.</td>
</tr>
<tr>
<td>get_numb_aparam()</td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td>get_numb_fparam()</td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td>init_variables(graph, graph_def[, suffix])</td>
<td>Init the fitting net variables with the given dict.</td>
</tr>
<tr>
<td>register(key)</td>
<td>Register a Fitting plugin.</td>
</tr>
</tbody>
</table>

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

change_energy_bias(data, frozen_model, origin_type_map, full_type_map, bias_shift='delta',
ntest=10) → 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_shift
[str] The mode for changing energy bias : ['delta', 'statistic']
'delta' : perform predictions on energies of target dataset,
and do least square on the errors to obtain the target shift as bias.
'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.

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_output_stats(all_stat: dict, mixed_type: bool = False) → None
Compute the output 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_attn.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

enable_mixed_precision(mixed_prec: Optional[dict] = None) → None
Reveivethemixedprecisionsetting.

Parameters
mixed_prec
The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → Loss
Get the loss function.

Parameters
loss
[dict]The loss function parameters.

lr
[LearningRateExp] The learning rate.

Returns
Loss
The loss function.

get_numb_aparam() → int
Get the number of atomic parameters.

get_numb_fparam() → int
Get the number of frame parameters.

init_variables(graph: Graph, graph_def: 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

deeper.fit.fitting module

class deeper.fit.fitting.Fitting(*args, **kwargs)
Bases: PluginVariant

Attributes

precision
Precision of fitting network.
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, suffix])</code></td>
<td>Init the fitting net variables with the given dict.</td>
</tr>
<tr>
<td><code>register(key)</code></td>
<td>Register a Fitting plugin.</td>
</tr>
</tbody>
</table>

### Abstract `get_loss` (loss: dict, lr) → Loss

Get the loss function.

Parameters

- `loss`: the loss dict
- `lr`: the learning rate

Returns

- `Loss`: the loss function

### `init_variables` (graph: Graph, graph_def: GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

- `graph`: The input frozen model graph
- `graph_def`: The input frozen model graph_def
- `suffix`: suffix to name scope

### Notes

This method is called by others when the fitting supported initialization from the given variables.

### Property `precision`: DType

Precision of fitting network.

### Static `register` (key: str) → Callable

Register a Fitting plugin.

Parameters

- `key`: the key of a Fitting

Returns

- `Fitting`: the registered Fitting
Examples

```python
>>> @Fitting.register("some_fitting")
    class SomeFitting(Fitting):
        pass
```

deepmfitpolar module

class deepmd.fit.polar.GlobalPolarFittingSeA(descrpt: Tensor, neuron: List[int] = [120, 120, 120],
resnet_dt: bool = True, sel_type: Optional[List[int]] = None, fit_diag: bool = True, scale:
Optional[List[float]] = None, diag_shift:
Optional[List[float]] = None, seed: Optional[int] = None, activation_function: str = ‘tanh’, precision: str
= ‘default’)

Bases: object

Fit the system polarizability with descriptor se_a.

Parameters

descrpt
    [tf.Tensor] The descriptor

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 “relu”,

precision
    [str] The precision of the embedding net parameters. Supported options are “de-
fault”, “float16”, “float32”, “float64”, “bfloat16”.

368 Chapter 18. Python API
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(input_d, rot_mat, natoms[, ...])</code></td>
<td>Build the computational graph for fitting net.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Reiveive the mixed precision setting.</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function.</td>
</tr>
<tr>
<td><code>get_out_size()</code></td>
<td>Get the output size.</td>
</tr>
<tr>
<td><code>get_sel_type()</code></td>
<td>Get selected atom types.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, suffix])</code></td>
<td>Init the fitting net variables with the given dict.</td>
</tr>
</tbody>
</table>

**build**

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]: 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

**enable_mixed_precision**

Reiveive the mixed precision setting.

**Parameters**

- **mixed_prec**
The mixed precision setting used in the embedding net

**get_loss**

Get the loss function.

**Parameters**

- **loss**
  - [dict] the loss dict
- **lr**
  - [LearningRateExp] the learning rate

**Returns**

---

18.2. deepmd package 369
**Loss**

the loss function

**get_out_size() → int**

Get the output size. Should be 9.

**get_sel_type() → int**

Get selected atom types.

**init_variables(graph: Graph, graph_def: 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

**class deepmd.fit.polar.PolarFittingSeA(*args, **kwargs)**

Bases: Fitting

Fit the atomic polarizability with descriptor se_a.

Parameters

- **descrpt**
  - [tf.Tensor] The descriptor
- **neuron**
  - [List[int]] Number of neurons in each hidden layer of the fitting net
- **resnet_dt**
  - [bool] Time-step dt in the resnet construction: \( y = x + dt \cdot \phi(Wx + b) \)
- **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**
precision

[**str**] The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

uniform_seed

Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed.

Attributes

**precision**

Precision of fitting network.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>build</strong> (input_d, rot_mat, natoms[,...])</td>
<td>Build the computational graph for fitting net.</td>
</tr>
<tr>
<td><strong>compute_input_stats</strong> (all_stat[, protection])</td>
<td>Compute the input statistics.</td>
</tr>
<tr>
<td><strong>enable_mixed_precision</strong> ([mixed_prec])</td>
<td>Receive the mixed precision setting.</td>
</tr>
<tr>
<td><strong>get_loss</strong> (loss, lr)</td>
<td>Get the loss function.</td>
</tr>
<tr>
<td><strong>get_out_size</strong>()</td>
<td>Get the output size.</td>
</tr>
<tr>
<td><strong>get_sel_type</strong>()</td>
<td>Get selected atom types.</td>
</tr>
<tr>
<td><strong>init_variables</strong> (graph, graph_def[, suffix])</td>
<td>Init the fitting net variables with the given dict.</td>
</tr>
<tr>
<td><strong>register</strong> (key)</td>
<td>Register a Fitting plugin.</td>
</tr>
</tbody>
</table>


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.

**compute_input_stats** (all_stat, protection=0.01)

Compute the input statistics.
Parameters

all_stat
   Dictionary of inputs. can be prepared by model.make_stat_input

protection
   Divided-by-zero protection

enable_mixed_precision(mixed_prec: Optional[dict] = None) → None
   Reiveive the mixed precision setting.

Parameters

mixed_prec
   The mixed precision setting used in the embedding net

get_loss(loss: dict, lr) → Loss
   Get the loss function.

get_out_size() → int
   Get the output size. Should be 9.

get_sel_type() → List[int]
   Get selected atom types.

init_variables(graph: Graph, graph_def: 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

deepmd.infer package

Submodule containing all the implemented potentials.

class deepmd.infer.DeepDOS(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False,
   auto_batch_size: Union[bool, int, AutoBatchSize] = True, input_map: Optional[dict] = None)

   Bases: DeepEval

Constructor.

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
### Attributes

- **model_type**
  Get type of model.

- **model_version**
  Get version of model.

- **sess**
  Get TF session.

### Methods

- **build_neighbor_list(coords, cell, atype, ...)**
  Make the mesh with neighbor list for a single frame.

- **eval(coords, cells, atom_types[, atomic, ...])**
  Evaluate the dos, atom_dos by using this model.

- **eval_descriptor(coords, cells, atom_types[, ...])**
  Evaluate descriptors by using this DP.

- **eval_typeebd()**
  Evaluate output of type embedding network by using this model.

- **get_dim_aparam()**
  Get the number (dimension) of atomic parameters of this DP.

- **get_dim_fparam()**
  Get the number (dimension) of frame parameters of this DP.

- **get_ntypes()**
  Get the number of atom types of this model.

- **get_numb_dos()**
  Get the length of DOS output of this DP model.

- **get_rcut()**
  Get the cut-off radius of this model.

- **get_sel_type()**
  Unsupported in this model.

- **get_type_map()**
  Get the type map (element name of the atom types) of this model.

- **make_natom_vec(atom_types[, mixed_type])**
  Make the natom vector used by deepmd-kit.

- **reverse_map(vec, imap)**
  Reverse mapping of a vector according to the index map.

- **sort_input(coord, atom_type[, sel_atoms, ...])**
  Sort atoms in the system according their types.


Evaluate the dos, atom_dos by using this model.

Parameters
The coordinates of atoms. The array should be of size nframes x natoms x 3

The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

The atom types The list should contain natoms ints

Calculate the atomic energy and virial

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.

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.

Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_attn.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

The electron density of state.

The atom-sited density of state. Only returned when atomic == True

Evaluates descriptors by using this DP.

The coordinates of atoms. The array should be of size nframes x natoms x 3

The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

The atom types The list should contain natoms ints

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.

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.
The external field on atoms. The array should be of size nframes x natoms x 3

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.

get_dim_aparam() \rightarrow \text{int}
Get the number (dimension) of atomic parameters of this DP.

get_dim_fparam() \rightarrow \text{int}
Get the number (dimension) of frame parameters of this DP.

get_ntypes() \rightarrow \text{int}
Get the number of atom types of this model.

get_numb_dos() \rightarrow \text{int}
Get the length of DOS output of this DP model.

get_rcut() \rightarrow \text{float}
Get the cut-off radius of this model.

get_sel_type() \rightarrow \text{List[int]}
Unsupported in this model.

get_type_map() \rightarrow \text{List[str]}
Get the type map (element name of the atom types) of this model.

load_prefix: \text{str}


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

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.
Warning: For developers: DeepTensor initializer must be called at the end after self.tensors are modified because it uses the data in self.tensors dict. Do not change the order!

Attributes

model_type
Get type of model.

model_version
Get version of model.

sess
Get TF session.

Methods

build_neighbor_list(coords, cell, atype, ...)
Make the mesh with neighbor list for a single frame.

eval(coords, cells, atom_types[, atomic, ...])
Evaluate the model.

eval_full(coords, cells, atom_types[, ...])
Evaluate the model with interface similar to the energy model.

eval_typeebd()
Evaluate output of type embedding network by using this model.

get_dim_aparam()
Unsupported in this model.

get_dim_fparam()
Unsupported in this model.

get_ntypes()
Get the number of atom types of this model.

get_rcut()
Get the cut-off radius of this model.

get_sel_type()
Get the selected atom types of this model.

get_type_map()
Get the type map (element name of the atom types) of this model.

make_natoms_vec(atom_types[, mixed_type])
Make the natom vector used by deepmd-kit.

reverse_map(vec, imap)
Reverse mapping of a vector according to the index map.

sort_input(coord, atom_type[, sel_atoms, ...])
Sort atoms in the system according their types.

get_dim_aparam() → int
Unsupported in this model.

get_dim_fparam() → int
Unsupported in this model.


Bases: object

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.

Attributes

model_type
Get type of model.

model_version
Get version of model.

sess
Get TF session.

Methods

build_neighbor_list(coords, cell, atype, ...) Make the mesh with neighbor list for a single frame.

eval_typepdb() Evaluate output of type embedding network by using this model.

make_natoms_vec(atom_types[, mixed_type]) Make the natom vector used by deepmd-kit.

reverse_map(vec, imap) Reverse mapping of a vector according to the index map.

sort_input(coord, atom_type[, sel_atoms, ...]) Sort atoms in the system according their types.

build_neighbor_list(coords: ndarray, cell: Optional[ndarray], atype: ndarray, imap: ndarray, neighbor_list)
Make the mesh with neighbor list for a single frame.

Parameters

coods
[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

natom_vec

[np.ndarray] The number of atoms. This tensor has the length of Ntypes + 2
natom[0]: nloc natom[1]: nall natom[i]: 2 <= i < Ntypes+2, number of type
i atoms for nloc

cords

[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


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]

eval_typeebd() → 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.utils.type_embed.TypeEmbedNet

The type embedding network.

Examples

Get the output of type embedding network of graph.pb:

```python
>>> from deepmd.infer import DeepPotential
>>> dp = DeepPotential('graph.pb')
>>> dp.eval_typeebd()
```

load_prefix: str

make_natom_vec(atom_types: ndarray, mixed_type: bool = False) → 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_attn.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

property model_type: str
Get type of model.
:type:str

property model_version: str
Get version of model.

Returns

str
version of model

static reverse_map(vec: ndarray, imap: List[int]) → 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.

property sess: Session
Get TF session.

static sort_input(coord: ndarray, atom_type: ndarray, sel_atoms: Optional[List[int]] = 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_attn.md), in which frames in a sys-
tem 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 = co-
ord[:,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.

class deepmd.infer.DeepGlobalPolar(model_file: str, load_prefix: str = 'load',
default_tf_graph: bool = False, neighbor_list=None)
Bases: DeepTensor
Constructor.

Parameters

model_file
[st] 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

neighbor_list
[ase.neighborlist.NeighborList, optional] The neighbor list object. If None,
then build the native neighbor list.

Attributes

model_type
Get type of model.

model_version
Get version of model.

sess
Get TF session.
## Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build_neighbor_list</code></td>
<td>Make the mesh with neighbor list for a single frame.</td>
</tr>
<tr>
<td><code>eval</code></td>
<td>Evaluate the model.</td>
</tr>
<tr>
<td><code>eval_full</code></td>
<td>Evaluate the model with interface similar to the energy model.</td>
</tr>
<tr>
<td><code>eval_typeebd</code></td>
<td>Evaluate output of type embedding network by using this model.</td>
</tr>
<tr>
<td><code>get_dim_aparam</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_dim_fparam</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_ntypes</code></td>
<td>Get the number of atom types of this model.</td>
</tr>
<tr>
<td><code>get_rcut</code></td>
<td>Get the cut-off radius of this model.</td>
</tr>
<tr>
<td><code>get_sel_type</code></td>
<td>Get the selected atom types of this model.</td>
</tr>
<tr>
<td><code>get_type_map</code></td>
<td>Get the type map (element name of the atom types) of this model.</td>
</tr>
<tr>
<td><code>make_natoms_vec</code></td>
<td>Make the natom vector used by deepmd-kit.</td>
</tr>
<tr>
<td><code>reverse_map</code></td>
<td>Reverse mapping of a vector according to the index map.</td>
</tr>
<tr>
<td><code>sort_input</code></td>
<td>Sort atoms in the system according their types.</td>
</tr>
</tbody>
</table>

### `eval`

**signature**

```
```

**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`:
  Not used in this model

- `fparam`:
  Not used in this model

- `aparam`:
  Not used in this model

- `efield`:
  Not used in this model

**Returns**

- `tensor`:
  The returned tensor. If atomic == False then of size nframes x variable_dof else of size nframes x natoms x variable_dof
DeePMD-kit

```python
def get_dim_aparam() -> int
    Unsupported in this model.

def get_dim_fparam() -> int
    Unsupported in this model.

class deepmd.infer.DeepPolar(model_file: Path, load_prefix: str = 'load',
                           default_tf_graph: bool = False, input_map: Optional[dict] = None,
                           neighbor_list: Optional = None):
    bases: DeepTensor

    Constructor.

    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.
    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.

    Warning: For developers: DeepTensor initializer must be called at the end after self.tensors are modified because it uses the data in self.tensors dict. Do not change the order!

Attributes
----------
model_type
    Get type of model.
model_version
    Get version of model.
sess
    Get TF session.
```

Chapter 18. Python API
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build_neighbor_list(coords, cell, atype, ...)</code></td>
<td>Make the mesh with neighbor list for a single frame.</td>
</tr>
<tr>
<td><code>eval(coords, cells, atom_types[, atomic, ...])</code></td>
<td>Evaluate the model.</td>
</tr>
<tr>
<td><code>eval_full(coords, cells, atom_types[, ...])</code></td>
<td>Evaluate the model with interface similar to the energy model.</td>
</tr>
<tr>
<td><code>eval_typeebd()</code></td>
<td>Evaluate output of type embedding network by using this model.</td>
</tr>
<tr>
<td><code>get_dim_aparam()</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_dim_fparam()</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Get the number of atom types of this model.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get the cut-off radius of this model.</td>
</tr>
<tr>
<td><code>get_sel_type()</code></td>
<td>Get the selected atom types of this model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map (element name of the atom types) of this model.</td>
</tr>
<tr>
<td><code>make_natoms_vec(atom_types[, mixed_type])</code></td>
<td>Make the natom vector used by deepmd-kit.</td>
</tr>
<tr>
<td><code>reverse_map(vec, imap)</code></td>
<td>Reverse mapping of a vector according to the index map.</td>
</tr>
<tr>
<td><code>sort_input(coord, atom_type[, sel_atoms, ...])</code></td>
<td>Sort atoms in the system according their types.</td>
</tr>
</tbody>
</table>

### get_dim_aparam() → int
Unsupported in this model.

### get_dim_fparam() → int
Unsupported in this model.

```python
```

**Bases:** `DeepEval`

**Constructor.**

**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 AutoBatchSize, default: True] 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.
Warning: For developers: DeepTensor initializer must be called at the end after self.tensors are modified because it uses the data in self.tensors dict. Do not change the order!

Examples

```python
gg from deepmd.infer import DeepPot
gg import numpy as np
gg dp = DeepPot('graph.pb')
gg coord = np.array([[1,0,0], [0,0,1.5], [1,0,3]]).reshape([1, -1])
gg cell = np.diag(10 * np.ones(3)).reshape([1, -1])
gg atype = [1,0,1]
gg e, f, v = dp.eval(coord, cell, atype)
```

where e, f and v are predicted energy, force and virial of the system, respectively.

Attributes

- **model_type**: Get type of model.
- **model_version**: Get version of model.
- **sess**: Get TF session.

Methods

- **build_neighbor_list(coords, cell, atype, ...)**: Make the mesh with neighbor list for a single frame.
- **eval(coords, cells, atom_types[, atomic, ...])**: Evaluate the energy, force and virial by using this DP.
- **eval_descriptor(coords, cells, atom_types[, ...])**: Evaluate descriptors by using this DP.
- **eval_typeebd()**: Evaluate output of type embedding network by using this model.
- **get_descriptor_type()**: Get the descriptor type of this model.
- **get_dim_aparam()**: Get the number (dimension) of atomic parameters of this DP.
- **get_dim_fparam()**: Get the number (dimension) of frame parameters of this DP.
- **get_natypes()**: Get the number of atom types of this model.
- **get_natypes_spin()**: Get the number of spin atom types of this model.
- **get_rcut()**: Get the cut-off radius of this model.
- **get_sel_type()**: Unsupported in this model.
- **get_type_map()**: Get the type map (element name of the atom types) of this model.
- **make_natoms_vec(atom_types[, mixed_type])**: Make the natom vector used by deepmd-kit.
- **reverse_map(vec, imap)**: Reverse mapping of a vector according to the index map.
- **sort_input(coord, atom_type[, sel_atoms, ...])**: Sort atoms in the system according their types.

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

mixed_type
   Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_attn.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

**Returns**

energy
   The system energy.

force
   The force on each atom

virial
   The virial

atom_energy
   The atomic energy. Only returned when atomic == True

atom_virial
   The atomic virial. Only returned when atomic == True


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.

get_descriptor_type() → List[int]
Get the descriptor type of this model.

get_dim_aparam() → int
Get the number (dimension) of atomic parameters of this DP.

get_dim_fparam() → int
Get the number (dimension) of frame parameters of this DP.

get_ntypes() → int
Get the number of atom types of this model.

get_ntypes_spin()
Get the number of spin atom types of this model.

get_rcut() → float
Get the cut-off radius of this model.

get_sel_type() → List[int]
Unsupported in this model.

get_type_map() → List[str]
Get the type map (element name of the atom types) of this model.

load_prefix: str
Factory function that will initialize appropriate potential read from model_file.

Parameters

- **model_file**: The name of the frozen model file.
- **load_prefix**: The prefix in the load computational graph.
- **default_tf_graph**: If uses the default tf graph, otherwise build a new tf graph for evaluation.
- **input_map**: The input map for tf.import_graph_def. Only work with default tf graph.
- **neighbor_list**: The neighbor list object. If None, then build the native neighbor list.

Returns


Raises

- **RuntimeError** if model file does not correspond to any implemented potential.

### class deepmd.infer.DeepWFC

**Bases:** DeepTensor

**Constructor.**

**Parameters**

- **model_file**: The name of the frozen model file.
- **load_prefix**: str
- **default_tf_graph**: bool
- **input_map**: dict

**Warning:** For developers: DeepTensor initializer must be called at the end after self.tensors are modified because it uses the data in self.tensors dict. Do not change the order!

**Attributes**
**model_type**
Get type of model.

**model_version**
Get version of model.

**sess**
Get TF session.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>build_neighbor_list</strong>(coords, cell, atype,...)</td>
<td>Make the mesh with neighbor list for a single frame.</td>
</tr>
<tr>
<td><strong>eval</strong>(coords, cells, atom_types[, atomic,...])</td>
<td>Evaluate the model.</td>
</tr>
<tr>
<td><strong>eval_full</strong>(coords, cells, atom_types[, ...])</td>
<td>Evaluate the model with interface similar to the energy model.</td>
</tr>
<tr>
<td><strong>eval_typeebd</strong></td>
<td>Evaluate output of type embedding network by using this model.</td>
</tr>
<tr>
<td><strong>get_dim_aparam</strong></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><strong>get_dim_fparam</strong></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><strong>get_natypes</strong></td>
<td>Get the number of atom types of this model.</td>
</tr>
<tr>
<td><strong>get_rcut</strong></td>
<td>Get the cut-off radius of this model.</td>
</tr>
<tr>
<td><strong>get_sel_type</strong></td>
<td>Get the selected atom types of this model.</td>
</tr>
<tr>
<td><strong>get_type_map</strong></td>
<td>Get the type map (element name of the atom types) of this model.</td>
</tr>
<tr>
<td><strong>make_natoms_vec</strong>(atom_types[, mixed_type])</td>
<td>Make the natom vector used by deepmd-kit.</td>
</tr>
<tr>
<td><strong>reverse_map</strong>(vec, imap)</td>
<td>Reverse mapping of a vector according to the index map.</td>
</tr>
<tr>
<td><strong>sort_input</strong>(coord, atom_type[, sel_atoms,...])</td>
<td>Sort atoms in the system according their types.</td>
</tr>
</tbody>
</table>

**get_dim_aparam** → int
Unsupported in this model.

**get_dim_fparam** → int
Unsupported in this model.

**class deepmd.infer.DipoleChargeModifier**(model_name: str, model_charge_map: List[float], sys_charge_map: List[float], ewald_h: float = 1, ewald_beta: float = 1)

Bases: DeepDipole

Parameters

model_name
The model file for the DeepDipole model

model_charge_map
Gives the amount of charge for the fcc

sys_charge_map
Gives the amount of charge for the real atoms

ewald_h
Grid spacing of the reciprocal part of Ewald sum. Unit: Å
ewald\_beta
Splitting parameter of the Ewald sum. Unit: \( \text{Å}^{-1} \)

Attributes

- **model\_type**
  Get type of model.

- **model\_version**
  Get version of model.

- **sess**
  Get TF session.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build\_fv\_graph()</code></td>
<td>Build the computational graph for the force and virial inference.</td>
</tr>
<tr>
<td><code>build\_neighbor\_list(coords, cell, atype, ...)</code></td>
<td>Make the mesh with neighbor list for a single frame.</td>
</tr>
<tr>
<td><code>eval(coord, box, atype[, eval\_fv])</code></td>
<td>Evaluate the modification.</td>
</tr>
<tr>
<td><code>eval\_full(coords, cells, atom\_types[, ...])</code></td>
<td>Evaluate the model with interface similar to the energy model.</td>
</tr>
<tr>
<td><code>eval\_type\_ebd()</code></td>
<td>Evaluate output of type embedding network by using this model.</td>
</tr>
<tr>
<td><code>get\_dim\_a\_param()</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get\_dim\_device\_param()</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get\_ntypes()</code></td>
<td>Get the number of atom types of this model.</td>
</tr>
<tr>
<td><code>get\_rcut()</code></td>
<td>Get the cut-off radius of this model.</td>
</tr>
<tr>
<td><code>get\_sel\_type()</code></td>
<td>Get the selected atom types of this model.</td>
</tr>
<tr>
<td><code>get\_type\_map()</code></td>
<td>Get the type map (element name of the atom types) of this model.</td>
</tr>
<tr>
<td><code>make\_natoms\_vec(atom\_types[, mixed\_type])</code></td>
<td>Make the natom vector used by deepmd-kit.</td>
</tr>
<tr>
<td><code>modify\_data(data, data\_sys)</code></td>
<td>Modify data.</td>
</tr>
<tr>
<td><code>reverse\_map(vec, imap)</code></td>
<td>Reverse mapping of a vector according to the index map.</td>
</tr>
<tr>
<td><code>sort\_input(coord, atom\_type[, sel\_atoms, ...])</code></td>
<td>Sort atoms in the system according to their types.</td>
</tr>
</tbody>
</table>

**`build\_fv\_graph()`** → Tensor
Build the computational graph for the force and virial inference.

**`eval(coord: ndarray, box: ndarray, atype: ndarray, eval\_fv: bool = True) → Tuple[ndarray, ndarray]`**
Evaluate the modification.

**Parameters**

- **coord**
  The coordinates of atoms

- **box**
  The simulation region. PBC is assumed

- **atype**
  The atom types
DeePMD-kit

eval_fv
Evaluate force and virial

Returns

tot_e
The energy modification
tot_f
The force modification
tot_v
The virial modification
modify_data(data: dict, data_sys: 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.infer.EwaldRecp(hh, beta)
Bases: object
Evaluate the reciprocal part of the Ewald sum.

Methods

eval(coord, charge, box)
Evaluate.

eval(coord: ndarray, charge: ndarray, box: ndarray) → Tuple[ndarray, ndarray, 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.infer.calc_model_devi(coord, box, atype, models, fname=None, frequency=1,

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

```python
>>> 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)
```

Submodules

deepmd.infer.data_modifier module

class deepmd.infer.data_modifier.DipoleChargeModifier

Bases: DeepDipole

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: Å
- **ewald_beta**
  Splitting parameter of the Ewald sum. Unit: Å^{-1}

Attributes

- **model_type**
  Get type of model.
- **model_version**
  Get version of model.
- **sess**
  Get TF session.
### Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build_fv_graph()</code></td>
<td>Build the computational graph for the force and virial inference.</td>
</tr>
<tr>
<td><code>build_neighbor_list()</code></td>
<td>Make the mesh with neighbor list for a single frame.</td>
</tr>
<tr>
<td><code>eval(coord, box, atype[, eval_fv])</code></td>
<td>Evaluate the modification.</td>
</tr>
<tr>
<td><code>eval_full(coords, cells, atom_types[, ...])</code></td>
<td>Evaluate the model with interface similar to the energy model.</td>
</tr>
<tr>
<td><code>eval_typeebd()</code></td>
<td>Evaluate output of type embedding network by using this model.</td>
</tr>
<tr>
<td><code>get_dim_aparam()</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_dim_fparam()</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Get the number of atom types of this model.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get the cut-off radius of this model.</td>
</tr>
<tr>
<td><code>get_sel_type()</code></td>
<td>Get the selected atom types of this model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map (element name of the atom types) of this model.</td>
</tr>
<tr>
<td><code>make_natoms_vec(atom_types[, mixed_type])</code></td>
<td>Make the natom vector used by deepmd-kit.</td>
</tr>
<tr>
<td><code>modify_data(data, data_sys)</code></td>
<td>Modify data.</td>
</tr>
<tr>
<td><code>reverse_map(vec, imap)</code></td>
<td>Reverse mapping of a vector according to the index map.</td>
</tr>
<tr>
<td><code>sort_input(coord, atom_type[, sel_atoms, ...])</code></td>
<td>Sort atoms in the system according their types.</td>
</tr>
</tbody>
</table>

**build_fv_graph() → Tensor**

Build the computational graph for the force and virial inference.

**eval(coord: ndarray, box: ndarray, atype: ndarray, eval_fv: bool = True) → Tuple[ndarray, 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
modify_data(data: dict, data_sys: 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.infer.deep_dipole module

class deepmd.infer.deep_dipole.DeepDipole(model_file: Path, load_prefix: str = 'load',
default_tf_graph: bool = False, input_map: Optional[dict] = None, neighbor_list=None)

Bases: DeepTensor

Constructor.

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.

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.

Warning: For developers: DeepTensor initializer must be called at the end after self.tensors are modified because it uses the data in self.tensors dict. Do not change the order!

Attributes

model_type
Get type of model.

model_version
Get version of model.

sess
Get TF session.
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build_neighbor_list(coords, cell, atype, ...)</code></td>
<td>Make the mesh with neighbor list for a single frame.</td>
</tr>
<tr>
<td><code>eval(coords, cells, atom_types[, atomic, ...])</code></td>
<td>Evaluate the model.</td>
</tr>
<tr>
<td><code>eval_full(coords, cells, atom_types[, ...])</code></td>
<td>Evaluate the model with interface similar to the energy model.</td>
</tr>
<tr>
<td><code>eval_typeebd()</code></td>
<td>Evaluate output of type embedding network by using this model.</td>
</tr>
<tr>
<td><code>get_dim_aparam()</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_dim_fparam()</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Get the number of atom types of this model.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get the cut-off radius of this model.</td>
</tr>
<tr>
<td><code>get_sel_type()</code></td>
<td>Get the selected atom types of this model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map (element name of the atom types) of this model.</td>
</tr>
<tr>
<td><code>make_natoms_vec(atom_types[, mixed_type])</code></td>
<td>Make the natom vector used by deepmd-kit.</td>
</tr>
<tr>
<td><code>reverse_map(vec, imap)</code></td>
<td>Reverse mapping of a vector according to the index map.</td>
</tr>
<tr>
<td><code>sort_input(coord, atom_type[, sel_atoms, ...])</code></td>
<td>Sort atoms in the system according their types.</td>
</tr>
</tbody>
</table>

### get_dim_aparam() → int

Unsupported in this model.

### get_dim_fparam() → int

Unsupported in this model.

## deepmd.infer.deep_dos module

```python
```

**Bases:** `DeepEval`

Constructor.

**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: True  
  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
Warning: For developers: DeepTensor initializer must be called at the end after self.tensors are modified because it uses the data in self.tensors dict. Do not change the order!

Attributes

**model_type**
Get type of model.

**model_version**
Get version of model.

**sess**
Get TF session.

Methods

**build_neighbor_list**(coords, cell, atype, ...)
Make the mesh with neighbor list for a single frame.

**eval**(coords, cells, atom_types[, atomic, ...])
Evaluate the dos, atom_dos by using this model.

**eval_descriptor**(coords, cells, atom_types[, ...])
Evaluate descriptors by using this DP.

**eval_typeebd**()
Evaluate output of type embedding network by using this model.

**get_dim_aparam**()
Get the number (dimension) of atomic parameters of this DP.

**get_dim_fparam**()
Get the number (dimension) of frame parameters of this DP.

**get_ntypes**()
Get the number of atom types of this model.

**get_numb_dos**()
Get the length of DOS output of this DP model.

**get_rcut**()
Get the cut-off radius of this model.

**get_sel_type**()
Unsupported in this model.

**get_type_map**()
Get the type map (element name of the atom types) of this model.

**make_natoms_vec**(atom_types[, mixed_type])
Make the natom vector used by deepmd-kit.

**reverse_map**(vec, imap)
Reverse mapping of a vector according to the index map.

**sort_input**(coord, atom_type[, sel_atoms, ...])
Sort atoms in the system according their types.

**eval**(coords: ndarray, cells: ndarray, atom_types: List[int], atomic: bool = False, fparam:
  Optional[ndarray] = None, aparam: Optional[ndarray] = None, mixed_type: bool = False) →
  Tuple[ndarray, ...]
Evaluate the dos, atom_dos by using this 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
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.

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_attn.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

dos
The electron density of state.

atom_dos
The atom-sited density of state. Only returned when atomic == True

eval_descriptor

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_attn.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns
DeePMD-kit

**descriptor**

Descriptors.

- **get_dim_aparam() → int**
  Get the number (dimension) of atomic parameters of this DP.

- **get_dim_fparam() → int**
  Get the number (dimension) of frame parameters of this DP.

- **get_ntypes() → int**
  Get the number of atom types of this model.

- **get_numb_dos() → int**
  Get the length of DOS output of this DP model.

- **get_rcut() → float**
  Get the cut-off radius of this model.

- **get_sel_type() → List[int]**
  Unsupported in this model.

- **get_type_map() → List[str]**
  Get the type map (element name of the atom types) of this model.

**load_prefix: str**

---

**deepmd.infer.deep_eval module**


**Parameters**

- **model_file**
  [Path] The name of the frozen model file.

- **load_prefix**
  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.
**Attributes**

- **model_type**
  Get type of model.

- **model_version**
  Get version of model.

- **sess**
  Get TF session.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>build_neighbor_list</strong></td>
<td>Make the mesh with neighbor list for a single frame.</td>
</tr>
<tr>
<td></td>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td></td>
<td>coords: [np.ndarray], cell: (Optional[np.ndarray]), atype: [np.ndarray], imap: [np.ndarray], neighbor_list: [ase.neighborlist.NewPrimitiveNeighborList]</td>
</tr>
<tr>
<td></td>
<td>Parameters: <strong>coords</strong></td>
</tr>
<tr>
<td></td>
<td>[np.ndarray] The coordinates of atoms. Should be of shape [natoms, 3]</td>
</tr>
<tr>
<td></td>
<td>cell: (Optional[np.ndarray]) The cell of the system. Should be of shape [3, 3]</td>
</tr>
<tr>
<td></td>
<td>atype: [np.ndarray] The type of atoms. Should be of shape [natoms]</td>
</tr>
<tr>
<td></td>
<td>imap: [np.ndarray] The index map of atoms. Should be of shape [natoms]</td>
</tr>
<tr>
<td></td>
<td>neighbor_list: [ase.neighborlist.New Primitive Neighbor List] ASEC neighbor list. The following method or attribute will be used/set: bothways, self_interaction, update, build, first_neigh, pair_second, offset_vec.</td>
</tr>
<tr>
<td></td>
<td>Returns</td>
</tr>
<tr>
<td></td>
<td>natoms_vec: [np.ndarray] The number of atoms. This tensor has the length of Ntypes + 2</td>
</tr>
<tr>
<td></td>
<td>natoms[0]: nloc natoms[1]: null natoms[1]: 2 &lt;= i &lt; Ntypes+2, number of type</td>
</tr>
<tr>
<td></td>
<td>i atoms for nloc</td>
</tr>
<tr>
<td></td>
<td>coords: [np.ndarray] The coordinates of atoms, including ghost atoms. Should be of shape</td>
</tr>
<tr>
<td></td>
<td>[nframes, null, 3]</td>
</tr>
<tr>
<td></td>
<td>atype: [np.ndarray] The type of atoms, including ghost atoms. Should be of shape</td>
</tr>
<tr>
<td></td>
<td>[null]</td>
</tr>
</tbody>
</table>

**eval_typeebd()**

Evaluate output of type embedding network by using this model.

**make_natoms_vec(atom_types[, mixed_type])**

Make the natom vector used by deepmd-kit.

**reverse_map(vec, imap)**

Reverse mapping of a vector according to the index map.

**sort_input(coord, atom_type[, sel_atoms,...])**

Sort atoms in the system according their types.
DeePMD-kit

mesh

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]

eval_typeebd() → 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:

depmd.utils.type_embed.TypeEmbedNet
The type embedding network.

Examples

Get the output of type embedding network of graph.pb:

```python
>>> from deepmd.infer import DeepPotential
>>> dp = DeepPotential('graph.pb')
>>> dp.eval_typeebd()
```

load_prefix: str

make_natoms_vec(atom_types: ndarray, mixed_type: bool = False) → 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_attn.md), in which frames in a sys-
tem 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[1]:
2 <= i < Ntypes+2, number of type i atoms
property model_type: \texttt{str}
Get type of model.

property model_version: \texttt{str}
Get version of model.

Returns
\texttt{str}
version of model

static reverse_map(vec: ndarray, imap: List[int]) → 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.

property sess: \texttt{Session}
Get TF session.

static sort_input(coord: ndarray, atom_type: ndarray, sel_atoms: Optional[List[int]] = None, mixed_type: bool = False)
Sort atoms in the system according thei 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_attn.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,:]
DeePMD-kit

```

```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.

**deepmd.infer.deep_polar module**

```
class deepmd.infer.deep_polar.DeepGlobalPolar(model_file: str, load_prefix: str = 'load',
                                              default_tf_graph: bool = False,
                                              neighbor_list=None)
```

Bases: `DeepTensor`

Constructor.

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

- `neighbor_list`:
  [ase.neighborlist.NeighborList, optional] The neighbor list object. If None, then build the native neighbor list.

Attributes

- `model_type`
  Get type of model.

- `model_version`
  Get version of model.

- `sess`
  Get TF session.
Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build_neighbor_list</code></td>
<td>Make the mesh with neighbor list for a single frame.</td>
</tr>
<tr>
<td><code>eval</code></td>
<td>Evaluate the model.</td>
</tr>
<tr>
<td><code>eval_full</code></td>
<td>Evaluate the model with interface similar to the energy model.</td>
</tr>
<tr>
<td><code>eval_typeebd</code></td>
<td>Evaluate output of type embedding network by using this model.</td>
</tr>
<tr>
<td><code>get_dim_aparam</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_dim_fparam</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_ntypes</code></td>
<td>Get the number of atom types of this model.</td>
</tr>
<tr>
<td><code>get_rcut</code></td>
<td>Get the cut-off radius of this model.</td>
</tr>
<tr>
<td><code>get_sel_type</code></td>
<td>Get the selected atom types of this model.</td>
</tr>
<tr>
<td><code>get_type_map</code></td>
<td>Get the type map (element name of the atom types) of this model.</td>
</tr>
<tr>
<td><code>make_natoms_vec</code></td>
<td>Make the natom vector used by deepmd-kit.</td>
</tr>
<tr>
<td><code>reverse_map</code></td>
<td>Reverse mapping of a vector according to the index map.</td>
</tr>
<tr>
<td><code>sort_input</code></td>
<td>Sort atoms in the system according their types.</td>
</tr>
</tbody>
</table>

**eval**

```python
```

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`:
  - Not used in this model
- `fparam`:
  - Not used in this model
- `aparam`:
  - Not used in this model
- `efield`:
  - Not used in this model

Returns

- `tensor`:
  - The returned tensor. If atomic == False then of size nframes x variable_dof else of size nframes x natoms x variable_dof
get_dim_aparam() → int
Unsupported in this model.

get_dim_fparam() → int
Unsupported in this model.

class deepmd.infer.deep_polar.DeepPolar(model_file: Path, load_prefix: str = 'load',
   default_tf_graph: bool = False, input_map: Optional[dict] = None, neighbor_list=None)

Bases: DeepTensor

Constructor.

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

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.

Warning: For developers: DeepTensor initializer must be called at the end after self.tensors are
modified because it uses the data in self.tensors dict. Do not change the order!

Attributes

model_type
   Get type of model.

model_version
   Get version of model.

sess
   Get TF session.
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build_neighbor_list(coords, cell, atype, ...)</code></td>
<td>Make the mesh with neighbor list for a single frame.</td>
</tr>
<tr>
<td><code>eval(coords, cells, atom_types[, atomic, ...])</code></td>
<td>Evaluate the model.</td>
</tr>
<tr>
<td><code>eval_full(coords, cells, atom_types[, ...])</code></td>
<td>Evaluate the model with interface similar to the energy model.</td>
</tr>
<tr>
<td><code>eval_typeebd()</code></td>
<td>Evaluate output of type embedding network by using this model.</td>
</tr>
<tr>
<td><code>get_dim_aparam()</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_dim_fparam()</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_nparams()</code></td>
<td>Get the number of atom types of this model.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get the cut-off radius of this model.</td>
</tr>
<tr>
<td><code>get_sel_type()</code></td>
<td>Get the selected atom types of this model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map (element name of the atom types) of this model.</td>
</tr>
<tr>
<td><code>make_natoms_vec(atom_types[, mixed_type])</code></td>
<td>Make the natom vector used by deepmd-kit.</td>
</tr>
<tr>
<td><code>reverse_map(vec, imap)</code></td>
<td>Reverse mapping of a vector according to the index map.</td>
</tr>
<tr>
<td><code>sort_input(coord, atom_type[, sel_atoms, ...])</code></td>
<td>Sort atoms in the system according their types.</td>
</tr>
</tbody>
</table>

### `get_dim_aparam()` → int

Unsupported in this model.

### `get_dim_fparam()` → int

Unsupported in this model.

### deepmd.infer.deep_pot module


Bases: `DeepEval`

Constructor.

- **Parameters**
  - `model_file`
    - [Path] The name of the frozen model file.
  - `load_prefix`
    - 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: True] 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.

Warning: For developers: DeepTensor initializer must be called at the end after self.tensors are
modified because it uses the data in self.tensors dict. Do not change the order!

Examples

```python
>>> 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.

Attributes

model_type
Get type of model.

model_version
Get version of model.

sess
Get TF session.
Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build_neighbor_list</code> (coords, cell, atype, ...)</td>
<td>Make the mesh with neighbor list for a single frame.</td>
</tr>
<tr>
<td><code>eval</code> (coords, cells, atom_types[, atomic, ...])</td>
<td>Evaluate the energy, force and virial by using this DP.</td>
</tr>
<tr>
<td><code>eval_descriptor</code> (coords, cells, atom_types[, ...])</td>
<td>Evaluate descriptors by using this DP.</td>
</tr>
<tr>
<td><code>eval_typeebd</code> ()</td>
<td>Evaluate output of type embedding network by using this model.</td>
</tr>
<tr>
<td><code>get_descriptor_type</code> ()</td>
<td>Get the descriptor type of this model.</td>
</tr>
<tr>
<td><code>get_dim_aparam</code> ()</td>
<td>Get the number (dimension) of atomic parameters of this DP.</td>
</tr>
<tr>
<td><code>get_dim_fparam</code> ()</td>
<td>Get the number (dimension) of frame parameters of this DP.</td>
</tr>
<tr>
<td><code>get_ntypes</code> ()</td>
<td>Get the number of atom types of this model.</td>
</tr>
<tr>
<td><code>get_ntypes_spin</code> ()</td>
<td>Get the number of spin atom types of this model.</td>
</tr>
<tr>
<td><code>get_rcut</code> ()</td>
<td>Get the cut-off radius of this model.</td>
</tr>
<tr>
<td><code>get_sel_type</code> ()</td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_type_map</code> ()</td>
<td>Get the type map (element name of the atom types) of this model.</td>
</tr>
<tr>
<td><code>make_natoms_vec</code> (atom_types[, mixed_type])</td>
<td>Make the natom vector used by deepmd-kit.</td>
</tr>
<tr>
<td><code>reverse_map</code> (vec, imap)</td>
<td>Reverse mapping of a vector according to the index map.</td>
</tr>
<tr>
<td><code>sort_input</code> (coord, atom_type[, sel_atoms, ...])</td>
<td>Sort atoms in the system according to their types.</td>
</tr>
</tbody>
</table>


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.
DeePMD-kit

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

energy
The system energy.

force
The force on each atom

virial
The virial

atom_energy
The atomic energy. Only returned when atomic == True

atom_virial
The atomic virial. Only returned when atomic == True


Evaluate descriptors by using this DP.

Parameters

coods
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

408 Chapter 18. Python API
descriptor
  Descriptors.

get_descriptor_type() -> List[int]
  Get the descriptor type of this model.

get_dim_aparam() -> int
  Get the number (dimension) of atomic parameters of this DP.

get_dim_fparam() -> int
  Get the number (dimension) of frame parameters of this DP.

get_ntypes() -> int
  Get the number of atom types of this model.

get_ntypes_spin()
  Get the number of spin atom types of this model.

get_rcut() -> float
  Get the cut-off radius of this model.

get_sel_type() -> List[int]
  Unsupported in this model.

get_type_map() -> List[str]
  Get the type map (element name of the atom types) of this model.

load_prefix: str

deepmd.infer.deep_tensor module

class deepmd.infer.deep_tensor.DeepTensor(model_file: Path, load_prefix: str = 'load',
                                          default_tf_graph: bool = False, input_map: Optional[dict] = None, neighbor_list=None)

  Bases: DeepEval

  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.

  Attributes
model_type
Get type of model.

model_version
Get version of model.

sess
Get TF session.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build_neighbor_list(coords, cell, atype, ...)</td>
<td>Make the mesh with neighbor list for a single frame.</td>
</tr>
<tr>
<td>eval(coords, cells, atom_types[, atomic, ...])</td>
<td>Evaluate the model.</td>
</tr>
<tr>
<td>eval_full(coords, cells, atom_types[, ...])</td>
<td>Evaluate the model with interface similar to the energy model.</td>
</tr>
<tr>
<td>eval_typeebd()</td>
<td>Evaluate output of type embedding network by using this model.</td>
</tr>
<tr>
<td>get_dim_aparam()</td>
<td>Get the number (dimension) of atomic parameters of this DP.</td>
</tr>
<tr>
<td>get_dim_fparam()</td>
<td>Get the number (dimension) of frame parameters of this DP.</td>
</tr>
<tr>
<td>get_ntypes()</td>
<td>Get the number of atom types of this model.</td>
</tr>
<tr>
<td>get_rcut()</td>
<td>Get the cut-off radius of this model.</td>
</tr>
<tr>
<td>get_sel_type()</td>
<td>Get the selected atom types of this model.</td>
</tr>
<tr>
<td>get_type_map()</td>
<td>Get the type map (element name of the atom types) of this model.</td>
</tr>
<tr>
<td>make_natoms_vec(atom_types[, mixed_type])</td>
<td>Make the natom vector used by deepmd-kit.</td>
</tr>
<tr>
<td>reverse_map(vec, imap)</td>
<td>Reverse mapping of a vector according to the index map.</td>
</tr>
<tr>
<td>sort_input(coord, atom_type[, sel_atoms, ...])</td>
<td>Sort atoms in the system according their types.</td>
</tr>
</tbody>
</table>


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 sys-
    tem 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: ndarray, cells: ndarray, atom_types: List[int], atomic: bool = False, fparam:
    Optional[array] = None, aparam: Optional[array] = None, efield: Optional[array] =
    None, mixed_type: bool = False) → Tuple[ndarray, ...]

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 sys-
    tem 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]

get_dim_aparam() → int
Get the number (dimension) of atomic parameters of this DP.

get_dim_fparam() → int
Get the number (dimension) of frame parameters of this DP.

get_ntypes() → int
Get the number of atom types of this model.

get_rcut() → float
Get the cut-off radius of this model.

get_sel_type() → List[int]
Get the selected atom types of this model.

get_type_map() → List[str]
Get the type map (element name of the atom types) of this model.

tensors: ClassVar[Dict[str, str]] = {'t_box': 't_box:0', 't_coord': 't_coord:0',
't_mesh': 't_mesh:0', 't_natoms': 't_natoms:0', 't_ntypes':
'descriptor_attr/ntypes:0', 't_output_dim': 'model_attr/output_dim:0', 't_rcut':
descriptor_attr/rcut:0', 't_sel_type': 'model_attr/sel_type:0', 't_tmap':
'model_attr/tmap:0', 't_type': 't_type:0')

deepmd.infer.deep_wfc module

class deepmd.infer.deep_wfc.DeepWFC(model_file: Path, load_prefix: str = 'load',
default_tf_graph: bool = False, input_map: Optional[dict] = None)

Bases: DeepTensor

Constructor.

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

input_map
[dict, optional] The input map for tf.import_graph_def. Only work with default
tf graph

Warning: For developers: DeepTensor initializer must be called at the end after self.tensors are
modified because it uses the data in self.tensors dict. Do not change the order!
Attributes

- **model_type**
  Get type of model.

- **model_version**
  Get version of model.

- **sess**
  Get TF session.

Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build_neighbor_list(coords, cell, atype, ...)</code></td>
<td>Make the mesh with neighbor list for a single frame.</td>
</tr>
<tr>
<td><code>eval(coords, cells, atom_types[, atomic, ...])</code></td>
<td>Evaluate the model.</td>
</tr>
<tr>
<td><code>eval_full(coords, cells, atom_types[, ...])</code></td>
<td>Evaluate the model with interface similar to the energy model.</td>
</tr>
<tr>
<td><code>eval_typeebd()</code></td>
<td>Evaluate output of type embedding network by using this model.</td>
</tr>
<tr>
<td><code>get_dim_aparam()</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_dim_fparam()</code></td>
<td>Unsupported in this model.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Get the number of atom types of this model.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get the cut-off radius of this model.</td>
</tr>
<tr>
<td><code>get_sel_type()</code></td>
<td>Get the selected atom types of this model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map (element name of the atom types) of this model.</td>
</tr>
<tr>
<td><code>make_natoms_vec(atom_types[, mixed_type])</code></td>
<td>Make the natom vector used by deepmd-kit.</td>
</tr>
<tr>
<td><code>reverse_map(vec, imap)</code></td>
<td>Reverse mapping of a vector according to the index map.</td>
</tr>
<tr>
<td><code>sort_input(coord, atom_type[, sel_atoms, ...])</code></td>
<td>Sort atoms in the system according their types.</td>
</tr>
</tbody>
</table>

- **get_dim_aparam()** → int
  Unsupported in this model.

- **get_dim_fparam()** → int
  Unsupported in this model.

- **load_prefix**: str

**deepmd.infer.ewald_recp module**

- **class** `deepmd.infer.ewald_recp.EwaldRecp(hh, beta)`
  Bases: object
  Evaluate the reciprocal part of the Ewald sum.
Methods

**eval**(coord, charge, box)  Evaluate.

$\textbf{eval}(\text{coord: ndarray, charge: ndarray, box: ndarray}) \rightarrow \text{Tuple[ndarray, ndarray, ndarray]}$

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.infer.model_devi module**

$\textbf{deepmd.infer.model_devi.calc_model_devi}(\text{coord, box, atype, models, fname=None, frequency=1, mixed_type=False, fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, real_data: Optional[dict] = None, atomic: bool = False, relative: Optional[float] = None, relative_v: Optional[float] = None})$

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

```python
>>> 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 = [1,1,1]
>>> graphs = [DP("graph.000.pb"), DP("graph.001.pb")]
>>> model_devi = calc_model_devi(coord, cell, atype, graphs)
```

depdm.infer.model_devi.calc_model_devi_e(es: ndarray, real_e: Optional[ndarray] = None) →
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

depmd.infer.model_devi.calc_model_devi_f(fs: ndarray, real_f: Optional[ndarray] = None, relative: Optional[float] = None, *, atomic: Literal[True]) → Tuple[ndarray, ndarray, ndarray, 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

depmd.infer.model_devi.calc_model_devi_v(vs: ndarray, real_v: Optional[ndarray] = None, relative: Optional[float] = None) → Tuple[ndarray, ndarray, 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
[\text{numpy.ndarray}] maximum deviation of virial in 9 elements

min_devi_v
[\text{numpy.ndarray}] minimum deviation of virial in 9 elements

avg_devi_v
[\text{numpy.ndarray}] average deviation of virial in 9 elements


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

set_prefix
[str] The set prefix of the system

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 parameter 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.infer.model_devi.write_model_devi_out(devi: ndarray, fname: str, header: str = '', atomic: bool = False)

Write output of model deviation.

Parameters
devi
[\text{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.loggers package

Alias of deepmd-utils.loggers for backward compatibility.

deepmd.loggers.set_log_handles(level: int, log_path: Optional[Path] = None, mpi_log: Optional[str] = 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:

<table>
<thead>
<tr>
<th></th>
<th>our notation</th>
<th>python logging</th>
<th>tensorflow cpp</th>
<th>OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>1/on/true/yes</td>
</tr>
<tr>
<td>info</td>
<td>20</td>
<td>20</td>
<td>1</td>
<td>0/off/false/no</td>
</tr>
<tr>
<td>warning</td>
<td>30</td>
<td>30</td>
<td>2</td>
<td>0/off/false/no</td>
</tr>
<tr>
<td>error</td>
<td>40</td>
<td>40</td>
<td>3</td>
<td>0/off/false/no</td>
</tr>
</tbody>
</table>
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

Submodules

deepmd.loggers.loggers module

Alias of deepmd-utils.loggers.loggers for backward compatibility.

deepmd.loggers.loggers.set_log_handles(level: int, log_path: Optional[Path] = None, mpi_log: Optional[str] = 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:

<table>
<thead>
<tr>
<th></th>
<th>our notation</th>
<th>python logging</th>
<th>tensorflow cpp</th>
<th>OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>1/on/true/yes</td>
</tr>
<tr>
<td>info</td>
<td>20</td>
<td>20</td>
<td>1</td>
<td>0/off/false/no</td>
</tr>
<tr>
<td>warning</td>
<td>30</td>
<td>30</td>
<td>2</td>
<td>0/off/false/no</td>
</tr>
<tr>
<td>error</td>
<td>40</td>
<td>40</td>
<td>3</td>
<td>0/off/false/no</td>
</tr>
</tbody>
</table>
## References

- [DeepMD-kit](https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U)
- [Avoid TensorFlow print on standard error](https://stackoverflow.com/questions/35869137/)
- [Suppress OpenMP debug messages when running TensorFlow on CPU](https://stackoverflow.com/questions/56085015/)

## deepmd.loss package

```python
class deepmd.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, start_pref_cdf: float = 1e-08, log_fit: bool = False, **kwargs):
    Bases: Loss
    Loss function for DeepDOS models.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build(...)</td>
<td>Build the loss function graph.</td>
</tr>
<tr>
<td>display_if_exist(...)</td>
<td>Display NaN if labeled property is not found.</td>
</tr>
<tr>
<td>eval(...)</td>
<td>Eval the loss function.</td>
</tr>
</tbody>
</table>
```

### build(learning_rate, natoms, model_dict, ...) Build the loss function graph.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>learning_rate</td>
<td>[tf.Tensor] learning rate</td>
</tr>
<tr>
<td>natoms</td>
<td>[tf.Tensor] number of atoms</td>
</tr>
<tr>
<td>model_dict</td>
<td>[dict[str, tf.Tensor]] A dictionary that maps model keys to tensors</td>
</tr>
<tr>
<td>label_dict</td>
<td>[dict[str, tf.Tensor]] A dictionary that maps label keys to tensors</td>
</tr>
<tr>
<td>suffix</td>
<td>[str] suffix</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tf.Tensor</td>
<td>the total squared loss</td>
</tr>
<tr>
<td>dict[str, tf.Tensor]</td>
<td>A dictionary that maps loss keys to more loss tensors</td>
</tr>
</tbody>
</table>

### 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.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: Loss

Methods

- **build** (learning_rate, natoms, model_dict, ...) Build the loss function graph.
- **display_if_exist** (loss, find_property) Display NaN if labeled property is not found.
- **eval** (sess, feed_dict, natoms) Eval 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.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, relative_f: Optional[float] = None, enable_atom_ener_coeff: bool = False, use_spin: Optional[list] = None)

Bases: Loss

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>Build the loss function graph.</td>
</tr>
<tr>
<td>display_if_exist</td>
<td>Display NaN if labeled property is not found.</td>
</tr>
<tr>
<td>eval</td>
<td>Eval the loss function.</td>
</tr>
</tbody>
</table>

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

tf.Session] TensorFlow session

feed_dict
A dictionary that maps graph elements to values

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.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_auc: float
= 0.0, limit_pref_auc: float = 0.0, start_pref_pf: float = 0.0,
limit_pref_pf: float = 0.0, relative_f: Optional[float] = 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: Loss

Standard loss function for DP models.

Parameters

tensorflow.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_auc
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.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>build</strong>(learning_rate, natoms, model_dict, ...)</td>
<td>Build the loss function graph.</td>
</tr>
<tr>
<td><strong>display_if_exist</strong>(loss, find_property)</td>
<td>Display NaN if labeled property is not found.</td>
</tr>
<tr>
<td><strong>eval</strong>(sess, feed_dict, natoms)</td>
<td>Eval the loss function.</td>
</tr>
</tbody>
</table>

**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[\text{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.loss.TensorLoss(jdata, **kwargs)

Bases: Loss

Loss function for tensorial properties.

Methods

\begin{align*}
\text{build}(\text{learning_rate, natoms, model_dict, ...}) & \quad \text{Build the loss function graph.} \\
\text{display_if_exist}(\text{loss, find_property}) & \quad \text{Display NaN if labeled property is not found.} \\
\text{eval}(\text{sess, feed_dict, natoms}) & \quad \text{Eval the loss function.}
\end{align*}

\text{build}(\text{learning_rate, natoms, model_dict, label_dict, suffix})
Build the loss function graph.

Parameters

\begin{align*}
\text{learning_rate} & \quad [\text{tf.Tensor}] \text{ learning rate} \\
natoms & \quad [\text{tf.Tensor}] \text{ number of atoms} \\
\text{model_dict} & \quad [\text{dict[tf.placeholder, tf.Tensor]}] \text{ A dictionary that maps model keys to tensors} \\
\text{label_dict} & \quad [\text{dict[tf.placeholder, tf.Tensor]}] \text{ A dictionary that maps label keys to tensors} \\
suffix & \quad [\text{str}] \text{ suffix}
\end{align*}

Returns

\begin{align*}
\text{tf.Tensor} & \quad \text{the total squared loss} \\
dict[\text{str, tf.Tensor}] & \quad \text{A dictionary that maps loss keys to more loss tensors}
\end{align*}
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

Submodules

deepmd.loss.dos module

class deepmd.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: Loss

Loss function for DeepDOS models.

Methods

build(learning_rate, natoms, model_dict, ...) Build the loss function graph.
display_if_exist(loss, find_property) Display NaN if labeled property is not found.
eval(sess, feed_dict, natoms) Evaul 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.loss.ener module

class deepmd.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: Loss

Methods

build(learning_rate, natoms, model_dict, ...)
Build the loss function graph.
display_if_exist(loss, find_property)
Display NaN if labeled property is not found.
eval(sess, feed_dict, natoms)
Eval 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
DeePMD-kit

```
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.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_ac: float = 0.0,
    limit_pref_ac: float = 0.0, start_pref_pf: float = 0.0,
    limit_pref_pf: float = 0.0, relative_f: Optional[float] = None,
    enable_atom_ener_coeff: bool = False, use_spin: Optional[list] = None)

Bases: Loss

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>Build the loss function graph.</td>
</tr>
<tr>
<td>display_if_exist</td>
<td>Display NaN if labeled property is not found.</td>
</tr>
<tr>
<td>eval</td>
<td>Eval the loss function.</td>
</tr>
</tbody>
</table>

```

**build**(*learning_rate*, *natoms*, *model_dict*, *label_dict*, *suffix*)

Build the loss function graph.

**Parameters**

- **learning_rate**
  - `[tf.Tensor]` learning rate
- **natom**
  - `[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*, *natom*)

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
- **natom**
  - `[tf.Tensor]` number of atoms

**Returns**

- `[dict]` A dictionary that maps keys to values. It should contain key natom

**print_header()**

**print_on_training**(tb_writer, *cur_batch*, *sess*, *natom*, *feed_dict_test*, *feed_dict_batch*)

**class deepmd.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: Optional[float] = 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**: 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.
## Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build</code></td>
<td>Build the loss function graph.</td>
</tr>
<tr>
<td><code>display_if_exist</code></td>
<td>Display NaN if labeled property is not found.</td>
</tr>
<tr>
<td><code>eval</code></td>
<td>Eval the loss function.</td>
</tr>
</tbody>
</table>

### build(learning_rate, natoms, model_dict, ...)  
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.loss.loss.Loss

Bases: object

The abstract class for the loss function.

Methods


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

static display_if_exist(loss: Tensor, find_property: float) → Tensor

Display NaN if labeled property is not found.

Parameters

- loss
  - [tf.Tensor] the loss tensor
- find_property
  - [float] whether the property is found

abstract eval(sess: Session, feed_dict: Dict[placeholder, Tensor], natoms: 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

deepmd.loss.tensor module

class deepmd.loss.tensor.TensorLoss(jdata, **kwarg)

Bases: Loss

Loss function for tensorial properties.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>Build the loss function graph.</td>
</tr>
<tr>
<td>display_if_exist</td>
<td>Display NaN if labeled property is not found.</td>
</tr>
<tr>
<td>eval</td>
<td>Eval the loss function.</td>
</tr>
</tbody>
</table>

build(learning_rate, natoms, model_dict, ...) 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

18.2. deepmd package
```python
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.model package

class deepmd.model.DOSModel(*args, **kwargs)
Bases: 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
```
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box, mesh, ...)</code></td>
<td>Build the model.</td>
</tr>
<tr>
<td><code>build_descrpt(coord_, atype_, natoms, box, ...)</code></td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td><code>build_type_embedding(ntypes[, frz_model, ...])</code></td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td><code>change_energy_bias(data, frozen_model, ...)</code></td>
<td>Change the energy bias according to the input data</td>
</tr>
<tr>
<td><code>data_stat(data)</code></td>
<td>Data statics.</td>
</tr>
<tr>
<td><code>enable_compression([suffix])</code></td>
<td>Enable compression.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td><code>get_class_by_input(input)</code></td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td><code>get_feed_dict(coord_, atype_, natoms, box, ...)</code></td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td><code>get_fitting()</code></td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td><code>get_numb_aparam()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get_numb_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get_numb_fparam()</code></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get the cutoff radius of the model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, ...])</code></td>
<td>Init the embedding net variables with the given</td>
</tr>
<tr>
<td></td>
<td>frozen model.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor</td>
</tr>
<tr>
<td></td>
<td>statistics.</td>
</tr>
</tbody>
</table>

**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

---

18.2. **deepmd package** 435
DeePMD-kit

reuse
    [bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns
    dict
    The output dict

data_stat(data)
    Data staticis.

get_ntypes()
    Get the number of types.

get_numb_aparam()  \rightarrow  int
    Get the number of atomic parameters.

get_numb_dos()
    Get the number of gridpoints in energy space.

get_numb_fparam()  \rightarrow  int
    Get the number of frame parameters.

get_rcut()
    Get cutoff radius of the model.

get_type_map()
    Get the type map.

init_variables(graph: Graph, graph_def: GraphDef, model_type: str = 'original_model', suffix: str = '')  \rightarrow  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

model_type = 'dos'

class deepmd.model.DipoleModel(*args, **kwargs)
    Bases: TensorModel
## Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build(coord_, atype_, natoms, box, mesh,...)</td>
<td>Build the model.</td>
</tr>
<tr>
<td>build_descrpt(coord_, atype_, natoms, box,...)</td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td>build_type_embedding(ntypes[, frz_model,...])</td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td>change_energy_bias(data, frozen_model,...)</td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td>data_stat(data)</td>
<td>Data statics.</td>
</tr>
<tr>
<td>enable_compression([suffix])</td>
<td>Enable compression.</td>
</tr>
<tr>
<td>enable_mixed_precision(mixed_prec)</td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td>get_class_by_input(input)</td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td>get_feed_dict(coord_, atype_, natoms, box,...)</td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td>get_fitting()</td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td>get_loss(loss, lr)</td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td>get_numb_aparam()</td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td>get_numb_dos()</td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td>get_numb_fparam()</td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td>get_rcut()</td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td>get_type_map()</td>
<td>Get the type map.</td>
</tr>
<tr>
<td>init_variables(graph, graph_def[, ...])</td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td>update_sel(global_jdata, local_jdata)</td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

### Class: deepmd.model.EnerModel(*args, **kwargs)

**Bases:** 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_nbath
  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 \((N_t + 1) \times N_t / 2 + 1\) columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

**smin** alpha

The short-range tabulated interaction will be switched 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**

**data_stat_nsample**

The number of training samples in a system to compute and change the energy bias.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>build(coord, atype, natoms, box, mesh, ...)</strong></td>
<td>Build the model.</td>
</tr>
<tr>
<td><strong>build_descrpt(coord, atype, natoms, box, ...)</strong></td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td><strong>build_type_embedding(ntypes[, frz_model, ...])</strong></td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td><strong>change_energy_bias(data, frozen_model, ...)</strong></td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td><strong>data_stat(data)</strong></td>
<td>Data statics.</td>
</tr>
<tr>
<td><strong>enable_compression([suffix])</strong></td>
<td>Enable compression.</td>
</tr>
<tr>
<td><strong>enable_mixed_precision(mixed_prec)</strong></td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td><strong>get_class_by_input(input)</strong></td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td><strong>get_feed_dict(coord, atype, natoms, box, ...)</strong></td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td><strong>get_fitting()</strong></td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td><strong>get_loss(loss, lr)</strong></td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td><strong>get_ntypes()</strong></td>
<td>Get the number of types.</td>
</tr>
<tr>
<td><strong>get_num_aparam()</strong></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><strong>get_num_dos()</strong></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><strong>get_num_fpamam()</strong></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><strong>get_rcut()</strong></td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td><strong>get_type_map()</strong></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><strong>init_variables(graph, graph_def[, ...])</strong></td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td><strong>update_sel(global_jdata, local_jdata)</strong></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>
build(coord_, atype_, natoms, box, mesh, input_dict, frz_model=None, ckpt_meta: Optional[str] = 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

change_energy_bias(data: DeepmdDataSystem, frozen_model: str, origin_type_map: list,
full_type_map: str, bias_shift: str = 'delta') → 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_shift

[str] The mode for changing energy bias: ['delta', 'statistic']
'delta': perform predictions on energies of target dataset,
and do least square on the errors to obtain the target shift as bias.
'statistic': directly use the statistic energy bias in the target dataset.

data_stat(data)
Data statics.

get_ntypes()
Get the number of types.

get_numb_aparam() \rightarrow \text{int}
Get the number of atomic parameters.

get_numb_fparam() \rightarrow \text{int}
Get the number of frame parameters.

get_rcut()
Get cutoff radius of the model.

get_type_map()
Get the type map.

init_variables(graph: Graph, graph_def: GraphDef, model_type: str = 'original_model', suffix: str = '') \rightarrow 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

model_type = 'ener'

natoms_match(force, natoms)
natoms_not_match(force, natoms, atype)

class deepmd.model.GlobalPolarModel(*args, **kwargs)
Bases: TensorModel
### Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build(coord, atype, natoms, box, mesh)</td>
<td>Build the model.</td>
</tr>
<tr>
<td>build_descrpt(coord, atype, natoms, box)</td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td>build_type_embedding(types[, frz_model])</td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td>change_energy_bias(data, frozen_model)</td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td>data_stat(data)</td>
<td>Data staticis.</td>
</tr>
<tr>
<td>enable_compression([suffix])</td>
<td>Enable compression.</td>
</tr>
<tr>
<td>enable_mixed_precision(mixed_prec)</td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td>get_class_by_input(input)</td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td>get_feed_dict(coord, atype, natoms, box)</td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td>get_fitting()</td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td>get_loss(loss, lr)</td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td>get_ntypes()</td>
<td>Get the number of types.</td>
</tr>
<tr>
<td>get_numb_aparam()</td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td>get_numb_dos()</td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td>get_numb_fparam()</td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td>get_rcut()</td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td>get_type_map()</td>
<td>Get the type map.</td>
</tr>
<tr>
<td>init_variables(graph, graph_def[, ...])</td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td>update_sel(global_jdata, local_jdata)</td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```python

def get_out_size() -> None:
    pass

def get_sel_type() -> None:
    pass
```

class deepmd.model.MultiModel(*args, **kwargs):
    Bases: 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 \((N_t + 1) \times N_t / 2 + 1\) columns. The first column is the distance
between atoms. The second to the last columns are energies for pairs of certain types.
For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for
0-0, 0-1 and 1-1 correspondingly.

smin_alpha
The short-range tabulated interaction will be switched 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_rmax
The upper boundary of the interpolation between short-range tabulated interaction
and DP. It is only required when use_srtab is provided.

Methods

```
build(coord, atype, natoms, box, mesh, ...) Build the model.
build_descript(coord, atype, natoms, box, ...) Build the descriptor part of the model.
build_type_embedding(ntypes[, frz_model, ...]) Build the type embedding part of the model.
change_energy_bias(data, frozen_model, ...) Change the energy bias according to the input
data and the pretrained model.
data_stat(data) Data staticis.
enable_compression(suffix) Enable compression.
enable_mixed_precision(mixed_prec) Enable mixed precision for the model.
get_class_by_input(input) Get the class by input data.
get_feed_dict(coord, atype, natoms, box, ...) Generate the feed_dict for current descriptor.
get_fitting() Get the fitting(s).
get_loss(loss, lr) Get the loss function(s).
get_ntypes() Get the number of types.
get_numb_aparam() Get the number of atomic parameters.
get_numb_dos() Get the number of gridpoints in energy space.
get_numb_fparam() Get the number of frame parameters.
get_rcut() Get cutoff radius of the model.
get_type_map() Get the type map.
init_variables(graph, graph_def[, ...]) Init the embedding net variables with the given
frozen model.
update_sel(global_jdata, local_jdata) Update the selection and perform neighbor
statistics.
```

```
build(coord, atype, natoms, box, mesh, input_dict, frz_model=None, ckpt_meta: Optional[str] =
None, suffix='', reuse=None)
Build the model.

Parameters

coord
    [tf.Tensor] The coordinates of atoms
```
The atom types of atoms

The number of atoms

The box vectors

The mesh vectors

The input dict

The path to the frozen model

The path prefix of the checkpoint and meta files

The suffix of the scope

Whether to reuse the variables

The output dict

Data staticis.

Enable mixed precision for the model.

The mixed precision config

Get the fitting(s).

Get the loss function(s).

Get the number of types.

Get the number of atomic parameters.

Get the number of gridpoints in energy space.

Get the number of frame parameters.
get_rcut()
Get cutoff radius of the model.

get_type_map()
Get the type map.

init_variables(graph: Graph, graph_def: 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

model_type = 'multi_task'

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.model.PolarModel(*args, **kwargs)
Bases: TensorModel
Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box, mesh, ...)</code></td>
<td>Build the model.</td>
</tr>
<tr>
<td><code>build_descrpt(coord_, atype_, natoms, box, ...)</code></td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td><code>build_type_embedding(ntypes[, frz_model, ...])</code></td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td><code>change_energy_bias(data, frozen_model, ...)</code></td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td><code>data_stat(data)</code></td>
<td>Data statistics.</td>
</tr>
<tr>
<td><code>enable_compression([suffix])</code></td>
<td>Enable compression.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td><code>get_class_by_input(input)</code></td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td><code>get_feed_dict(coord_, atype_, natoms, box, ...)</code></td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td><code>get_fitting()</code></td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Get the number of types.</td>
</tr>
<tr>
<td><code>get_numb_aparam()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get_numb_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get_numb_fparam()</code></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, ...])</code></td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```
class deepmd.model.WFCModel(*args, **kwargs)
```

Bases: `TensorModel`
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box, mesh, ...)</code></td>
<td>Build the model.</td>
</tr>
<tr>
<td><code>build_descrpt(coord_, atype_, natoms, box, ...)</code></td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td><code>build_type_embedding(ntypes[, frz_model, ...])</code></td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td><code>change_energy_bias(data, frozen_model, ...)</code></td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td><code>data_stat(data)</code></td>
<td>Data statics.</td>
</tr>
<tr>
<td><code>enable_compression([suffix])</code></td>
<td>Enable compression.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td><code>get_class_by_input(input)</code></td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td><code>get_feed_dict(coord_, atype_, natoms, box, ...)</code></td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td><code>get_fitting()</code></td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Get the number of types.</td>
</tr>
<tr>
<td><code>get_numb_aparam()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get_numb_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get_numb_fparam()</code></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, ...])</code></td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

Submodules

deepmd.model.dos module

class deepmd.model.dos.DOSModel(*args, **kwargs)

Bases: StandardModel

DOS model.

Parameters

descriptor
descriptor

fitting_net

Fit 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 statistic parameters

- **data_stat_nbatch**: Number of frames used for data statistic
- **data_stat_protect**: Protect parameter for atomic energy regression

### Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box, mesh,...)</code></td>
<td>Build the model.</td>
</tr>
<tr>
<td><code>build_descrpt(coord_, atype_, natoms, box,...)</code></td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td><code>build_type_embedding(ntypes[, frz_model, ...])</code></td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td><code>change_energy_bias(data, frozen_model, ...)</code></td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td><code>data_stat(data)</code></td>
<td>Data statistics.</td>
</tr>
<tr>
<td><code>enable_compression([suffix])</code></td>
<td>Enable compression.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td><code>get_class_by_input(input)</code></td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td><code>get_feed_dict(coord_, atype_, natoms, box,...)</code></td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td><code>get_fitting()</code></td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Get the number of types.</td>
</tr>
<tr>
<td><code>get_numb_aparam()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get_numb_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get_numb_fparam()</code></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, ...])</code></td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

#### 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
DeePMD-kit

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

data_stat(data)
    Data staticis.

get_natypes()
    Get the number of types.

get_numb_aparam() \rightarrow int
    Get the number of atomic parameters.

get_numb_dos()
    Get the number of gridpoints in energy space.

get_numb_fparam() \rightarrow int
    Get the number of frame parameters.

get_rcut()
    Get cutoff radius of the model.

get_type_map()
    Get the type map.

init_variables(graph: Graph, graph_def: GraphDef, model_type: str = 'original_model', suffix: str = '') \rightarrow 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

model_type = 'dos'
**deepmd.model.ener module**

```python
class deepmd.model.ener.EnerModel(*args, **kwargs)
    Bases: 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 columns. The first column is the distance
        between atoms. The second to the last columns are energies for pairs of certain types.
        For example we have two atom types, 0 and 1. The columns 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 tabu-
        lated 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.
```
### Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box, mesh, ...)</code></td>
<td>Build the model.</td>
</tr>
<tr>
<td><code>build_descrpt(coord_, atype_, natoms, box, ...)</code></td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td><code>build_type_embedding(ntypes[, frz_model, ...])</code></td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td><code>change_energy_bias(data, frozen_model, ...)</code></td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td><code>data_stat(data)</code></td>
<td>Data statics.</td>
</tr>
<tr>
<td><code>enable_compression([suffix])</code></td>
<td>Enable compression.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td><code>get_class_by_input(input)</code></td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td><code>get_feed_dict(coord_, atype_, natoms, box, ...)</code></td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td><code>get_fitting()</code></td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Get the number of types.</td>
</tr>
<tr>
<td><code>get numb_aparam()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get numb_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get numb fparam()</code></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, ...])</code></td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```python
build(coord_, atype_, natoms, box, mesh, ..., frz_model=None, ckpt_meta: Optional[str] = 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
```python
ckp_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

dict

change_energy_bias(data: DeepmdDataSystem, frozen_model: str,
    origin_type_map: list,
    full_type_map: str, bias_shift: str = ‘delta’) → 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_shift
        [str] The mode for changing energy bias : [‘delta’, ‘statistic’] ‘delta’ : perform
predictions on energies of target dataset,
        and do least square on the errors to obtain the target shift as bias.
        ‘statistic’ : directly use the statistic energy bias in the target dataset.

data_stat(data)
    Data statics.

get_ntypes()
    Get the number of types.

get_numb_aparam() → int
    Get the number of atomic parameters.

get_numb_fparam() → int
    Get the number of frame parameters.

get_rcut()
    Get cutoff radius of the model.

get_type_map()
    Get the type map.

init_variables(graph: Graph, graph_def: GraphDef, model_type: str = ‘original_model’, suffix: str = “”) → None
    Init the embedding net variables with the given frozen model.
```

18.2. deepmd package
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

```python
model_type = 'ener'
natoms_match(force, natoms)
natoms_not_match(force, natoms, atype)
```

**deepmd.model.frozen module**

class deepmd.model.frozen.FrozenModel(*args, **kwargs)

Bases: Model

Load model from a frozen model, which cannot be trained.

Parameters

- model_file
  - [str] The path to the frozen model
### Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box, mesh, ...)</code></td>
<td>Build the model.</td>
</tr>
<tr>
<td><code>build_descriptor(coord_, atype_, natoms, box, ...)</code></td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td><code>build_type_embedding(ntypes[, frz_model, ...])</code></td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td><code>change_energy_bias(data, frozen_model, ...)</code></td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td><code>data_stat(data)</code></td>
<td>Data statics.</td>
</tr>
<tr>
<td><code>enable_compression([suffix])</code></td>
<td>Enable compression.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td><code>get_class_by_input(input)</code></td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td><code>get_feed_dict(coord_, atype_, natoms, box, ...)</code></td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td><code>get_fitting()</code></td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Get the number of types.</td>
</tr>
<tr>
<td><code>get_num_aparam()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get_num_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get_num_fparam()</code></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, ...])</code></td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>


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
DeePMD-kit

import tensorflow as tf

suffix

[\texttt{str}, \texttt{optional}] The suffix of the scope

reuse

[\texttt{bool} or \texttt{tf.AUTO_REUSE, optional}] Whether to reuse the variables

Returns

dict

The output dict
data_stat(data)

Data staticis.

\texttt{enable_compression(suffix: str = '')} $\rightarrow$ \texttt{None}

Enable compression.

Parameters

suffix

[\texttt{str}] suffix to name scope

get_fitting() $\rightarrow$ Union[\texttt{Fitting, dict}]

Get the fitting(s).

get_loss(loss: dict, lr) $\rightarrow$ Optional[Union[\texttt{Loss, dict}]]

Get the loss function(s).

get_ntypes() $\rightarrow$ int

Get the number of types.

get_rcut()

Get cutoff radius of the model.

get_type_map() $\rightarrow$ list

Get the type map.

\texttt{init_variables(graph: Graph, graph_def: GraphDef, model_type: str = 'original_model', suffix: str = '')} $\rightarrow$ \texttt{None}

Init the embedding net variables with the given frozen model.

Parameters

graph

[\texttt{tf.Graph}] The input frozen model graph

graph_def

[\texttt{tf.GraphDef}] The input frozen model graph_def

model_type

[\texttt{str}] the type of the model

suffix

[\texttt{str}] suffix to name scope

classmethod update_sel(global_jdata: dict, local_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global_jdata

[\texttt{dict}] The global data, containing the training section
deepmd.model.linear module

class deepmd.model.linear.LinearEnergyModel(*args, **kwargs)
    Bases: LinearModel

Linear energy model make linear combinations of several existing energy models.

Methods

`build(coord_, atype_, natoms, box, mesh, ...)` Build the model.

`build_descrpt(coord_, atype_, natoms, box, ...)` Build the descriptor part of the model.

`build_type_embedding(ntypes[, frz_model, ...])` Build the type embedding part of the model.

`change_energy_bias(data, frozen_model, ...)` Change the energy bias according to the input data and the pretrained model.

`data_stat(data)` Data statics.

`enable_compression([suffix])` Enable compression.

`enable_mixed_precision(mixed_prec)` Enable mixed precision for the model.

`get_class_by_input(input)` Get the class by input data.

`get_feed_dict(coord_, atype_, natoms, box, ...)` Generate the feed_dict for current descriptor.

`get_fitting()` Get the fitting(s).

`get_loss(loss, lr)` Get the loss function(s).

`get_numb_aparam()` Get the number of atomic parameters.

`get_numb_dos()` Get the number of gridpoints in energy space.

`get_numb_fparam()` Get the number of frame parameters.

`get_rcut()` Get cutoff radius of the model.

`get_type_map()` Get the type map.

`init_variables(graph, graph_def[, ...])` Init the embedding net variables with the given frozen model.

`update_sel(global_jdata, local_jdata)` Update the selection and perform neighbor statistics.

`get_ntypes`
DeePMD-kit

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

model_type = 'ener'

class deepmd.model.linear.LinearModel(*args, **kwargs)

    Bases: 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.
### Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box, mesh,...)</code></td>
<td>Build the model.</td>
</tr>
<tr>
<td><code>build_descrpt(coord_, atype_, natoms, box,...)</code></td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td><code>build_type_embedding(types[, frz_model,...])</code></td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td><code>change_energy_bias(data, frozen_model,...)</code></td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td><code>data_stat(data)</code></td>
<td>Data statistics.</td>
</tr>
<tr>
<td><code>enable_compression([suffix])</code></td>
<td>Enable compression.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td><code>get_class_by_input(input)</code></td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td><code>get_feed_dict(coord_, atype_, natoms, box,...)</code></td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td><code>get_fitting()</code></td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td><code>get_numb_aparam()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get_numb_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get_numb_fparam()</code></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, ...])</code></td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```python
get_ntypes()
```

data_stat(data)
Data statics.

```python
def enable_compression(suffix: str = '') → None
    Enable compression.
    Parameters
    suffix
        [str] suffix to name scope
```

```python
def get_fitting() → Union[Fitting, dict]
    Get the fitting(s).
```

```python
def get_loss(loss: dict, lr) → Optional[Union[Loss, dict]]
    Get the loss function(s).
```

```python
def get_ntypes() → int
    Get the number of types.
```

```python
def get_rcut()
    Get cutoff radius of the model.
```

```python
def get_type_map() → list
    Get the type map.
```
**init_variables**(graph: Graph, graph_def: 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

**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.model.model module**

**class deepmd.model.model.Model(***args, **kwargs)

Bases: ABC

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 \((N_t + 1) * N_t / 2 + 1\) columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.
smin_alpha
The short-range tabulated interaction will be switched 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

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>Build the model.</td>
</tr>
<tr>
<td>build_descrpt</td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td>build_type_embedding</td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td>change_energy_bias</td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td>data_stat</td>
<td>Data statistics.</td>
</tr>
<tr>
<td>enable_compression</td>
<td>Enable compression.</td>
</tr>
<tr>
<td>enable_mixed_precision</td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td>get_class_by_input</td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td>get_feed_dict</td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td>get_fitting</td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td>get_loss</td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td>get_ntypes</td>
<td>Get the number of types.</td>
</tr>
<tr>
<td>get_numb_aparam</td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td>get_numb_dos</td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td>get_numb_fparam</td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td>get_rcut</td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td>get_type_map</td>
<td>Get the type map.</td>
</tr>
<tr>
<td>init_variables</td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td>update_sel</td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>


Build the model.

Parameters
coord_  
  [tf.Tensor] The coordinates of atoms
attype_  
  [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

def build_descrpt(coord_: Tensor, atype_: Tensor, natoms: Tensor, box: Tensor, mesh: Tensor,  
  input_dict: dict, frz_model: Optional[str] = None, ckpt_meta: Optional[str] = None,  
  suffix: str = '', reuse: Optional[Union[bool, Enum]] = None)

Build the descriptor part of the model.

Parameters

coord_  
  [tf.Tensor] The coordinates of atoms
attype_  
  [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
build_type_embedding( ntypes: int, frz_model: Optional[str] = None, ckpt_meta: Optional[str] = None, suffix: str = '', reuse: Optional[Union[bool, Enum]] = None) → 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

ckp_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

change_energy_bias( data: DeepmdDataSystem, frozen_model: str, origin_type_map: list, full_type_map: str, bias_shift: str = 'delta') → 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_shift

[str] The mode for changing energy bias : ['delta', 'statistic'] 'delta': perform predictions on energies of target dataset, and do least square on the errors to obtain the target shift as bias.
‘statistic’ : directly use the statistic energy bias in the target dataset.

**abstract data_stat** *(data: dict)*
Data staticis.

**enable_compression** *(suffix: str = ‘’)*
Enable compression.

**Parameters**

**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

**classmethod get_class_by_input** *(input: dict)*
Get the class by input data.

**Parameters**

**input**
dict The input data

**get_feed_dict** *(coord_: Tensor, atype_: Tensor, natoms: Tensor, box: Tensor, mesh: Tensor, 
**kwargs) → Dict[str, 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.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[tf.Tensor] The output feed_dict of current descriptor
abstract get_fitting() → Union[Fitting, dict]
    Get the fitting(s).

abstract get_loss(loss: dict, lr) → Optional[Union[Loss, dict]]
    Get the loss function(s).

abstract get_ntypes() → int
    Get the number of types.

get_numb_aparam() → Union[int, dict]
    Get the number of atomic parameters.

get_numb_dos() → Union[int, dict]
    Get the number of gridpoints in energy space.

get_numb_fparam() → Union[int, dict]
    Get the number of frame parameters.

abstract get_rcut() → float
    Get cutoff radius of the model.

get_type_map() → list
    Get the type map.

init_variables(graph: Graph, graph_def: 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

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.

class deepmd.model.model.StandardModel(*args, **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

Methods

build(coord_, atype_, natoms, box, mesh, ...) Build the model.

build_descrpt(coord_, atype_, natoms, box, ...) Build the descriptor part of the model.

build_type_embedding(ntypes[, frz_model, ...]) Build the type embedding part of the model.

change_energy_bias(data, frozen_model, ...) Change the energy bias according to the input data and the pretrained model.

data_stat(data) Data statics.

enable_compression([suffix]) Enable compression.

enable_mixed_precision(mixed_prec) Enable mixed precision for the model.

get_class_by_input(input) Get the class by input data.

get_feed_dict(coord_, atype_, natoms, box, ...) Generate the feed_dict for current descriptor.

get_fitting() Get the fitting(s).

get_loss(loss, lr) Get the loss function(s).

get_ntypes() Get the number of types.

get_numb_aparam() Get the number of atomic parameters.

get_numb_dos() Get the number of gridpoints in energy space.

get_numb_fparam() Get the number of frame parameters.

get_rcut() Get cutoff radius of the model.

get_type_map() Get the type map.

init_variables(graph, graph_def[, ...]) Init the embedding net variables with the given frozen model.

update_sel(global_jdata, local_jdata) Update the selection and perform neighbor statistics.

enable_compression(suffix: str = '') Enable compression.

Parameters

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_fitting() → Union[Fitting, dict]
    Get the fitting(s).

get_loss(loss: dict, lr) → Union[Loss, dict]
    Get the loss function(s).

get_ntypes() → int
    Get the number of types.

get_rcut() → float
    Get cutoff radius of the model.

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.model.model_stat module

Alias for backward compatibility.

deepmd.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.model.model_stat.merge_sys_stat(all_stat)

**deepmd.model.multi module**

```python
class deepmd.model.multi.MultiModel(*args, **kwargs)
    Bases: 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 columns. The first column is the distance
        between atoms. The second to the last columns are energies for pairs of certain types.
        For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for
        0-0, 0-1 and 1-1 correspondingly.
    smin_alpha
        The short-range tabulated interaction will be switched 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.
```
## Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box, mesh, ...)</code></td>
<td>Build the model.</td>
</tr>
<tr>
<td><code>build_descrpt(coord_, atype_, natoms, box, ...)</code></td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td><code>build_type_embedding(ntypes[, frz_model, ...])</code></td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td><code>change_energy_bias(data, frozen_model, ...)</code></td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td><code>data_stat(data)</code></td>
<td>Data statics.</td>
</tr>
<tr>
<td><code>enable_compression([suffix])</code></td>
<td>Enable compression.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td><code>get_class_by_input(input)</code></td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td><code>get_feed_dict(coord_, atype_, natoms, box, ...)</code></td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td><code>get_fitting()</code></td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td><code>get_numb_aparam()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get_numb_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get_numb_fparam()</code></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, ...])</code></td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

### build

```python
build(coord_, atype_, natoms, box, mesh, input_dict, frz_model=None, ckpt_meta: Optional[str] = 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
DeePMD-kit

reuse
    [bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns
dict
    The output dict

data_stat(data)
    Data statics.

enable_mixed_precision(mixed_prec: dict)
    Enable mixed precision for the model.

Parameters
    mixed_prec
        [dict] The mixed precision config

get_fitting() → dict
    Get the fitting(s).

get_loss(loss: dict, lr: dict) → Dict[str, Loss]
    Get the loss function(s).

get_ntypes()
    Get the number of types.

get_numb_aparam() → dict
    Get the number of atomic parameters.

get_numb_dos() → dict
    Get the number of gridpoints in energy space.

get_numb_fparam() → dict
    Get the number of frame parameters.

get_rcut()
    Get cutoff radius of the model.

get_type_map()
    Get the type map.

init_variables(graph: Graph, graph_def: 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
model_type = 'multi_task'

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.model.pairtab module

class deepmd.model.pairtab.PairTabModel(*args, **kwargs)
    Bases: 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 maximum number of atoms in the cut-off radius
## Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box, mesh, ...)</code></td>
<td>Build the model.</td>
</tr>
<tr>
<td><code>build_descrpt(coord_, atype_, natoms, box, ...)</code></td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td><code>build_type_embedding(ntypes[, frz_model, ...])</code></td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td><code>change_energy_bias(data, frozen_model, ...)</code></td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td><code>data_stat(data)</code></td>
<td>Data statics.</td>
</tr>
<tr>
<td><code>enable_compression([suffix])</code></td>
<td>Enable compression.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td><code>get_class_by_input(input)</code></td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td><code>get_feed_dict(coord_, atype_, natoms, box, ...)</code></td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td><code>get_fitting()</code></td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td><code>get_numb_aparam()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get_numb_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get_numb_fparam()</code></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, ...])</code></td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

### 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`: `Optional[str] = None` - The path to the frozen model
- `ckpt_meta`: `Optional[str] = None` - The path prefix of the checkpoint and meta files
- `suffix`: `str = ''` - A suffix
- `reuse`: `Optional[Union[bool, Enum]] = None` - A reuse flag

Build the model.
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

data_stat(data: dict)
Data statics.

enable_compression(suffix: str = '') → None
Enable compression.

Parameters
    suffix
        [str] suffix to name scope

get_fitting() → Union[Fitting, dict]
Get the fitting(s).

get_loss(loss: dict, lr) → Optional[Union[Loss, dict]]
Get the loss function(s).

get_ntypes() → int
Get the number of types.

get_rcut() → float
Get cutoff radius of the model.

init_variables(graph: Graph, graph_def: 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

model_type = 'ener'

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
DeePMD-kit

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.model.pairwise_dprc module

class deepmd.model.pairwise_dprc.PairwiseDPRc(*args, **kwargs)

Bases: Model

Pairwise Deep Potential - Range Correction.

Methods

build(coord_, atype_, natoms, box_, mesh, ...) Build the model.
build_descrpt(coord_, atype_, natoms, box, ...) Build the descriptor part of the model.
build_type_embedding(utypes[, frz_model, ...]) Build the type embedding part of the model.
change_energy_bias(data, frozen_model, ...) Change the energy bias according to the input data and the pretrained model.
data_stat(data) Data statics.
enable_compression([suffix]) Enable compression.
enable_mixed_precision(mixed_prec) Enable mixed precision for the model.
get_class_by_input(input) Get the class by input data.
get_feed_dict(coord_, atype_, natoms, box, ...) Generate the feed_dict for current descriptor.
get_fitting() Get the fitting(s).
get_loss(loss, lr) Get the loss function(s).
get_ntypes() Get the number of types.
get_numb_aparam() Get the number of atomic parameters.
get_numb_dos() Get the number of gridpoints in energy space.
get_numb_fparam() Get the number of frame parameters.
get_rcut() Get cutoff radius of the model.
get_type_map() Get the type map.
init_variables(graph, graph_def[, ...]) Init the embedding net variables with the given frozen model.
update_sel(global_jdata, local_jdata) Update the selection and perform neighbor statistics.


Build the model.

Parameters
coord_  
  [tf.Tensor] The coordinates of atoms
atypex_  
  [tf.Tensor] The atom types of atoms
natomx_  
  [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

data_stat(data)
  Data staticis.

enable_compression(suffix: str = '') → None
  Enable compression.

  Parameters

  suffix
    [str] suffix to name scope

get_feed_dict(coord_: Tensor, atype_: Tensor, natoms: Tensor, box: Tensor, mesh: Tensor, **kwargs) → Dict[str, 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.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] The output feed_dict of current descriptor

get_fitting() → Union[str, dict]
Get the fitting(s).

get_loss(loss: dict, lr) → Union[Loss, dict]
Get the loss function(s).

get_ntypes() → int
Get the number of types.

get_rcut() 
Get cutoff radius of the model.

init_variables(graph: Graph, graph_def: 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

model_type = 'ener'

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.model.pairwise_dprc.gather_placeholder(params: Tensor, indices: Tensor, placeholder: float = 0.0, **kwargs) → Tensor

Call tf.gather but allow indices to contain placeholders (-1).

deepmd.model.tensor module

class deepmd.model.tensor.DipoleModel(*args, **kwargs)
Bases: TensorModel

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build(coord, atype, natoms, box, mesh, ...)</td>
<td>Build the model.</td>
</tr>
<tr>
<td>build_descript(coord, atype, natoms, box, ...)</td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td>build_type_embedding(utypes[, frz_model, ...])</td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td>change_energy_bias(data, frozen_model, ...)</td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td>data_stat(data)</td>
<td>Data statics.</td>
</tr>
<tr>
<td>enable_compression([suffix])</td>
<td>Enable compression.</td>
</tr>
<tr>
<td>enable_mixed_precision(mixed_prec)</td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td>get_class_by_input(input)</td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td>get_feed_dict(coord, atype, natoms, box, ...)</td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td>get_fitting()</td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td>get_loss(loss, lr)</td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td>get_ntypes()</td>
<td>Get the number of types.</td>
</tr>
<tr>
<td>get_numb_aparam()</td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td>get_numb_dos()</td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td>get_numb_fparam()</td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td>get_rcut()</td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td>get_type_map()</td>
<td>Get the type map.</td>
</tr>
<tr>
<td>init_variables(graph, graph_def[, ...])</td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td>update_sel(global_jdata, local_jdata)</td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

class deepmd.model.tensor.GlobalPolarModel(*args, **kwargs)
Bases: TensorModel

18.2. deepmd package
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box, mesh, ...)</code></td>
<td>Build the model.</td>
</tr>
<tr>
<td><code>build_descrpt(coord_, atype_, natoms, box, ...)</code></td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td><code>build_type_embedding(ntypes[, frz_model, ...])</code></td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td><code>change_energy_bias(data, frozen_model, ...)</code></td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td><code>data_stat(data)</code></td>
<td>Data staticis.</td>
</tr>
<tr>
<td><code>enable_compression([suffix])</code></td>
<td>Enable compression.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td><code>get_class_by_input(input)</code></td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td><code>get_feed_dict(coord_, atype_, natoms, box, ...)</code></td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td><code>get_fitting()</code></td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Get the number of types.</td>
</tr>
<tr>
<td><code>get_nbr_aparam()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get_nbr_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, ...])</code></td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

```python

class deepmd.model.tensor.PolarModel(*args, **kwargs)
Bases: TensorModel
```
## Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box, mesh,...)</code></td>
<td>Build the model.</td>
</tr>
<tr>
<td><code>build_descrpt(coord_, atype_, natoms, box,...)</code></td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td><code>build_type_embedding(ntypes[, frz_model,...])</code></td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td><code>change_energy_bias(data, frozen_model,...)</code></td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td><code>data_stat(data)</code></td>
<td>Data staticis.</td>
</tr>
<tr>
<td><code>enable_compression([suffix])</code></td>
<td>Enable compression.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td><code>get_class_by_input(input)</code></td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td><code>get_feed_dict(coord_, atype_, natoms, box,...)</code></td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td><code>get_fitting()</code></td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td><code>get_numb_aparam()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get_numb_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get_numb_fparam()</code></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, ...])</code></td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

### class `deepmd.model.tensor.TensorModel(*args, **kwargs)`

**Bases:** `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_nbatches` Number of frames used for data statistic
- `data_stat_protect` Protect parameter for atomic energy regression
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord_, atype_, natoms, box, mesh, ...)</code></td>
<td>Build the model.</td>
</tr>
<tr>
<td><code>build_descrpt(coord_, atype_, natoms, box, ...)</code></td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td><code>build_type_embedding(ntypes[, frz_model, ...])</code></td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td><code>change_energy_bias(data, frozen_model, ...)</code></td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td><code>data_stat(data)</code></td>
<td>Data statistics.</td>
</tr>
<tr>
<td><code>enable_compression([suffix])</code></td>
<td>Enable compression.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td><code>get_class_by_input(input)</code></td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td><code>get_feed_dict(coord_, atype_, natoms, box, ...)</code></td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td><code>get_fitting()</code></td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Get the number of types.</td>
</tr>
<tr>
<td><code>get_numb_aparam()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get_numb_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get_numb_fparam()</code></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, ...])</code></td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

**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

Build the model.

```python
build(coord_, atype_, natoms, box, mesh, input_dict, frz_model=None, ckpt_meta: Optional[str] = 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

Chapter 18. Python API
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

data_stat(data)
    Data statistics.

get_ntypes()
    Get the number of types.

get_out_size()

get_rcut()
    Get cutoff radius of the model.

get_sel_type()

get_type_map()
    Get the type map.

init_variables(graph: Graph, graph_def: 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.model.tensor.WFCModel(*args, **kwargs)

Bases: TensorModel
## Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build(coord, atype, natoms, box, mesh, ...)</code></td>
<td>Build the model.</td>
</tr>
<tr>
<td><code>build_descriptor(coord, atype, natoms, box, ...)</code></td>
<td>Build the descriptor part of the model.</td>
</tr>
<tr>
<td><code>build_type_embedding(ntypes[, frz_model, ...])</code></td>
<td>Build the type embedding part of the model.</td>
</tr>
<tr>
<td><code>change_energy_bias(data, frozen_model, ...)</code></td>
<td>Change the energy bias according to the input data and the pretrained model.</td>
</tr>
<tr>
<td><code>data_stat(data)</code></td>
<td>Data statistics.</td>
</tr>
<tr>
<td><code>enable_compression([suffix])</code></td>
<td>Enable compression.</td>
</tr>
<tr>
<td><code>enable_mixed_precision(mixed_prec)</code></td>
<td>Enable mixed precision for the model.</td>
</tr>
<tr>
<td><code>get_class_by_input(input)</code></td>
<td>Get the class by input data.</td>
</tr>
<tr>
<td><code>get_feed_dict(coord, atype, natoms, box, ...)</code></td>
<td>Generate the feed_dict for current descriptor.</td>
</tr>
<tr>
<td><code>get_fitting()</code></td>
<td>Get the fitting(s).</td>
</tr>
<tr>
<td><code>get_loss(loss, lr)</code></td>
<td>Get the loss function(s).</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Get the number of types.</td>
</tr>
<tr>
<td><code>get_num_a_param()</code></td>
<td>Get the number of atomic parameters.</td>
</tr>
<tr>
<td><code>get_num_dos()</code></td>
<td>Get the number of gridpoints in energy space.</td>
</tr>
<tr>
<td><code>get_num_f_param()</code></td>
<td>Get the number of frame parameters.</td>
</tr>
<tr>
<td><code>get_rcut()</code></td>
<td>Get cutoff radius of the model.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>init_variables(graph, graph_def[, ...])</code></td>
<td>Init the embedding net variables with the given frozen model.</td>
</tr>
<tr>
<td><code>update_sel(global_jdata, local_jdata)</code></td>
<td>Update the selection and perform neighbor statistics.</td>
</tr>
</tbody>
</table>

### deeppmd.nvnmd Package

#### Subpackages

- **deeppmd.nvnmd.data Package**

  Provides
  1. hardware configuration
  2. default input script
  3. title and citation
Data

jdata_sys
    action configuration

jdata_config
    hardware configuration
    dscp
        descriptor configuration
    fitn
        fitting network configuration
    size
        ram capacity
    ctrl
        control flag, such as Time Division Multiplexing (TDM)
    nbit
        number of bits of fixed-point number

jdata_config_16 (disable)
    difference with configure fitting size as 16

jdata_config_32 (disable)
    difference with configure fitting size as 32

jdata_config_64 (disable)
    difference with configure fitting size as 64

jdata_config_128 (default)
    difference with configure fitting size as 128

jdata_configs
    all configure of jdata_config{nfit_node}

jdata_deepmd_input
    default input script for nvnmd training

NVNMD_WELCOME
    nvnmd title when logging

NVNMD_CITATION
    citation of nvnmd

Submodules

depm.nvnmd.data.data module

depm.nvnmd.descriptor package

nvnmd.se_a ==========

Provides

1. building descriptor with continuous embedding network
2. building descriptor with quantized embedding network
Submodules

deepmd.nvnmd.descriptor.se_a module

deepmd.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.nvnmd.descriptor.se_a.build_op_descriptor()
    Replace se_a.py/DescrptSeA/build.

deepmd.nvnmd.descriptor.se_a.check_switch_range(davg, dstd)
    Check the range of switch, let it in range [-2, 14].

deepmd.nvnmd.descriptor.se_a.descript2r4(inputs, natoms)
    Replace \( r_{ji} \rightarrow r'_{ji} \) 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.nvnmd.descriptor.se_a.filter_GR2D(xyz_scatter_1)
    Replace se_a.py/_filter.

deepmd.nvnmd.descriptor.se_a.filter_lower_R42GR(inputs_i, atype)
    Replace se_a.py/DescrptSeA/_filter_lower.

482 Chapter 18. Python API
**deepmd.nvnmd.entrypoints package**

**class deepmd.nvnmd.entrypoints.MapTable**(config_file: str, weight_file: str, map_file: str)

```
Bases: object

Generate the mapping table describing the relationship of atomic distance, cutoff function, and embedding matrix.

three mapping table will be built:

\[
\begin{align*}
  r_{ji}^2 & \rightarrow s_{ji} \\
  r_{ji}^2 & \rightarrow h_{ji} \\
  r_{ji}^2 & \rightarrow G_{ji}
\end{align*}
\]

where \( s_{ji} \) is cut-off function, \( h_{ji} = \frac{s(r_{ji})}{r_{ji}} \), and \( G_{ji} \) is embedding matrix.

The mapping function can be define as:

\[
y = f(x) = y_k + (x - x_k) \ast dy_k \\
y_k = f(x_k) \\
dy_k = \frac{f(x_{k+1}) - f(x_k)}{dx} \\
x_k \leq x < x_{k+1} \\
x_k = k \ast dx
\]

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
**Methods**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>build_grad(x, y, Nr, Nc)</code></td>
<td>: Build gradient of tensor y of x.</td>
</tr>
<tr>
<td><code>build_map_coef(cfgs, x, ys, grads, ...)</code></td>
<td>Build mapping table coefficient cfgs: cfg list cfg = x0, x1, dx.</td>
</tr>
<tr>
<td><code>build_s2g(s)</code></td>
<td>Build s -&gt; G s is switch function G is embedding net output.</td>
</tr>
<tr>
<td><code>build_s2g_grad()</code></td>
<td>Build gradient of G with respect to s.</td>
</tr>
<tr>
<td><code>build_t2g()</code></td>
<td>Build t -&gt; G t is chemical species of center atom and neighbor atom G is embedding net output of type.</td>
</tr>
<tr>
<td><code>build_u2s(r2)</code></td>
<td>Build tensor s, s = s(r2).</td>
</tr>
<tr>
<td><code>build_u2s_grad()</code></td>
<td>Build gradient of s with respect to u (r^2).</td>
</tr>
<tr>
<td><code>cal_coef4(cfgs, x, y, dy)</code></td>
<td>Build mapping table coefficient for one line coef4: a x^3 + b x^2 + c x + d = y: /d = y0</td>
</tr>
<tr>
<td></td>
<td>c = y0’</td>
</tr>
<tr>
<td><code>mapping(x, dic_map, cfgs)</code></td>
<td>Evaluate value by mapping table operation of tensorflow.</td>
</tr>
<tr>
<td><code>mapping2(x, dic_map, cfgs)</code></td>
<td>Evaluate value by mapping table of numpy.</td>
</tr>
<tr>
<td><code>plot_lines(x, dic1[, dic2])</code></td>
<td>Plot lines to see accuracy.</td>
</tr>
<tr>
<td><code>run_s2g()</code></td>
<td>Build s -&gt; graph and run it to get value of mapping table.</td>
</tr>
<tr>
<td><code>run_t2g()</code></td>
<td>Build t -&gt; graph and run it to get value of mapping table.</td>
</tr>
<tr>
<td><code>run_u2s()</code></td>
<td>Build u -&gt; s graph and run it to get value of mapping table.</td>
</tr>
<tr>
<td><code>build_davg_dstd()</code></td>
<td></td>
</tr>
<tr>
<td><code>build_embedding_net()</code></td>
<td>(xx, wbs, activation_fn=&lt;function tanh&gt;)</td>
</tr>
<tr>
<td><code>build_grad(x, y, Nr, Nc)</code></td>
<td>: Build gradient of tensor y of x.</td>
</tr>
<tr>
<td><code>build_map()</code></td>
<td></td>
</tr>
<tr>
<td><code>build_map_coef(cfgs, x, ys, grads, grad_grads, Nr, Nc)</code></td>
<td>Build mapping table coefficient cfgs: cfg list cfg = x0, x1, dx.</td>
</tr>
<tr>
<td></td>
<td>coef4: a x^3 + b x^2 + c x + d = y: /d = y0</td>
</tr>
<tr>
<td><code>build_s2g(s)</code></td>
<td>Build s -&gt; G s is switch function G is embedding net output.</td>
</tr>
<tr>
<td><code>build_s2g_grad()</code></td>
<td>Build gradient of G with respect to s.</td>
</tr>
</tbody>
</table>
build_t2g()
Build t->G t is chemical species of center atom and neighbor atom G is embedding net output of type.

build_u2s(r2)
Build tensor s, s=s(r2).

build_u2s_grad()
Build gradient of s with respect to u (r^2).

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.

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.

run_s2g()
Build s-> graph and run it to get value of mapping table.

run_t2g()
Build t-> graph and run it to get value of mapping table.

run_u2s()
Build u->s graph and run it to get value of mapping table.

class deepmd.nvnmd.entrypoints.Wrap(config_file: str, weight_file: str, map_file: str, model_file: str)
Bases: object
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:

<table>
<thead>
<tr>
<th>pair_style</th>
<th>nvnmd model.pb</th>
</tr>
</thead>
<tbody>
<tr>
<td>pair_coeff</td>
<td>* *</td>
</tr>
</tbody>
</table>

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

Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrap_dscp</td>
<td>Wrap the configuration of descriptor.</td>
</tr>
<tr>
<td>wrap_fitn</td>
<td>Wrap the weights of fitting net.</td>
</tr>
<tr>
<td>wrap_head</td>
<td>Wrap the head information.</td>
</tr>
<tr>
<td>wrap_lut</td>
<td>Wrap the LUT.</td>
</tr>
<tr>
<td>wrap_map</td>
<td>Wrap the mapping table of embedding network.</td>
</tr>
<tr>
<td>wrap_weight</td>
<td>Wrap the weights of fittingNet.</td>
</tr>
</tbody>
</table>

wrap()                      Wrap the configuration of descriptor.

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_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_lut()                  Wrap the LUT.

wrap_map()                  Wrap the mapping table of embedding network.

DeePMD-kit

deepmd.nvnmd.entrypoints.save_weight(sess, file_name: str = 'nvnm/weight.npy')

Save the dictionary of weight to a npy file.

Submodules

deepmd.nvnmd.entrypoints.freeze module

deepmd.nvnmd.entrypoints.freeze.filter_tensorVariableList(tensorVariableList) → dict

Get the name of variable for NVNMD.

\begin{verbatim}
train_attr/min_nbor_dist
descrpt_attr/t_avg:0
descrpt_attr/t_std:0
type_embed_net/matrix_{layer 1}:0
type_embed_net/bias_{layer 1}:0

version 0: | filter_type_{atom i}/matrix_{layer 1}_{atomj}:0

filter_type_{atom i}/bias_{layer 1}_{atomj}:0
layer_{layer 1}.type_{atom i}/matrix:0
layer_{layer 1}.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 1}:0

filter_type_all/bias_{layer 1}:0
filter_type_all/matrix_{layer 1}_two_side_ebd:0
filter_type_all/bias_{layer 1}_two_side_ebd:0
layer_{layer 1}/matrix:0
layer_{layer 1}/bias:0
final_layer/matrix:0
final_layer/bias:0
\end{verbatim}

deepmd.nvnmd.entrypoints.freeze.save_weight(sess, file_name: str = 'nvnm/weight.npy')

Save the dictionary of weight to a npy file.
deepmd.nvnmd.entrypoints.mapt module

class deepmd.nvnmd.entrypoints.mapt.MapTable(config_file: str, weight_file: str, map_file: str)

    Bases: object

    Generate the mapping table describing the relationship of atomic distance, cutoff function, and embedding matrix.

    three mapping table will be built:

    \[
    r_{ji}^2 \rightarrow s_{ji} \\
    r_{ji}^2 \rightarrow h_{ji} \\
    r_{ji}^2 \rightarrow G_{ji}
    \]

    where \(s_{ji}\) is cut-off function, \(h_{ji} = \frac{s(r_{ji})}{r_{ji}}\), and \(G_{ji}\) is embedding matrix.

    The mapping function can be define as:

    \[
    y = f(x) = y_k + (x - x_k) \cdot dy_k \\
    y_k = f(x_k) \\
    dy_k = \frac{f(x_{k+1}) - f(x_k)}{dx} \\
    x_k \leq x < x_{k+1} \\
    x_k = k \cdot dx
    \]

    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
Methods

\texttt{build\_grad}(x, y, Nr, Nc) \quad : \text{Build gradient of tensor y of x.}

\texttt{build\_map\_coef}(cfgs, x, ys, grads, ...) \quad Build mapping table coefficient cfgs: cfg list cfg = x0, x1, dx.

\texttt{build\_s2g}(s) \quad Build \text{s} \rightarrow \text{G} s is switch function G is embedding net output.

\texttt{build\_s2g\_grad()} \quad Build gradient of G with respect to s.

\texttt{build\_t2g()} \quad Build \text{t} \rightarrow \text{G} t is chemical species of center atom and neighbor atom G is embedding net output of type.

\texttt{build\_u2s}(r2) \quad Build tensor s, s=s(r2).

\texttt{build\_u2s\_grad()} \quad Build gradient of s with respect to u (r^-2).

\texttt{cal\_coef4}(cfgs, x, y, dy) \quad 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' - 2y1 + dx y0' + 2y0) / dx^3

\texttt{mapping}(x, dic\_map, cfgs) \quad Evaluate value by mapping table operation of tensorflow.

\texttt{mapping2}(x, dic\_map, cfgs) \quad Evaluate value by mapping table of numpy.

\texttt{plot\_lines}(x, dic1[, dic2]) \quad Plot lines to see accuracy.

\texttt{run\_s2g()} \quad Build s \rightarrow graph and run it to get value of mapping table.

\texttt{run\_t2g()} \quad Build t \rightarrow graph and run it to get value of mapping table.

\texttt{run\_u2s()} \quad Build u \rightarrow graph and run it to get value of mapping table.
**DeePMD-kit**

```python
build_t2g()
    Build t->G t is chemical species of center atom and neighbor atom G is embedding net output of type.

build_u2s(r2)
    Build tensor s, s=s(r2).

build_u2s_grad()
    Build gradient of s with respect to u (r^2).

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.

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.

run_s2g()
    Build s-> graph and run it to get value of mapping table.

run_t2g()
    Build t-> graph and run it to get value of mapping table.

run_u2s()
    Build u->s graph and run it to get value of mapping table.
```

**deepmd.nvnmd.entrypoints.map**

```python
depchnmd.nvnmd.entrypoints.mapt.map(*, nvnmd_config: Optional[str] = 'nvnmd/config.npy',
    nvnmd_weight: Optional[str] = 'nvnmd/weight.npy',
    nvnmd_map: Optional[str] = 'nvnmd/map.npy', **kwargs)
```

**deepmd.nvnmd.entrypoints.train module**

```python
depchnmd.nvnmd.entrypoints.train.normalized_input(fn, PATH_CNN, CONFIG_CNN)
    Normalize a input script file for continuous neural network.

depchnmd.nvnmd.entrypoints.train.normalized_input_qnn(jdata, PATH_QNN, CONFIG_CNN,
    WEIGHT_CNN, MAP_CNN)
    Normalize a input script file for quantize neural network.

depchnmd.nvnmd.entrypoints.train.train_nvnmd(*, INPUT: str, init_model: Optional[str], restart:
    Optional[str], step: str, skip_neighbor_stat: bool = False, **kwargs)
```
DeepMD-kit

deepmd.nvnmd.entrypoints.wrap module

```python
class deepmd.nvnmd.entrypoints.wrap.Wrap(
    config_file: str, weight_file: str, map_file: str, model_file: str
)
```

Bases: object

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:

```plaintext
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

Methods

- `wrap_dscrp()`: Wrap the configuration of descriptor.
- `wrap_fittn()`: Wrap the weights of fitting net.
- `wrap_head(nhs, nws)`: Wrap the head information.
- `wrap_lut()`: Wrap the LUT.
- `wrap_map()`: Wrap the mapping table of embedding network.

```python
wrap
wrap_bias
```

18.2. deepmd package

491
wrap_dscp()
Wrap the configuration of descriptor.
version 0: [NBIT_IDX_S2G-1:0] SHIFT_IDX_S2G
[NBIT_NEIB*NTYPE-1:0] SELs [NBITFIXD*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.
weight b bias

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_lut()
Wrap the LUT.

wrap_map()
Wrap the mapping table of embedding network.

wrap_weight(weight, NBIT_DISP, NBIT_WEIGHT)

deepmd.nvnmd.entrypoints.wrap.wrap(*, nvnmd_config: Optional[str] = ‘nvnmd/config.npy’,
nvnmd_weight: Optional[str] = ‘nvnmd/weight.npy’,

deepmd.nvnmd.fit package

nvnmd.fit =======
Provides
  1. continuous fitting network
  2. quantized fitting network

Submodules

deepmd.nvnmd.fit.ener module

deepmd.nvnmd.fit.ener.one_layer_nvnmd(inputs, outputs_size, activation_fn=<function tanh>,
  precision=tf.float64, 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 initialized with random or constant value.

**deepmd.nvnmd.utils package**

**class deepmd.nvnmd.utils.Encode**

- **Bases:** object

Encoding value as hex, bin, and dec format.

**Methods**

- `bin2hex(data)` Convert binary string list to hex string list.
- `bin2hex_str(sbin)` Convert binary string to hex string.
- `byte2hex(bs, nbyte)` Convert byte into hex bs: low byte in the first hex: low byte in the right.
- `check_dec(idec, nbit[, signed, name])` Check whether the data (idec) is in the range range is \([0, 2^{nbit} - 1]\) for unsigned range is \([-2^{nbit-1}, 2^{nbit-1} - 1]\) for signed.
- `dec2bin(idec[, nbit, signed, name])` Convert dec array to binary string list.
- `extend_bin(sbin, nfull)` Extend the element of list (sbin) to the length (nfull).
- `extend_hex(slhex, nfull)` Extend the element of list (slhex) to the length (nfull).
- `extend_list(sbin, nfull)` Extend the list (sbin) to the length (nfull) the attached element of list is 0.
- `flt2bin(data, nbit_exp, nbit_frac)` Convert float into binary string list.
- `hex2bin(data)` Convert hex string list to binary string list.
- `hex2bin_str(shex)` Convert hex string to binary string.
- `merge_bin(sbin, nmerge)` Merge binary string list per nmerge value.
- `qc(v[, nbit])` Quantize value using ceil.
- `qf(v[, nbit])` Quantize value using floor.
- `qr(v[, nbit])` Quantize value using round.
- `reverse_bin(sbin, nreverse)` Reverse binary string list per nreverse value.
- `split_bin(sbin, nbit)` Split sbin into many segment with the length nbit.

**find_max_expo**
**flt2bin_one**
**norm_expo**
**split_exp_zero**

- `bin2hex(data)` Convert binary string list to hex string list.
- `bin2hex_str(sbin)` Convert binary string to hex string.
DeePMD-kit

```python
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 $[0, 2^nbit - 1]$ for unsigned range is $[-2^{nbit-1}, 2^{nbit-1} - 1]$ for signed.

dec2bin(idec, nbit=10, signed=False, name='')
    Convert dec array to binary string list.

extend_bin(slbin, nfull)
    Extend the element of list (slbin) to the length (nfull).
    such as, when

    slbin = ['10010', '10100'],
    nfull = 6

    extend to
    ['010010', '010100']

extend_hex(slhex, nfull)
    Extend the element of list (slhex) to the length (nfull).

extend_list(slbin, nfull)
    Extend the list (slbin) to the length (nfull) the attached element of list is 0.
    such as, when

    slbin = ['10010', '10100'],
    nfull = 4

    extend it to
    ['10010', '10100', '00000', '00000']

find_max_expo(v, expo_min=-1000)

flt2bin(data, nbit_expo, nbit_frac)
    Convert float into binary string list.

flt2bin_one(v, nbit_expo, nbit_frac)

hex2bin(data)
    Convert hex string list to binary string list.

hex2bin_str(shex)
    Convert hex string to binary string.

merge_bin(slbin, nmerge)
    Merge binary string list per nmerge value.

norm_expo(v, nbit_frac=20, expo_min=-1000)
```

Chapter 18. Python API
qc(v, nbit: int = 14)
    Quantize value using ceil.
qf(v, nbit: int = 14)
    Quantize value using floor.
qr(v, nbit: int = 14)
    Quantize value using round.
reverse_bin(slbin, nreverse)
    Reverse binary string list per nreverse value.
split_bin(sbin, nbit: int)
    Split sbin into many segment with the length nbit.
split_expo_mant(v, min=-1000)

class deepmd.nvnmd.utils.FioBin
    Bases: object
    Input and output for binary file.

    Methods

    load(file_name, default_value)
        Load binary file into bytes value.
    save(file_name, data)
        Save hex string into 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.nvnmd.utils.FioDic
    Bases: object
    Input and output for dict class data the file can be .json or .npy file containing a dictionary.

    Methods

    update(jdata, jdata_o)
        Update key-value pair is key in jdata_o.keys().
    get
    load
    save

    get(jdata, key, default_value)
    load(file_name=",", default_value={})
    save(file_name=",", dic={})
update(jdata, jdata_o)
    Update key-value pair is key in jdata_o.keys().
    Parameters
    jdata
    new_jdata
    jdata_o
    origin_jdata
class deepmd.nvnmd.utils.FioTxt
    Bases: object
    Input and output for .txt file with string.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>load(file_name=default_value)</td>
<td>Load .txt file into string list.</td>
<td>file_name: str, default_value: list</td>
</tr>
<tr>
<td>save(file_name=data)</td>
<td>Save string list into .txt file.</td>
<td>file_name: str, data: list</td>
</tr>
</tbody>
</table>

deemp.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~n-type-1
    layer_l
        layer order in embedding network 1~n-layer

deemp.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~n-type-1
    layer_l
        [int] layer order in embedding network 0~n-layer-1
    nlayer
        [int] number of layers
deepmd.nvnmd.utils.map_nvnmd(x, map_y, map_dy, prec, nbit=None)

Mapping function implemented by numpy.

deepmd.nvnmd.utils.nvnmd_args()

deepmd.nvnmd.utils.one_layer(inputs, outputs_size, activation_fn=<function tanh>,
    precision=tf.float64, 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.

Submodules

deepmd.nvnmd.utils.argcheck module

Alias for backward compatibility.

deepmd.nvnmd.utils.argcheck.nvnmd_args()

deepmd.nvnmd.utils.config module

class deepmd.nvnmd.utils.config.NvnmdConfig(jdata: dict)
  Bases: object

  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
Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>disp_message()</code></td>
<td>Display the log of NVNMD.</td>
</tr>
<tr>
<td><code>get_deepmd_jdata()</code></td>
<td>Generate input script with member element one by one.</td>
</tr>
<tr>
<td><code>get_dp_init_weights()</code></td>
<td>Build the weight dict for initialization of net.</td>
</tr>
<tr>
<td><code>get_dscp_jdata()</code></td>
<td>Generate model(descriptor) in input script.</td>
</tr>
<tr>
<td><code>get_fitn_jdata()</code></td>
<td>Generate model(gradient) in input script.</td>
</tr>
<tr>
<td><code>get_learning_rate_jdata()</code></td>
<td>Generate learning_rate in input script.</td>
</tr>
<tr>
<td><code>get_loss_jdata()</code></td>
<td>Generate loss in input script.</td>
</tr>
<tr>
<td><code>get_model_jdata()</code></td>
<td>Generate model in input script.</td>
</tr>
<tr>
<td><code>get_nvnmnd_jdata()</code></td>
<td>Generate nvnmnd in input script.</td>
</tr>
<tr>
<td><code>get_s_range(davg, dstd)</code></td>
<td>Get the range of switch function.</td>
</tr>
<tr>
<td><code>get_training_jdata()</code></td>
<td>Generate training in input script.</td>
</tr>
<tr>
<td><code>init_config_by_version(version, max_nnei)</code></td>
<td>Initialize version-dependent parameters.</td>
</tr>
<tr>
<td><code>init_ctrl(jdata[, jdata parent])</code></td>
<td>Initialize members about control signal.</td>
</tr>
<tr>
<td><code>init_dpin(jdata[, jdata parent])</code></td>
<td>Initialize members about other deepmd input.</td>
</tr>
<tr>
<td><code>init_dscp(jdata[, jdata parent])</code></td>
<td>Initialize members about descriptor.</td>
</tr>
<tr>
<td><code>init_fitn(jdata[, jdata parent])</code></td>
<td>Initialize members about fitting network.</td>
</tr>
<tr>
<td><code>init_from_config(jdata)</code></td>
<td>Initialize member element one by one.</td>
</tr>
<tr>
<td><code>init_from_deepmd_input(jdata)</code></td>
<td>Initialize members with input script of deepmd.</td>
</tr>
<tr>
<td><code>init_from_jdata(jdata)</code></td>
<td>Initialize this class with jdata loaded from input script.</td>
</tr>
<tr>
<td><code>init_nbit(jdata[, jdata parent])</code></td>
<td>Initialize members about quantification precision.</td>
</tr>
<tr>
<td><code>init_net_size()</code></td>
<td>Initialize net_size.</td>
</tr>
<tr>
<td><code>init_size(jdata[, jdata parent])</code></td>
<td>Initialize members about ram capacity.</td>
</tr>
<tr>
<td><code>init_train_mode([mod])</code></td>
<td>Configure for training cnn or qnn.</td>
</tr>
<tr>
<td><code>init_value()</code></td>
<td>Initialize member with dict.</td>
</tr>
<tr>
<td><code>save(file_name)</code></td>
<td>Save all configuration to file.</td>
</tr>
<tr>
<td><code>set_ntype(ntype)</code></td>
<td>Set the number of type.</td>
</tr>
<tr>
<td><code>update_config()</code></td>
<td>Update config from dict.</td>
</tr>
</tbody>
</table>

`disp_message()`
Display the log of NVNMD.

`get_deepmd_jdata()`
Generate input script with member element one by one.

`get_dp_init_weights()`
Build the weight dict for initialization of net.

`get_dscp_jdata()`
Generate model(descriptor) in input script.

`get_fitn_jdata()`
Generate model(gradient) in input script.

`get_learning_rate_jdata()`
Generate learning_rate in input script.

`get_loss_jdata()`
Generate loss in input script.
get_model_jdata()
Generate model in input script.

get_nvnmd_jdata()
Generate nvnmd in input script.

get_s_range(davg, dstd)
Get the range of switch function.

get_training_jdata()
Generate training in input script.

init_config_by_version(version, max_nnei)
Initialize version-dependent parameters.

init_ctrl(jdata: dict, jdata_parent: dict = {}): dict
Initialize members about control signal.

init_dpinc(jdata: dict, jdata_parent: dict = {}): dict
Initialize members about other deepmd input.

init_dscp(jdata: dict, jdata_parent: dict = {}): dict
Initialize members about descriptor.

init_finn(jdata: dict, jdata_parent: dict = {}): dict
Initialize members about fitting network.

init_from_config(jdata)
Initialize member element one by one.

init_from_deepmd_input(jdata)
Initialize members with input script of deepmd.

init_from_jdata(jdata: dict = {})
Initialize this class with jdata loaded from input script.

init_nbit(jdata: dict, jdata_parent: dict = {}): dict
Initialize members about quantification precision.

init_net_size()
Initialize net_size.

init_size(jdata: dict, jdata_parent: dict = {}): dict
Initialize members about ram capacity.

init_train_mode(mod='cnn')
Configure for training cnn or qnn.

init_value()
Initialize member with dict.

save(file_name=None)
Save all configuration to file.

set_ntype(ctype)
Set the number of type.

update_config()
Update config from dict.
deepmd.nvmd.utils.encode module

class deepmd.nvmd.utils.encode.Encode

Bases: object

Encoding value as hex, bin, and dec format.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bin2hex(data)</code></td>
<td>Convert binary string list to hex string list.</td>
</tr>
<tr>
<td><code>bin2hex_str(sbin)</code></td>
<td>Convert binary string to hex string.</td>
</tr>
<tr>
<td><code>byte2hex(bs, nbyte)</code></td>
<td>Convert byte into hex bs: low byte in the first hex: low byte in the right.</td>
</tr>
<tr>
<td><code>check_dec(idec, nbit[, signed, name])</code></td>
<td>Check whether the data (idec) is in the range range is <code>[0, 2^{nbit} − 1]</code> for unsigned range is <code>[-2^{nbit-1}, 2^{nbit-1} − 1]</code> for signed.</td>
</tr>
<tr>
<td><code>dec2bin(idec[, nbit, signed, name])</code></td>
<td>Convert dec array to binary string list.</td>
</tr>
<tr>
<td><code>extend_bin(sbin, nfull)</code></td>
<td>Extend the element of list (sbin) to the length (nfull).</td>
</tr>
<tr>
<td><code>extend_hex(slhex, nfull)</code></td>
<td>Extend the element of list (slhex) to the length (nfull).</td>
</tr>
<tr>
<td><code>extend_list(sbin, nfull)</code></td>
<td>Extend the list (sbin) to the length (nfull) the attched element of list is 0.</td>
</tr>
<tr>
<td><code>flt2bin(data, nbit_expo, nbit_frac)</code></td>
<td>Convert float into binary string list.</td>
</tr>
<tr>
<td><code>hex2bin(data)</code></td>
<td>Convert hex string list to binary string list.</td>
</tr>
<tr>
<td><code>hex2bin_str(shex)</code></td>
<td>Convert hex string to binary string.</td>
</tr>
<tr>
<td><code>merge_bin(sbin, nmerge)</code></td>
<td>Merge binary string list per nmerge value.</td>
</tr>
<tr>
<td><code>q(d[, nbit])</code></td>
<td>Quantize value using ceil.</td>
</tr>
<tr>
<td><code>qf(v[, nbit])</code></td>
<td>Quantize value using floor.</td>
</tr>
<tr>
<td><code>qr(v[, nbit])</code></td>
<td>Quantize value using round.</td>
</tr>
<tr>
<td><code>reverse_bin(sbin, nreverse)</code></td>
<td>Reverse binary string list per nreverse value.</td>
</tr>
<tr>
<td><code>split_bin(sbin, nbit)</code></td>
<td>Split sbin into many segment with the length nbit.</td>
</tr>
</tbody>
</table>

```
find_max_expo
flt2bin_one
norm_expo
split_exp_mant
```

`bin2hex(data)`

Convert binary string list to hex string list.

`bin2hex_str(sbin)`

Convert binary string to hex string.

`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^{nbit} − 1]` for unsigned range is `[-2^{nbit-1}, 2^{nbit-1} − 1]` for signed.
**Dec2bin**

\( \text{dec2bin}(\text{idec, nbit=10, signed=False, name='')} \)

Convert dec array to binary string list.

**extend_bin**

\( \text{extend_bin}(\text{slbin, nfull}) \)

Extend the element of list (slbin) to the length (nfull).

such as, when

\[
\text{slbin} = \left[ '10010', '10100' \right], \\
n\text{full} = 6
\]

extent to

\[
\left[ '010010', '010100' \right]
\]

**extend_hex**

\( \text{extend_hex}(\text{slhex, nfull}) \)

Extend the element of list (slhex) to the length (nfull).

**extend_list**

\( \text{extend_list}(\text{slbin, nfull}) \)

Extend the list (slbin) to the length (nfull) the attached element of list is 0.

such as, when

\[
\text{slbin} = \left[ '10010', '10100' \right], \\
n\text{full} = 4
\]

extent it to

\[
\left[ '10010', '10100', '00000', '00000' \right]
\]

**find_max_expo**

\( \text{find_max_expo}(v, \text{expo_min}=-1000) \)

**flt2bin**

\( \text{flt2bin}(\text{data, nbit\_expo, nbit\_frac}) \)

Convert float into binary string list.

**flt2bin\_one**

\( \text{flt2bin\_one}(v, \text{nbit\_expo, nbit\_frac}) \)

**hex2bin**

\( \text{hex2bin}(\text{data}) \)

Convert hex string list to binary string list.

**hex2bin\_str**

\( \text{hex2bin\_str}(\text{shex}) \)

Convert hex string to binary string.

**merge_bin**

\( \text{merge_bin}(\text{slbin, nmerge}) \)

Merge binary string list per nmerge value.

**norm_expo**

\( \text{norm_expo}(v, \text{nbit\_frac}=20, \text{expo\_min}=-1000) \)

**qc**

\( \text{qc}(v, \text{nbit: int}=14) \)

Quantize value using ceil.

**qf**

\( \text{qf}(v, \text{nbit: int}=14) \)

Quantize value using floor.
DeePMD-kit

qr(v, nbit: int = 14)
    Quantize value using round.
reverse_bin(sbin, nreverse)
    Reverse binary string list per nreverse value.
split_bin(sbin, nbit: int)
    Split sbin into many segment with the length nbit.
split_expo_mant(v, min=-1000)

deepmd.nvnmd.utils.fio module

class deepmd.nvnmd.utils.fio.Fio
    Bases: object
    Basic class for FIO.

    Methods

    create_file_path(file_name='')
    exits(file_name='')
    get_file_list(path) → list
    is_file(file_name)
    is_path(path)
    mkdir(path_name='')

class deepmd.nvnmd.utils.fio.FioBin
    Bases: object
    Input and output for binary file.
**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>load</code></td>
<td>Load binary file into bytes value.</td>
</tr>
<tr>
<td><code>save</code></td>
<td>Save hex string into binary file.</td>
</tr>
</tbody>
</table>

**class deepmd.nvnmd.utils.fio.FioDic**

**Bases:** object

Input and output for dict class data the file can be .json or .npy file containing a dictionary.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>update</code></td>
<td>Update key-value pair is key in jdata_o.keys().</td>
</tr>
</tbody>
</table>

**get** (jdata, key, default_value)

**load** (file_name="", default_value={})

**save** (file_name="", dic={})

**update** (jdata, jdata_o)

Update key-value pair is key in jdata_o.keys().

Parameters

- `jdata`
- `new_jdata`
- `jdata_o`
- `origin_jdata`

**class deepmd.nvnmd.utils.fio.FioJsonDic**

**Bases:** object

Input and output for .json file containing dictionary.
DeePMD-kit

Methods

<table>
<thead>
<tr>
<th>load(file_name, default_value)</th>
<th>Load json file into dict.</th>
</tr>
</thead>
<tbody>
<tr>
<td>save(file_name, dic)</td>
<td>Save dict into json file.</td>
</tr>
</tbody>
</table>

load(file_name="", default_value={})
Load json file into dict.

save(file_name="", dic={})
Save dict into json file.

class deepmd.nvnmd.utils.fio.FioNpyDic
Bases: object
Input and output for .npy file containing dictionary.

Methods

<table>
<thead>
<tr>
<th>load(file_name, default_value)</th>
<th>Load .txt file into string list.</th>
</tr>
</thead>
<tbody>
<tr>
<td>save(file_name, data)</td>
<td>Save string list into .txt file.</td>
</tr>
</tbody>
</table>

load(file_name="", default_value=[])  
Load .txt file into string list.

save(file_name: str = "", data: list = [])  
Save string list into .txt file.

deepmd.nvnmd.utils.network module

deepmd.nvnmd.utils.network.get_sess()

deepmd.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.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.nvnmd.utils.network.one_layer(inputs, outputs_size, activation_fn=<function tanh>,
precision=tf.float64, 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.nvnmd.utils.network.one_layer_t(shape, outputs_size, bavg, stddev, precision, trainable,
initial_variables, seed, uniform_seed, name)

deepmd.nvnmd.utils.network.qf(x, nbit)
Quantize and floor tensor x with quantification precision nbit.

deepmd.nvnmd.utils.network.qr(x, nbit)
Quantize and round tensor x with quantification precision nbit.

deepmd.nvnmd.utils.network.tanh4(x)

deepmd.nvnmd.utils.op module

deepmd.nvnmd.utils.op.map_nvnmd(x, map_y, map_dy, prec, nbit=None)
Mapping function implemented by numpy.

deepmd.nvnmd.utils.op.r2s(r, rmin, rmax)

deepmd.nvnmd.utils.weight module

deepmd.nvnmd.utils.weight.get_constant_initializer(weights, name)
Get initial value by name and create a initializer.

deepmd.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.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
DeePMD-kit

spe_j
[int] special order of neighbor atom j 0~ntype-1

layer_l
layer order in embedding network 1~nlayer

deepmd.nvmd.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.nvmd.utils.weight.get_normalize(weights: dict)
Get normalize parameter (avg and std) of s_{ji}.

deepmd.nvmd.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.nvmd.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.nvmd.utils.weight.get_weight(weights, key)
Get weight value according to key.
**deepmd.op package**

This module will house custom TF OPs after CMake installation.

```python
deepmd.op.import_ops()
```

Import all custom TF ops that are present in this submodule.

**Notes**

Initially, this subdir is unpopulated. CMake will install all the op module python files and shared libs.

**deepmd.train package**

**Submodules**

**deepmd.train.run_options module**

Module taking care of important package constants.

```python
class deepmd.train.run_options.RunOptions(init_model: Optional[str] = None, init_frz_model:
Optional[str] = None, finetune: Optional[str] = None, restart: Optional[str] = None, log_path:
Optional[str] = None, log_level: int = 0, mpi_log: str = 'master')
```

Bases: `object`

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 distributed training it is true for the main MPI process in serial 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
  - device type - gpu or cpu
Methods

```python
print_resource_summary()    Print build and current running cluster configuration summary.

gpus: Optional[List[int]]

property is_chief
    Whether my rank is 0.

my_device: str

my_rank: int

nodelist: List[int]

nodename: str

print_resource_summary()
    Print build and current running cluster configuration summary.

world_size: int
```

depmd.train.trainer module

class deepmd.train.trainer.DPTrainer(jdata, run_opt, is_compress=False)
    Bases: object

Methods

```python
save_compressed()    Save the compressed graph.

build(data=None, stop_batch=0, origin_type_map=None, suffix='')

static eval_single_list(single_batch_list, loss, sess, get_feed_dict_func, prefix='')

get_evaluation_results(batch_list)

get_feed_dict(batch, is_training)
```
get_global_step()

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)

save_checkpoint(cur_batch: int)

save_compressed()
    Save the compressed graph.

train(train_data=None, valid_data=None)

valid_on_the_fly(fp, train_batches, valid_batches, print_header=False, fitting_key=None)

class deepmd.train.trainer.DatasetLoader(train_data: DeepmdDataSystem)

    Bases: object

    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)

Methods

build() Build the OP that loads the training data.

get_data_dict(batch_list) Generate a dict of the loaded data.

build() → List[Tensor]
    Build the OP that loads the training data.

    Returns
    List[tf.Tensor]
        Tensor of the loaded data.

get_data_dict(batch_list: List[ndarray]) → Dict[str, ndarray]
    Generate a dict of the loaded data.

    Parameters
    batch_list
        [List[np.ndarray]] The loaded data.
Returns

\[
\text{Dict[str, np.ndarray]}
\]

The dict of the loaded data.

deepmd.utils package

class deepmd.utils.DeepmdData(sys_path: str, set_prefix: str = 'set', shuffle_test: bool = True,
type_map: Optional[List[str]] = None, optional_type_map: bool = True, modifier=None, trn_all_set: bool = False, sort_atoms: bool = True)

Bases: object

Class for a data system.

It loads data from hard disk, and maintains 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**
  \([\text{bool}]\) Sort atoms by atom types. Required to enable when the data is directly fed to descriptors except mixed types.
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(key, ndof[, atomic, must, high_prec, ...])</code></td>
<td>Add a data item that to be loaded.</td>
</tr>
<tr>
<td><code>avg(key)</code></td>
<td>Return the average value of an item.</td>
</tr>
<tr>
<td><code>check_batch_size(batch_size)</code></td>
<td>Check if the system can get a batch of data with batch_size frames.</td>
</tr>
<tr>
<td><code>check_test_size(test_size)</code></td>
<td>Check if the system can get a test dataset with test_size frames.</td>
</tr>
<tr>
<td><code>get_atom_type()</code></td>
<td>Get atom types.</td>
</tr>
<tr>
<td><code>get_batch(batch_size)</code></td>
<td>Get a batch of data with batch_size frames.</td>
</tr>
<tr>
<td><code>get_data_dict()</code></td>
<td>Get the data_dict.</td>
</tr>
<tr>
<td><code>get_natoms()</code></td>
<td>Get number of atoms.</td>
</tr>
<tr>
<td><code>get_natoms_vec(ntypes)</code></td>
<td>Get number of atoms and number of atoms in different types.</td>
</tr>
<tr>
<td><code>get_numb_batch(batch_size, set_idx)</code></td>
<td>Get the number of batches in a set.</td>
</tr>
<tr>
<td><code>get_num_set()</code></td>
<td>Get number of training sets.</td>
</tr>
<tr>
<td><code>get_sys_numb_batch(batch_size)</code></td>
<td>Get the number of batches in the data system.</td>
</tr>
<tr>
<td><code>get_test([ntests])</code></td>
<td>Get the test data with ntests frames.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>reduce(key_out, key_in)</code></td>
<td>Generate a new item from the reduction of another atom.</td>
</tr>
</tbody>
</table>

```python
reset_get_batch
```

```python
add(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: Optional[List[int]] = None, repeat: int = 1, default: float = 0.0, dtype: Optional[dtype] = None)
```

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

avg(key)
    Return the average value of an item.

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_atom_type() \rightarrow \text{List}[\text{int}]
    Get atom types.

get_batch(batch_size: \text{int}) \rightarrow \text{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_data_dict() \rightarrow \text{dict}
    Get the data_dict.

get_natoms()
    Get number of atoms.

get_natoms_vec(ntypes: \text{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

get_ntypes() \rightarrow \text{int}
    Number of atom types in the system.

get_numb_batch(batch_size: \text{int}, set_idx: \text{int}) \rightarrow \text{int}
    Get the number of batches in a set.

get_numb_set() \rightarrow \text{int}
    Get number of training sets.

get_sys_numb_batch(batch_size: \text{int}) \rightarrow \text{int}
    Get the number of batches in the data system.

get_test(ntests: \text{int} = -1) \rightarrow \text{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_type_map() → List[str]

Get the type map.

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

reset_get_batch()

class deepmd.utils.DeepmdDataSystem(
    systems: List[str],
    batch_size: int, test_size: int, rcut:
    Optional[float] = None,
    set_prefix: str = 'set',
    shuffle_test: bool = True,
    type_map: Optional[List[str]] = None,
    optional_type_map: bool = True,
    trn_all_set=False,
    sys_probs=None,
    auto_prob_style='prob_sys_size',
    sort_atoms: bool = True)

Bases: object

Class for manipulating many data systems.

It is implemented with the help of DeepmdData

Attributes

default_mesh
Mesh for each system.

Methods

add(key, ndof[, atomic, must, high_prec, ...])
Add a data item that to be loaded.

add_dict(adict)
Add items to the data system by a dict.

get_batch([sys_idx])
Get a batch of data from the data systems.

get_batch_mixed()
Get a batch of data from the data systems in the mixed way.

get_batch_size()
Get the batch size.

get_batch_standard([sys_idx])
Get a batch of data from the data systems in the standard way.

get_nbatches()
Get the total number of batches.

get_nsystems()
Get the number of data systems.

get_ntypes()
Get the number of types.

get_sys(idx)
Get a certain data system.

get_sys_ntest([sys_idx])
Get number of tests for the currently selected system, or one defined by sys_idx.

get_test([sys_idx, n_test])
Get test data from the the data systems.

get_type_map()
Get the type map.

reduce(key_out, key_in)
Generate a new item from the reduction of another atom.
```py
add(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: Optional[List[int]] = None, repeat: int = 1, default: float = 0.0)
```

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=`0.0`

Default value of data

```py
add_dict(adict: dict) → None
```

Add items to the data system by a dict. adict should have items like ..

```py
adict[key] = {
    "ndof": ndof, "atomic": atomic, "must": must, "high_prec": high_prec, "type_sel":
    type_sel, "repeat": repeat,
}
```

For the explanation of the keys see add

```py
compute_energy_shift(rcond=None, key='energy')
```

```py
property default_mesh: List[ndarray]
```

Mesh for each system.

```py
get_batch(sys_idx: Optional[int] = 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_mixed() → dict
Get a batch of data from the data systems in the mixed way.

Returns
dict
The batch data

get_batch_size() → int
Get the batch size.

get_batch_standard(sys_idx: Optional[int] = 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_data_dict(ii: int = 0) → dict

get_nbatches() → int
Get the total number of batches.

get_nsystems() → int
Get the number of data systems.

get_ntypes() → int
Get the number of types.

get_sys(idx: int) → DeepmdData
Get a certain data system.

get_sys_nbest(sys_idx=None)
Get number of tests for the currently selected system, or one defined by sys_idx.

get_test(sys_idx: Optional[int] = None, n_test: int = -1)
Get test data from the the data systems.

Parameters
sys_idx
The test data 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_type_map() \rightarrow \text{List}[\text{str}]
Get the type map.

print_summary(name)

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

set_sys_probs(sys_probs=None, auto_prob_style: str = 'prob_sys_size')

class deepmd.utils.LearningRateExp(start_lr: float, stop_lr: float = 5e-08, decay_steps: int = 5000, decay_rate: float = 0.95)

Bases: object

The exponentially decaying learning rate.
The learning rate at step $t$ is given by

$$\alpha(t) = \alpha_0 \lambda^{t/\tau}$$

where $\alpha$ is the learning rate, $\alpha_0$ is the starting learning rate, $\lambda$ is the decay rate, and $\tau$ is the decay steps.

Parameters
start_lr
Starting learning rate $\alpha_0$
stop_lr
Stop learning rate $\alpha_1$
decay_steps
Learning rate decay every this number of steps $\tau$
decay_rate
The decay rate $\lambda$. If stop_step is provided in build, then it will be determined automatically and overwritten.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>Build the learning rate.</td>
</tr>
<tr>
<td>start_lr()</td>
<td>Get the start lr.</td>
</tr>
<tr>
<td>value(step)</td>
<td>Get the lr at a certain step.</td>
</tr>
</tbody>
</table>

build(global_step, stop_step: Optional[int] = None) \rightarrow \text{Tensor}
Build the learning rate.

Parameters
global_step
   The tf Tensor providing 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.utils.PairTab(filename: str)
   Bases: object
   
   Pairwise tabulated potential.
   
   Parameters

   filename
      File name for the short-range tabulated potential. The table is a text data file with 
      \((N_t + 1) \times N_t / 2 + 1\) columns. The first column is the distance between atoms. The 
      second to the last columns are energies for pairs of certain types. For example we 
      have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 
      correspondingly.

   Methods

   get() Get the serialized table.

   reinit(filename) Initialize the tabulated interaction.

get() → Tuple[array, array]
   Get the serialized table.

reinit(filename: str) → None
   Initialize the tabulated interaction.
   
   Parameters

   filename
      File name for the short-range tabulated potential. The table is a text data file with 
      \((N_t + 1) \times N_t / 2 + 1\) columns. The first column is the distance between atoms. The 
      second to the last columns are energies for pairs of certain types. For example we 
      have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

class deepmd.utils.Plugin
   Bases: object
   
   A class to register and restore plugins.
Examples

```python
>>> plugin = Plugin()
>>> @plugin.register("xx")
    def xxx():
        pass
>>> print(plugin.plugins['xx'])
```

Attributes

- `plugins`:
  - `Dict[str, object]` plugins

Methods

- `get_plugin(key)` — Visit a plugin by key.
  - Parameters:
    - `key` — `str` key of the plugin
  - Returns:
    - `object` — the plugin

- `register(key)` — Register a plugin.
  - Parameters:
    - `key` — `str` key of the plugin
  - Returns:
    - `Callable[[object, object]]` — decorator

```python
class deepmd.utils.PluginVariant(*args, **kwargs)
Bases: object
```

A class to remove type from input arguments.
Submodules

**deepmd.utils.argcheck module**

Alias for backward compatibility.

- `deepmd.utils.argcheck.gen_args(**kwargs) → List[Argument]`
- `deepmd.utils.argcheck.gen_doc(*, make_anchor=True, make_link=True, **kwargs)`
- `deepmd.utils.argcheck.gen_json(**kwargs)`
- `deepmd.utils.argcheck.list_to_doc(xx)`
- `deepmd.utils.argcheck.normalize(data)`
- `deepmd.utils.argcheck.type_embedding_args()`

**deepmd.utils.batch_size module**

```python
class deepmd.utils.batch_size.AutoBatchSize(initial_batch_size: int = 1024, factor: float = 2.0)
    Bases: AutoBatchSize
```

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>execute(callable, start_index, natoms)</code></td>
<td>Execute a method with given batch size.</td>
</tr>
<tr>
<td><code>execute_all(callable, total_size, natoms, ...)</code></td>
<td>Execute a method with all given data.</td>
</tr>
<tr>
<td><code>is_gpu_available()</code></td>
<td>Check if GPU is available.</td>
</tr>
<tr>
<td><code>is_oom_error(e)</code></td>
<td>Check if the exception is an OOM error.</td>
</tr>
</tbody>
</table>

- `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.utils.compat module**

Alias for backward compatibility.

```
deepmd.utils.compat.convert_input_v0_v1(jdata: Dict[str, Any], warning: bool = True, dump: Optional[Union[str, Path]] = None) → Dict[str, Any]
```

Convert input from v0 format to v1.

**Parameters**

- `jdata`: Dict[str, Any] - loaded json/yaml file
- `warning`: bool - whether to show deprecation warning, by default True
- `dump`: Optional[Union[str, Path]] - whether to dump converted file, by default None

**Returns**

Dict[str, Any] - converted output

```
deepmd.utils.compat.convert_input_v1_v2(jdata: Dict[str, Any], warning: bool = True, dump: Optional[Union[str, Path]] = None) → Dict[str, Any]
```

```
deepmd.utils.compat.deprecate_numb_test(jdata: Dict[str, Any], warning: bool = True, dump: Optional[Union[str, Path]] = 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 - whether to show deprecation warning, by default True
- `dump`: Optional[Union[str, Path]] - 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: Optional[Union[str, Path]] = None) → Dict[str, Any]
```
deepmd.utils.compress module

deepmd.utils.compress.get_extra_side_embedding_net_variable(self, graph_def, type_side_suffix, variable_name, suffix)

deepmd.utils.compress.get_two_side_type_embedding(self, graph)

deepmd.utils.compress.get_type_embedding(self, graph)

deepmd.utils.compress.make_data(self, xx)

depdeepmd.utils.convert module

deepmd.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

depdeepmd.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

depdeepmd.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

depdeepmd.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.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.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.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.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.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.utils.convert.convert_dp20_to_dp21(fname: str)

deepmd.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.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.utils.convert.convert_to_21(input_model: str, output_model: str, version: Optional[str] = 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.utils.convert.detect_model_version(input_model: str)

Detect DP graph version.

Parameters

input_model

[str] filename of the input graph

deepmd.utils.data module

Alias for backward compatibility.

class deepmd.utils.data.DeepmdData(sys_path: str, set_prefix: str = 'set', shuffle_test: bool = True,
    type_map: Optional[List[str]] = None, optional_type_map: bool = True, modifier=None, trn_all_set: bool = False, sort_atoms: bool = True)

Bases: object

Class for a data system.

It loads data from hard disk, and maintains 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
DeePMD-kit

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.

Methods

add(key, ndof[, atomic, must, high_prec, ...]) Add a data item that to be loaded.

avg(key) Return the average value of an item.

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_atom_type() Get atom types.

get_batch(batch_size) Get a batch of data with batch_size frames.

get_data_dict() Get the data_dict.

get_natoms() Get number of atoms.

get_natoms_vec(ntypes) Get number of atoms and number of atoms in
different types.

get_ntypes() Number of atom types in the system.

get_numb_batch(batch_size, set_idx) Get the number of batches in a set.

get_numb_set() Get number of training sets.

get_sys_numb_batch(batch_size) Get the number of batches in the data system.

get_test([ntests]) Get the test data with ntests frames.

get_type_map() Get the type map.

date(key_out, key_in) 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

avg(key)
Return the average value of an item.

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_atom_type() → List[int]
Get atom types.

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

g_get_data_dict() → dict
Get the data_dict.

get_natoms()
Get number of atoms.

g_get_natsom_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

g_get_natypes() → int
Number of atom types in the system.

g_get_numb_batch(batch_size: int, set_idx: int) → int
Get the number of batches in a set.
DeePMD-kit

get_numb_set() \rightarrow \text{int}
Get number of training sets.

get_sys_numb_batch(batch_size: \text{int}) \rightarrow \text{int}
Get the number of batches in the data system.

get_test(ntests: int = -1) \rightarrow \text{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_type_map() \rightarrow \text{List[str]}
Get the type map.

reduce(key_out: \text{str}, key_in: \text{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

reset_get_batch()

deepmd.utils.data_system module

Alias for backward compatibility.

class deepmd.utils.data_system.DeepmdDataSystem(systems: \text{List[\text{str}]}, batch_size: \text{int}, test_size: \text{int},
rcut: Optional[\text{float}] = None, set_prefix: \text{str} =
‘set’, shuffle_test: \text{bool} = True, type_map:
Optional[\text{List[\text{str}]]} = None, optional_type_map:
\text{bool} = True, modifier=None, trn_all_set=False,
sys_probs=None,
auto_prob_style=’prob_sys_size’, sort_atoms:
\text{bool} = True)

Bases: \text{object}

Class for manipulating many data systems.
It is implemented with the help of DeepmdData

Attributes

\text{default_mesh}
Mesh for each system.

526 Chapter 18. Python API
Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(key, ndof[, atomic, must, high_prec, ...])</td>
<td>Add a data item that to be loaded.</td>
</tr>
<tr>
<td>add_dict(adict)</td>
<td>Add items to the data system by a dict.</td>
</tr>
<tr>
<td>get_batch(sys_idx)</td>
<td>Get a batch of data from the data systems.</td>
</tr>
<tr>
<td>get_batch_mixed()</td>
<td>Get a batch of data from the data systems in the mixed way.</td>
</tr>
<tr>
<td>get_batch_size()</td>
<td>Get the batch size.</td>
</tr>
<tr>
<td>get_batch_standard(sys_idx)</td>
<td>Get a batch of data from the data systems in the standard way.</td>
</tr>
<tr>
<td>get_nbatches()</td>
<td>Get the total number of batches.</td>
</tr>
<tr>
<td>get_nsystems()</td>
<td>Get the number of data systems.</td>
</tr>
<tr>
<td>get_ntypes()</td>
<td>Get the number of types.</td>
</tr>
<tr>
<td>get_sys(idx)</td>
<td>Get a certain data system.</td>
</tr>
<tr>
<td>get_sys_ntest(sys_idx)</td>
<td>Get number of tests for the currently selected system, or one defined by sys_idx.</td>
</tr>
<tr>
<td>get_test(sys_idx, n_test)</td>
<td>Get test data from the the data systems.</td>
</tr>
<tr>
<td>get_type_map()</td>
<td>Get the type map.</td>
</tr>
<tr>
<td>reduce(key_out, key_in)</td>
<td>Generate a new item from the reduction of another atom.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>compute_energy_shift</td>
</tr>
<tr>
<td>get_data_dict</td>
</tr>
<tr>
<td>print_summary</td>
</tr>
<tr>
<td>set_sys_probs</td>
</tr>
</tbody>
</table>

**add** (key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: Optional[List[int]] = None, repeat: int = 1, default: float = 0.0)

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.
**add_dict** (adict: dict) \(\rightarrow\) None

Add items to the data system by a dict. adict should have items like

```python
adict[key] = {
    "ndof": ndof, "atomic": atomic, "must": must, "high_prec": high_prec,
    "type_sel": type_sel, "repeat": repeat,
}
```

For the explanation of the keys see **add_compute_energy_shift** (rcond=None, key='energy')

**default_mesh**: List[ndarray]

Mesh for each system.

**get_batch** (sys_idx: Optional[int] = None) \(\rightarrow\) 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_mixed() \(\rightarrow\) dict

Get a batch of data from the data systems in the mixed way.

Returns

- **dict**
  - The batch data

**get_batch_size() \(\rightarrow\) int

Get the batch size.

**get_batch_standard** (sys_idx: Optional[int] = None) \(\rightarrow\) 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_data_dict** (ii: int = 0) \(\rightarrow\) dict
**get_nbatches() → int**
Get the total number of batches.

**get_nsystes() → int**
Get the number of data systems.

**get_ntypes() → int**
Get the number of types.

**get_sys(idx: int) → DeepmdData**
Get a certain data system.

**get_sys_ntest(sys_idx=None)**
Get number of tests for the currently selected system, or one defined by sys_idx.

**get_test(sys_idx: Optional[int] = None, n_test: int = -1)**
Get test data from the the data systems.

Parameters

- **sys_idx**
  The test data 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_type_map() → List[str]**
Get the type map.

**print_summary(name)**

**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

**set_sys_probs(sys_probs=None, auto_prob_style: str = 'prob_sys_size')**

**deepmd.utils.data_system.prob_sys_size_ext(keywords, nsystems, nbatch)**

**deepmd.utils.data_system.process_sys_probs(sys_probs, nbatch)**

**deepmd.utils.errors module**

**exception deepmd.utils.errors.GraphTooLargeError**
Bases: Exception
The graph is too large, exceeding protobuf's hard limit of 2GB.

**exception deepmd.utils.errors.GraphWithoutTensorError**
Bases: Exception
exception deepmd.utils.errors.OutOfMemoryError

Bases: Exception

This error is caused by out-of-memory (OOM).

**deepmd.utils.finetcune module**

**deepmd.utils.finetcune.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.utils.graph module**

**deepmd.utils.graph.get_attention_layer_nodes_from_graph_def(graph_def: 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.utils.graph.get_attention_layer_variables_from_graph_def(graph_def: 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.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.utils.graph.get_embedding_net_nodes_from_graph_def(graph_def: 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.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.utils.graph.get_embedding_net_variables_from_graph_def(graph_def: 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.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.utils.graph.get_extra_embedding_net_variables_from_graph_def(graph_def: GraphDef, suffix: str, extra_suffix: str, layer_size: int)

Get extra embedding net variables from the given tf.GraphDef object. The “extra embedding net” means the embedding net with only type embeddings input, which occurs in “se_atten_v2” and “se_a_ebd_v2” descriptor.

Parameters

graph_def
The input tf.GraphDef object

suffix
[str] The “common” suffix in the descriptor

extra_suffix
[str] This value depends on the value of “type_one_side”. It should always be “_one_side_ebd” or “_two_side_ebd”

layer_size
[int] The layer size of the embedding net

Returns

Dict
The extra embedding net variables with the given tf.GraphDef object

deepmd.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.utils.graph.get_fitting_net_nodes_from_graph_def(graph_def: 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
The fitting net nodes within the given tf.GraphDef object

deepmd.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.utils.graph.get_fitting_net_variables_from_graph_def(graph_def: 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 frozen model

deepmd.utils.graph.get_pattern_nodes_from_graph_def(graph_def: 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.utils.graph.get_tensor_by_name(model_file: str, tensor_name: str) → 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
depdeepmd.utils.graph.get_tensor_by_name_from_graph(graph: Graph, tensor_name: str) → 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
depdeepmd.utils.graph.get_tensor_by_type(node, data_type: dtype) → Tensor
    Get the tensor value within the given node according to the input data_type.

Parameters

node
    The given tensorflow graph node
data_type
    The data type of the node

Returns

tf.Tensor
    The tensor value of the given node
depdeepmd.utils.graph.get_type_embedding_net_nodes_from_graph_def(graph_def: 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
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

Get variables from the given tf.GraphDef object, with numpy array returns.

Parameters

graph_def
The input tf.GraphDef object
pattern
[str] The name of variable

Returns

np.ndarray
The numpy array of the variable

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.util.learning_rate module

class deepmd.util.learning_rate.LearningRateExp(start_lr: float, stop_lr: float = 5e-08,
decay_steps: int = 5000, decay_rate: float = 0.95)

Bases: object

The exponentially decaying learning rate.
The learning rate at step $t$ is given by

$$\alpha(t) = \alpha_0 \lambda^{t/\tau}$$
where $\alpha$ is the learning rate, $\alpha_0$ is the starting learning rate, $\lambda$ is the decay rate, and $\tau$ is the decay steps.

Parameters

- `start_lr`
  Starting learning rate $\alpha_0$
- `stop_lr`
  Stop learning rate $\alpha_1$
- `decay_steps`
  Learning rate decay every this number of steps $\tau$
- `decay_rate`
  The decay rate $\lambda$. If `stop_step` is provided in `build`, then it will be determined automatically and overwritten.

Methods

```python
build(global_step: int, stop_step: Optional[int] = None) → Tensor
Build the learning rate.
```

```
start_lr() → float
Get the start lr.
```

```
value(step: int) → float
Get the lr at a certain step.
```

`deepmd.utils.multi_init` module

```
deepmd.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
DeePMD-kit

pretrained_model
  [str] filename of the pretrained frozen multi-task united model

deepmd.utils.neighbor_stat module

class deepmd.utils.neighbor_stat.NeighborStat(ntyypes: int, rcut: float, one_type: bool = False)
    Bases: object
    Class for getting training data information.
    It loads data from DeepmdData object, and measures the data info, including nearest 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
    one_type
        [bool, optional, default=False] Treat all types as a single type.

    Methods

    get_stat(data) Get the data statistics of the training data, including nearest nbor distance between atoms, max nbor size of atoms.

    get_stat(data: DeepmdDataSystem) \to Tuple[float, List[int]]
        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
            A list with ntypes integers, denotes the actual achieved max sel
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]

$$
y = \mathcal{L}(x; w, b) = \begin{cases} 
\phi(x^T w + b) + x, & N_2 = N_1 \\
\phi(x^T w + b) + (x, x), & N_2 = 2N_1 \\
\phi(x^T w + b), & \text{otherwise}
\end{cases}
$$

where $x \in \mathbb{R}^{N_1}$ is the input vector and $y \in \mathbb{R}^{N_2}$ is the output vector. $w \in \mathbb{R}^{N_1 \times N_2}$ and $b \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable is True. $\phi$ 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 $\phi$

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.utils.network.embedding_net_rand_seed_shift(network_size)
deepmd.utils.network.one_layer(inputs, outputs_size, activation_fn=<function tanh>,
    precision=tf.float64, 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.utils.network.one_layer_rand_seed_shift()
deepmd.utils.network.variable_summaries(var: VariableV1, name: str)
    Attach a lot of summaries to a Tensor (for TensorBoard visualization).
    Parameters
    var [tf.Variable][description]
    name [str] variable name

deepmd.utils.pair_tab module

Alias for backward compatibility.
class deepmd.utils.pair_tab.PairTab(filename: str)
    Bases: object
    Pairwise tabulated potential.
    Parameters
    filename
        File name for the short-range tabulated potential. The table is a text data file with
        \((N_t + 1) \times N_t / 2 + 1\) columns. The first column is the distance between atoms. The
        second to the last columns are energies for pairs of certain types. For example we
        have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1
        correspondingly.

    Methods

    get() Get the serialized table.
    reinit(filename) Initialize the tabulated interaction.

    get() \rightarrow Tuple[array, array]
        Get the serialized table.
DeePMD-kit

```python
reinit(filename: str) → None
Initialize the tabulated interaction.

Parameters

filename
File name for the short-range tabulated potential. The table is a text data file with 
\((N_t + 1) * N_t / 2 + 1\) columns. The first column is the distance between atoms. 
The second to the last columns are energies for pairs of certain types. For example 
we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 
1-1 correspondingly.

deepmd.utils.parallel_op module

class deepmd.utils.parallel_op.ParallelOp(builder: Callable[[...], Tuple[Dict[str, Tensor], 
Tuple[Tensor]]], nthreads: Optional[int] = None, config: 
Optional[tf.ConfigProto] = None)

Bases: object
Run an op with data parallelism.

Parameters

builder
Callable[..., Tuple[Dict[str, tf.Tensor], Tuple[tf.Tensor]]] returns two ob-
jects: a dict which stores placeholders by key, and a tuple with the final op(s)
nthreads
[int, optional] the number of threads
config

Examples
```
```python
>>> from deepmd.env import tf
>>> from deepmd.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()))
```

Chapter 18. Python API
Methods

```python
generate(sess, feed) Returns a generator.

generate(sess: 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.utils.path module

Alias for backward compatibility.

class deepmd.utils.path.DPH5Path(path: str)

    Bases: DPath

    The path class to data system (DeepmdData) for HDF5 files.

    Parameters

        path
        [str] path

Notes

OS - HDF5 relationship:
    directory - Group file - Dataset

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>glob(pattern)</td>
<td>Search path using the glob pattern.</td>
</tr>
<tr>
<td>is_dir()</td>
<td>Check if self is directory.</td>
</tr>
<tr>
<td>is_file()</td>
<td>Check if self is file.</td>
</tr>
<tr>
<td>load_numpy()</td>
<td>Load NumPy array.</td>
</tr>
<tr>
<td>load_txt([dtype])</td>
<td>Load NumPy array from text.</td>
</tr>
<tr>
<td>rglob(pattern)</td>
<td>This is like calling DPath.glob() with **/ added in front of the given relative pattern.</td>
</tr>
</tbody>
</table>

18.2. deepmd package
**glob** (pattern: str) → List[DPPath]
Search path using the glob pattern.
Parameters
pattern
  [str] glob pattern
Returns
  List[DPPath]
  list of paths

**is_dir** () → bool
Check if self is directory.

**is_file** () → bool
Check if self is file.

**load_numpy** () → ndarray
Load NumPy array.
Returns
  np.ndarray
  loaded NumPy array

**load_txt** (dtype: Optional[dtype] = None, **kwargs) → ndarray
Load NumPy array from text.
Returns
  np.ndarray
  loaded NumPy array

**rglob** (pattern: str) → List[DPPath]
This is like calling **DPath.glob** () with **/ added in front of the given relative pattern.
Parameters
pattern
  [str] glob pattern
Returns
  List[DPPath]
  list of paths

**class deepmd.utils.path.DPOSPath** (path: str)
Bases: **DPath**
The OS path class to data system (DeepmdData) for real directories.
Parameters
path
  [str] path
### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>glob(pattern)</code></td>
<td>Search path using the glob pattern.</td>
</tr>
<tr>
<td><code>is_dir()</code></td>
<td>Check if self is directory.</td>
</tr>
<tr>
<td><code>is_file()</code></td>
<td>Check if self is file.</td>
</tr>
<tr>
<td><code>load_numpy()</code></td>
<td>Load NumPy array.</td>
</tr>
<tr>
<td><code>load_txt(**kwargs)</code></td>
<td>Load NumPy array from text.</td>
</tr>
<tr>
<td><code>rglob(pattern)</code></td>
<td>This is like calling <code>DPPath.glob()</code> with **/ added in front of the given relative pattern.</td>
</tr>
</tbody>
</table>

#### `glob(pattern: str) → List[DPPath]`
Search path using the glob pattern.

- **Parameters**
  - `pattern` [str] glob pattern

- **Returns**
  - List[DPPath]
    - list of paths

#### `is_dir()` → bool
Check if self is directory.

#### `is_file()` → bool
Check if self is file.

#### `load_numpy()` → ndarray
Load NumPy array.

- **Returns**
  - np.ndarray
    - loaded NumPy array

#### `load_txt(**kwargs) → ndarray`
Load NumPy array from text.

- **Returns**
  - np.ndarray
    - loaded NumPy array

#### `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
class deepmd.utils.path.DPPath(path: str)

Bases: ABC

The path class to data system (DeepmdData).

Parameters

path

[str] path

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>glob(pattern)</td>
<td>Search path using the glob pattern.</td>
</tr>
<tr>
<td>is_dir()</td>
<td>Check if self is directory.</td>
</tr>
<tr>
<td>is_file()</td>
<td>Check if self is file.</td>
</tr>
<tr>
<td>load_numpy()</td>
<td>Load NumPy array.</td>
</tr>
<tr>
<td>load_txt(**kwargs)</td>
<td>Load NumPy array from text.</td>
</tr>
<tr>
<td>rglob(pattern)</td>
<td>This is like calling DPPath.glob() with **) added in front of the given relative pattern.</td>
</tr>
</tbody>
</table>

abstract glob(pattern: str) → List[DPPath]

Search path using the glob pattern.

Parameters

pattern

[str] glob pattern

Returns

List[DPPath]

list of paths

abstract is_dir() → bool

Check if self is directory.

abstract is_file() → bool

Check if self is file.

abstract load_numpy() → ndarray

Load NumPy array.

Returns

np.ndarray

loaded NumPy array

abstract load_txt(**kwargs) → ndarray

Load NumPy array from text.

Returns

np.ndarray

loaded NumPy array

abstract rglob(pattern: str) → List[DPPath]

This is like calling DPPath.glob() with **) added in front of the given relative pattern.

Parameters
pattern

[\texttt{str}] glob pattern

Returns

\texttt{List[DPPath]}

list of paths

deepmd.utils.plugin module

Alias for backward compatibility.

class deepmd.utils.plugin.Plugin

Bases: \texttt{object}

A class to register and restore plugins.

Examples

```python
>>> plugin = Plugin()
>>> @plugin.register("xx")
    def xxx():
        pass
>>> print(plugin.plugins[\texttt{\'xx\'}])
```

Attributes

plugins

[\texttt{Dict[str,object]}] plugins

Methods

+ \texttt{get\_plugin(key)}: Visit a plugin by key.

+ \texttt{register(key)}: Register a plugin.

\texttt{get\_plugin(key)} \rightarrow \texttt{object}

Visit a plugin by key.

Parameters

key

[\texttt{str}] key of the plugin

Returns

\texttt{object}

the plugin

\texttt{register(key: str)} \rightarrow \texttt{Callable[[object], object]}

Register a plugin.

Parameters

key

[\texttt{str}] key of the plugin
Returns

Callable[[object], object]
decorator

class deepmd.utils.plugin.PluginVariant(*args, **kwargs)
Bases: object
A class to remove type from input arguments.

class deepmd.utils.plugin.VariantABCMeta(name, bases, namespace, **kwargs)
Bases: VariantMeta, ABCMeta

Methods

__call__(*args, **kwargs) Remove type and keys that starts with underline.
mro() Return a type’s method resolution order.
register(subclass) Register a virtual subclass of an ABC.

class deepmd.utils.plugin.VariantMeta
Bases: object

Methods

__call__(*args, **kwargs) Remove type and keys that starts with underline.

deepmd.utils.random module

Alias for backward compatibility.

deepmd.utils.random.choice(a: ndarray, p: Optional[ndarray] = None)
Generates a random sample from a given 1-D array.

Parameters

a
[ndarray] A random sample is generated from its elements.
p
[ndarray] The probabilities associated with each entry in a.

Returns

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

```python
np.ndarray
```
Arrays with results and their shapes.

```python
depdeepmd.utils.random.seed(val: Optional[int] = None)
```
Seed the generator.

Parameters

- `val` : [int] Seed.

```python
depdeepmd.utils.random.shuffle(x: ndarray)
```
Modify a sequence in-place by shuffling its contents.

Parameters

- `x` : [np.ndarray] The array or list to be shuffled.

**deepmd.utils.sess module**

```python
depdeepmd.utils.sess.run_sess(sess: Session,*args,**kwargs)
```
Run session with errors caught.

Parameters

- `sess` : [tf.Session] TensorFlow Session
- `*args` : Variable length argument list.
- `**kwargs` : Arbitrary keyword arguments.

Returns

- `Any` the result of sess.run()

**deepmd.utils.spin module**

```python
class deepmd.utils.spin.Spin(use_spin: Optional[List[bool]] = None, spin_norm: Optional[List[float]]
```
= None, virtual_len: Optional[List[float]] = None)
Bases: object

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

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build([reuse, suffix])</td>
<td>Build the computational graph for the spin.</td>
</tr>
<tr>
<td>get_ntypes_spin()</td>
<td>Returns the number of atom types which contain spin.</td>
</tr>
<tr>
<td>get_spin_norm()</td>
<td>Returns the list of magnitude of atomic spin for each atom type.</td>
</tr>
<tr>
<td>get_use_spin()</td>
<td>Returns the list of whether to use spin for each atom type.</td>
</tr>
<tr>
<td>get_virtual_len()</td>
<td>Returns the list of distance between real atom and virtual atom for each atom type.</td>
</tr>
</tbody>
</table>

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_spin_norm() → List[float]
Returns the list of magnitude of atomic spin for each atom type.

get_use_spin() → List[bool]
Returns the list of whether to use 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.utils.tabulate module

Bases: object

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 its 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

- **mathcal N**

- **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

Methods

**build**(min_nbor_dist, extrapolate, stride0, ...)
Build the tables for model compression.

**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
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

---

**deepmd.utils.type_embed module**

class `deepmd.utils.type_embed.TypeEmbedNet` (neuron: `List[int]` = [], resnet_dt: `bool` = `False`,
activation_function: `Optional[str]` = ‘tanh’, precision: 
str = ‘default’, trainable: `bool` = `True`, seed:
Optional[int] = `None`, uniform_seed: `bool` = `False`,
padding: `bool` = `False`, **kwargs)

Bases: `object`

Type embedding network.

Parameters

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 “relu”, “relu6”,
“softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.

precision

The precision of the embedding net parameters. Supported options are “default”,
“float16”, “float32”, “float64”, “bfloat16”.

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.
Methods

**build**(*ntypes*[, *reuse*, *suffix]*)

Build the computational graph for the descriptor.

Parameters

- **ntypes**: int
  Number of atom types.

- **reuse**: bool
  The weights in the networks should be reused when get the variable.

- **suffix**: str
  Name suffix to identify this descriptor

Returns

**embedded_types**

The computational graph for embedded types

**init_variables**(*graph*, *graph_def*[,...])

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**: str
  Name suffix to identify this descriptor

- **model_type**: str
  Indicator of whether this model is a compressed model

**deepmd.util.type_embed.embed_atom_type**(*ntypes*: int, *natom*: Tensor, *type_embedding*: 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**: int
  Number of types.

- **natom**: 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

- **type_embedding**: Tensor
  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]

deepmd.utils.weight_avg module

Alias for backward compatibility.

depdeepmd.utils.weight_avg.weighted_average(errors: List[Dict[str, Tuple[float, float]]]) → Dict
Compute weighted 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

18.2.2 Submodules

18.2.3 deepmd.calculator module

ASE calculator interface module.

class deepmd.calculator.DP(model: Union[str, Path], label: str = ‘DP’, type_dict: Optional[Dict[str, int]] = None, neighbor_list=None, **kwargs)

Bases: Calculator

Implementation of ASE deepmd calculator.

Implemented properties 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

```python
>>> from ase import Atoms
>>> from deepmd.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

```python
>>> from ase.optimize import BFGS
>>> dyn = BFGS(water)
>>> dyn.run(fmax=1e-6)
>>> print(water.get_positions())
```

Attributes

- directory
- label

Methods

- `band_structure()` Create band-structure object for plotting.
- `calculate(ates, properties, sys-
tem_changes)` Run calculation with deepmd model.
- `calculate_numerical_forces(ates[, d])` Calculate numerical forces using finite difference.
- `calculate_numerical_stress(ates[, d, voigt])` Calculate numerical stress using finite difference.
- `calculate_properties(ates, properties)` This method is experimental; currently for internal use.
- `check_state(ates[, tol])` Check for any system changes since last calculation.
- `get_magnetic_moments([ates])` Calculate magnetic moments projected onto atoms.
- `get_property(name[, atoms, allow_calculation])` Get the named property.
- `get_stresses([ates])` the calculator should return intensive stresses, i.e., such that stresses.sum(axis=0) == stress
- `read(label)` Read atoms, parameters and calculated properties from output file.
- `reset()` Clear all information from old calculation.
- `set(**kwargs)` Set parameters like set(key1=value1, key2=value2, ...).
- `set_label(label)` Set label and convert label to directory and prefix.

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

implemented_properties: ClassVar[List[str]] = ['energy', 'free_energy', 'forces', 'virial', 'stress']

  Properties calculator can handle (energy, forces, ...)

name = 'DP'

18.2.4 deepmd.common module

Collection of functions and classes used throughout the whole package.


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 whether the ndof keyword applies to per atom quantity or not, by default False

must
    [bool, optional] specifies if the *.npy data file must exist, by default False

high_prec
    [bool, optional] specifies if the 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 repeat 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

deepmd.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.common.clear_session()

Reset all state generated by DeePMD-kit.

deepmd.common.expand_sys_str(root_dir: Union[str, 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.gelu(x: Tensor) → 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


deepmd.common.gelu_tf(x: Tensor) → 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


deepmd.common.get_activation_func(activation_fn: Optional[_ACTIVATION]) → Optional[Callable[[Tensor], 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]
corresponding TF callable

Raises

RuntimeError
if unknown activation function is specified
deepmd.common.get_np_precision(precision: _PRECISION) → dtype
Get numpy precision constant from string.

Parameters

precision
  [._PRECISION] string name of numpy constant or default

Returns

np.dtype
  numpy precision constant

Raises

RuntimeError
  if string is invalid

deepmd.common.get_precision(precision: _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 a corresponding TF constant

deepmd.common.j_loader(filename: Union[str, 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.j_must_have(data: Dict[str, _DICT_VAL], key: str, deprecated_key: List[str] = []) → _DICT_VAL
Assert that supplied dictionary contains specified key.

Returns

_DICT_VAL
  value that was stored unde supplied key

Raises
DeePMD-kit

```
RuntimeError
if the key is not present
def make_default_mesh(pbc: bool, mixed_type: bool) -> 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

def safe_cast_tensor(input: Tensor, from_precision: DType, to_precision: DType) -> 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

def select_idx_map(atom_types: ndarray, select_types: ndarray) -> ndarray
    Build map of indices for element supplied element types from all atoms list.
Parameters
    atom_types
        [np.ndarray] array specifying 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
```
18.2.5 deepmd.env module

Module that sets tensorflow working environment and exports important constants.

```
deepmd.env.GLOBAL_ENER_FLOAT_PRECISION
alias of float64
```

```
deepmd.env.GLOBAL_NP_FLOAT_PRECISION
alias of float64
```

```
deepmd.env.global_cvt_2_ener_float(xx: Tensor) → 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.env.global_cvt_2_tf_float(xx: Tensor) → 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.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.
```

18.2.6 deepmd.lmp module

Register entry points for lammps-wheel.

```
deepmd.lmp.get_env(paths: List[Optional[str]]) → str
Get the environment variable from given paths.
```

```
deepmd.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.
```
DeePMD-kit

Returns

`list[str]`

The library path.

```
deepmd_lmp.get_op_dir() \rightarrow str
```

Get the directory of the deepmd-kit OP library.

## 18.3 deepmd_utils package

Utilization methods for DeePMD-kit.
The `__init__` module should not import any modules for performance.

### 18.3.1 Subpackages

**`deepmd_utils.entrypoints` package**

**Submodules**

**`deepmd_utils.entrypoints.doc` module**

Module that prints train input arguments docstrings.

```
deepmd_utils.entrypoints.doc.doc_train_input(*, out_type: str = ’rst’, **kwargs)
```

Print out training input arguments to console.

**`deepmd_utils.entrypoints.gui` module**

DP-GUI entrypoint.

```
deepmd_utils.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_utils.loggers package**

Module taking care of logging duties.

```python
deepmd_utils.loggers.set_log_handles(level: int, log_path: Optional[Path] = None, mpi_log: Optional[str] = None)
```

Set desired level for package loggers and add file handlers.

**Parameters**

- `level` [int] logging level
- `log_path` [Optional[Path]] 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:

<table>
<thead>
<tr>
<th></th>
<th>our notation</th>
<th>python logging</th>
<th>tensorflow cpp</th>
<th>OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>1/on/true/yes</td>
</tr>
<tr>
<td>info</td>
<td>20</td>
<td>20</td>
<td>1</td>
<td>0/off/false/no</td>
</tr>
<tr>
<td>warning</td>
<td>30</td>
<td>30</td>
<td>2</td>
<td>0/off/false/no</td>
</tr>
<tr>
<td>error</td>
<td>40</td>
<td>40</td>
<td>3</td>
<td>0/off/false/no</td>
</tr>
</tbody>
</table>

**References**

- https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U
- https://stackoverflow.com/questions/35869137/avoid-tensorflow-print-on-standard-error
**Submodules**

**deepmd_utils.loggers.loggers module**

Logger initialization for package.

```python
depdm_utils.loggers.loggers.set_log_handles(level: int, log_path: Optional[Path] = None, mpi_log: Optional[str] = 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:

<table>
<thead>
<tr>
<th></th>
<th>our notation</th>
<th>python logging</th>
<th>tensorflow cpp</th>
<th>OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>1/on/true/yes</td>
</tr>
<tr>
<td>info</td>
<td>20</td>
<td>20</td>
<td>1</td>
<td>0/off/false/no</td>
</tr>
<tr>
<td>warning</td>
<td>30</td>
<td>30</td>
<td>2</td>
<td>0/off/false/no</td>
</tr>
<tr>
<td>error</td>
<td>40</td>
<td>40</td>
<td>3</td>
<td>0/off/false/no</td>
</tr>
</tbody>
</table>

**References**

- https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U
- https://stackoverflow.com/questions/35869137/avoid-tensorflow-print-on-standard-error
deepmd_utils.model_format package


Bases: NativeOP

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor \( D^i \in \mathbb{R}^{M_1 \times M_2} \) is given by [1]

\[
D^i = (G^i)^T R^i (R^i)^T G^i_<
\]

where \( R^i \in \mathbb{R}^{N \times 4} \) is the coordinate matrix, and each row of \( R^i \) can be constructed as follows

\[
(R^i)_j = \begin{bmatrix}
\frac{s(r_{ji})}{s(r_{ji}) x_{ji}} \\
\frac{r_{ji}}{s(r_{ji}) y_{ji}} \\
\frac{r_{ji}}{s(r_{ji}) z_{ji}}
\end{bmatrix}
\]

where \( R_{ji} = R_j - R_i = (x_{ji}, y_{ji}, z_{ji}) \) is the relative coordinate and \( r_{ji} = ||R_{ji}|| \) is its norm. The switching function \( s(r) \) is defined as:

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

Each row of the embedding matrix \( G^i \in \mathbb{R}^{N \times M_1} \) consists of outputs of an embedding network \( N \) of \( s(r_{ji}) \):

\[
(G^i)_j = N(s(r_{ji}))
\]

\( G^i_< \in \mathbb{R}^{N \times M_2} \) takes first \( M_2 \) columns of \( G^i \). The equation of embedding network \( N \) can be found at deepmd.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**
  \([list[str]]\) sel[i] specifies the maximum number of type i atoms in the cut-off radius

- **neuron**
  \([list[int]]\) Number of neurons in each hidden layers of the embedding net \( N \)

- **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.

set_davg_zero
  Set the shift of embedding net input to zero.

activation_function
  The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.

precision
  The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

multi_task
  If the model has multi fitting nets to train.

spin
  The deepspin object.

References

[1]

Methods

__call__(*args, **kwargs) Forward pass in NumPy implementation.
call(coord_ext, atype_ext, nlist) Compute the descriptor.

cal_g(ss, ll)
call(coord_ext, atype_ext, nlist)
  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

Returns
descriptor
The descriptor. shape: nf x nloc x ng x axis_neuron
classmethod deserialize(data: dict) → DescriptSeA
serialize() → dict

depdm_utils.model_format.EmbeddingNet
alias of EN
class depdm_utils.model_format.EnvMat(rcut, rcut_smth)
Bases: NativeOP

Methods

__call__(*args, **kwargs) Forward pass in NumPy implementation.
call(coord_ext, atype_ext, nlist[, davg, dstd]) Compute the environment matrix.

deserialize
serialize

call(coord_ext: ndarray, atype_ext: ndarray, nlist: ndarray, davg: Optional[ndarray] = None, dstd: Optional[ndarray] = None) → 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 atom types. shape: nf x nall
davg
The data avg. shape: nt x nnei x 4
dstd
The inverse of data std. shape: nt x nnei x 4

Returns
env_mat
The environment matrix. shape: nf x nloc x nnei x 4
switch
The value of switch function. shape: nf x nloc x nnei
classmethod deserialize(data: dict) → EnvMat
serialize() → dict

deepmd_utils.model_format.FittingNet
    alias of FN

class deepmd_utils.model_format.FittingOutputDef(var_defs: List[OutputVariableDef])
    Bases: object
        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.

        Methods

            get_data
            keys

    get_data() → Dict[str, OutputVariableDef]

    keys()

class deepmd_utils.model_format.ModelOutputDef(fit_defs: FittingOutputDef)
    Bases: object
        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

        Methods

            get_data
            keys
            keys_derv_c
            keys_derv_r
            keys_outp
            keys_redu

    get_data(key: str) → Dict[str, OutputVariableDef]

    keys()
class deepmd_utils.model_format.NativeLayer(num_in, num_out, bias: bool = True, use_timestep: bool = False, activation_function: Optional[str] = None, resnet: bool = False, precision: str = 'float64')

Bases: 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.
- `activation_function` [str, optional] The activation function of the layer.
- `resnet` [bool, optional] Whether the layer is a residual layer.

Methods

- `__call__(*args, **kwargs)` Forward pass in NumPy implementation.
- `call(x)` Forward pass.
- `deserialize(data)` Deserialize the layer from a dict.
- `serialize()` Serialize the layer to a dict.

```
check_shape_consistency
check_type_consistency
dim_in
dim_out
```

call(x: ndarray) → ndarray

Forward pass.

Parameters

- `x` [np.ndarray] The input.

Returns

- `np.ndarray` The output.
check_shape_consistency()
check_type_consistency()

classmethod deserialize(data: dict) → NativeLayer
    Deserialize the layer from a dict.
    Parameters
    data
        [dict] The dict to deserialize from.
    dim_in() → int
    dim_out() → int
    serialize() → dict
        Serialize the layer to a dict.
        Returns
        dict
            The serialized layer.

deepmd_utils.model_format.NativeNet
    alias of NN

class deepmd_utils.model_format.NativeOP
    Bases: ABC
    The unit operation of a native model.

Methods

_call_(*args, **kwargs)  Forward pass in NumPy implementation.
call(*args, **kwargs)  Forward pass in NumPy implementation.
call(*args, **kwargs)
    Forward pass in NumPy implementation.

class deepmd_utils.model_format.NetworkCollection
    (ndim: int, ntypes: int, network_type: str =
        'network', networks: ~typing.List[~typing.Union[~deepmd_utils.model_format.network.make_multilayer_network.<locals>.NN,
            dict]] = [])

    Bases: object

    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.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>check_completeness()</td>
<td>Check whether the collection is complete.</td>
</tr>
<tr>
<td>deserialize(data)</td>
<td>Deserialize the networks from a dict.</td>
</tr>
<tr>
<td>serialize()</td>
<td>Serialize the networks to a dict.</td>
</tr>
</tbody>
</table>

NETWORK_TYPE_MAP: ClassVar[Dict[str, type]] = {'embedding_network': <class
'deepmd_utils.model_format.network.make_embedding_network.<locals>.EN'>,
'fitting_network': <class
'deepmd_utils.model_format.network.make_fitting_network.<locals>.FN'>, 'network':
<class 'deepmd_utils.model_format.network.make_multilayer_network.<locals>.NN'>}

check_completeness()
Check whether the collection is complete.

Raises

RuntimeError
If the collection is incomplete.

classmethod deserialize(data: dict) → NetworkCollection
Deserialize the networks from a dict.

Parameters

data [dict] The dict to deserialize from.

serialize() → dict
Serialize the networks to a dict.

Returns

dict
The serialized networks.

class deepmd_utils.model_format.OutputVariableDef(name: str, shape: List[int], reduciable: bool = False, differentiable: bool = False, atomic: bool = True)

Bases: object

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], polarizability should be [3,3].
If the variable is reduced.

differentiable
If the variable is differentiated with respect to coordinates of atoms and cell tensor (pbc case). Only reduciable variable are differentiable.

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.

def get_deriv_name(name: str) -> Tuple[str, str]

def get_reduce_name(name: str) -> str

def 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.

def make_embedding_network(T_Network, T_NetworkLayer)

def make_fitting_network(T_EmbeddingNet, T_Network, T_NetworkLayer)

def make_multilayer_network(T_NetworkLayer, ModuleBase)

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.

def save_dp_model(filename: str, model_dict: dict, extra_info: Optional[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.
    extra_info
        [dict, optional] Extra meta information to save.

def traverse_model_dict(model_obj, callback: callable, is_variable: bool = False)
    Traverse a model dict and call callback on each variable.
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_utils.model_format.common module

class deepmd_utils.model_format.common.NativeOP
    Bases: ABC
    The unit operation of a native model.

Methods

__call__(*args, **kwargs) Forward pass in NumPy implementation.
call(*args, **kwargs) Forward pass in NumPy implementation.

call(*args, **kwargs)
    Forward pass in NumPy implementation.

deepmd_utils.model_format.env_mat module

class deepmd_utils.model_format.env_mat.EnvMat(rcut, rcut_smth)
    Bases: NativeOP

Methods

__call__(*args, **kwargs) Forward pass in NumPy implementation.
call(coord_ext, atype_ext, nlist[, davg, dstd]) Compute the environment matrix.

deserialize
serialize

call(coord_ext: ndarray, atype_ext: ndarray, nlist: ndarray, davg: Optional[ndarray] = None, dstd: Optional[ndarray] = None) → 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 atom types. shape: nf x nall

davg
The data avg. shape: nt x nnei x 4

dstd
The inverse of data std. shape: nt x nnei x 4

Returns
env_mat
The environment matrix. shape: nf x nloc x nnei x 4

switch
The value of switch function. shape: nf x nloc x nnei

classmethod deserialize(data: dict) → EnvMat

serialize() → dict

deepmd_utils.model_format.env_mat.compute_smooth_weight(distance: ndarray, rmin: float, rmax: float)

Compute smooth weight for descriptor elements.

deepmd_utils.model_format.network module

Native DP model format for multiple backends.
See issue #2982 for more information.

class deepmd_utils.model_format.network.Counter
    Bases: object
    A callable counter.

Examples

```python
>>> counter = Counter()
>>> counter()
0
>>> counter()
1
```
Methods

__call__()  
Call self as a function.

depdm_utils.model_format.network.EmbeddingNet  
alias of EN

depdm_utils.model_format.network.FittingNet  
alias of FN

class deepdm_utils.model_format.network.NativeLayer(num_in, num_out, bias: bool = True,  
use_timestep: bool = False,  
activation_function: Optional[str] = None,  
resnet: bool = False, precision: str =  
‘float64’)  

Bases: 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  
activation_function  
[str, optional] The activation function of the layer.
resnet  
[bool, optional] Whether the layer is a residual layer.

Methods

__call__(*args, **kwargs)  
Forward pass in NumPy implementation.
call(x)  
Forward pass.
deserialize(data)  
Deserialize the layer from a dict.
serialize()  
Serialize the layer to a dict.

check_shape_consistency
check_type_consistency
dim_in
dim_out

call(x: ndarray) → ndarray  
Forward pass.

Parameters  
x  
[np.ndarray] The input.
DeePMD-kit

Returns

np.ndarray
The output.

check_shape_consistency()

check_type_consistency()

classmethod deserialize(data: dict) → NativeLayer
    Deserialize the layer from a dict.

Parameters

    data [dict] The dict to deserialize from.

dim_in() → int
dim_out() → int

serialize() → dict
    Serialize the layer to a dict.

Returns

    dict
    The serialized layer.

deepmd_utils.model_format.network.NativeNet
    alias of NN

class deepmd_utils.model_format.network.NetworkCollection(ndim: int, ntypes: int, network_type: str = 'network', networks: ~typing.List[~typing.Union[~deepmd_utils.model_format.network.make_multilayer_network.<locals>.NN, dict]] = [])

Bases: object

A collection of networks for multiple elements.

The number of dimensions 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.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>check_completeness()</code></td>
<td>Check whether the collection is complete.</td>
</tr>
<tr>
<td><code>deserialize(data)</code></td>
<td>Deserialize the networks from a dict.</td>
</tr>
<tr>
<td><code>serialize()</code></td>
<td>Serialize the networks to a dict.</td>
</tr>
</tbody>
</table>

**NETWORK_TYPE_MAP**: `ClassVar[Dict[str, type]] = {'embedding_network': <class 'deepmd_utils.model_format.network.make_embedding_network.<locals>.EN'>, 'fitting_network': <class 'deepmd_utils.model_format.network.make_fitting_network.<locals>.FN'>, 'network': <class 'deepmd_utils.model_format.network.make_multilayer_network.<locals>.NN'>}

`check_completeness()`
Check whether the collection is complete.

Raises

`RuntimeError`
If the collection is incomplete.

**class method** `deserialize(data: dict) → NetworkCollection`
Deserializes the networks from a dict.

Parameters

- `data` `[dict] The dict to deserialize from.]

`serialize() → dict`
Serialize the networks to a dict.

Returns

- `dict` The serialized networks.

**deepmd_utils.model_format.network.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_utils.model_format.network.make_embedding_network(T_Network, T_NetworkLayer)**

**deepmd_utils.model_format.network.make_fitting_network(T_EmbeddingNet, T_Network, T_NetworkLayer)**

**deepmd_utils.model_format.network.make_multilayer_network(T_NetworkLayer, ModuleBase)**

**deepmd_utils.model_format.network.save_dp_model(filename: str, model_dict: dict, extra_info: Optional[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.
extra_info
    [dict, optional] Extra meta information to save.

deepmd_utils.model_format.network.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.

deepmd_utils.model_format.output_def module

class deepmd_utils.model_format.output_def.FittingOutputDef(var_defs: List[OutputVariableDef])

Bases: object

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.

Methods

get_data

keys

get_data() → Dict[str, OutputVariableDef]

keys()
class `deepmd_utils.model_format.output_def.ModelOutputDef`(`fit_defs`: `FittingOutputDef`)

Bases: `object`

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

Methods

- `get_data`(key: `str`) → `Dict[str, OutputVariableDef]`
- `keys()`
- `keys_derv_c()`
- `keys_derv_r()`
- `keys_outp()`
- `keys_redu()`

class `deepmd_utils.model_format.output_def.OutputVariableDef`(`name`: `str`, `shape`: `List[int]`,
  `reducible`: `bool = False`,
  `differentiable`: `bool = False`,
  `atomic`: `bool = True`)

Bases: `object`

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], polarizability should be [3,3].

- `reducible`  
  If the variable is reduced.
differentiable

If the variable is differentiated with respect to coordinates of atoms and cell tensor (pbc case). Only reduciable variable are differentiable.

deepmd_utils.model_format.output_def.check_shape(shape: List[int], def_shape: List[int])
    Check if the shape satisfies the defined shape.

deepmd_utils.model_format.output_def.check_var(var, var_def)

deepmd_utils.model_format.output_def.do_derivative(def_outp: FittingOutputDef) →
    Tuple[Dict[str, OutputVariableDef], Dict[str, OutputVariableDef]]

deepmd_utils.model_format.output_def.do_reduce(def_outp: FittingOutputDef) → Dict[str, OutputVariableDef]

deepmd_utils.model_format.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.

deepmd_utils.model_format.output_def.get_deriv_name(name: str) → Tuple[str, str]

deepmd_utils.model_format.output_def.get_reduce_name(name: str) → str

deepmd_utils.model_format.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_utils.model_format.se_e2_a module

class deepmd_utils.model_format.se_e2_a.DescrptSeA(rcut: float, rcut_smth: float, sel: List[str],
    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]] = [],
    set_davg_zero: bool = False,

    Bases: NativeOP

    DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

    The descriptor \(D^i \in \mathbb{R}^{M_1 \times M_2}\) is given by [1]

    \[D^i = (G^i)^T R^i (R^i)^T G^i <\]

    where \(R^i \in \mathbb{R}^{N \times 4}\) is the coordinate matrix, and each row of \(R^i\) can be constructed as follows

    \[
    (R^i)_j = \begin{bmatrix}
    \frac{s(r_{ji})}{s(r_{ji})x_{ji}} \\
    \frac{r_{ji}}{s(r_{ji})y_{ji}} \\
    \frac{s(r_{ji})y_{ji}}{r_{ji}} \\
    \end{bmatrix}
    \]
where $R_{ji} = R_j - R_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \|R_{ji}\|$ is its norm. The switching function $s(r)$ is defined as:
\[
s(r) = \begin{cases} 
\frac{1}{r}, & r < r_s \\
\frac{1}{r} \left( \left( \frac{r-r_s}{r_c-r_s} \right)^3 - 6 \left( \frac{r-r_s}{r_c-r_s} \right)^2 + 15 \frac{r-r_s}{r_c-r_s} - 10 \right) + 1, & r_s \leq r < r_c \\
0, & r \geq r_c
\end{cases}
\]

Each row of the embedding matrix $G^i \in \mathbb{R}^{N \times M_1}$ consists of outputs of an embedding network $\mathcal{N}$ of $s(r_{ji})$:
\[
(G^i)_j = \mathcal{N}(s(r_{ji}))
\]

$G^i_j \in \mathbb{R}^{N \times M_2}$ takes first $M_2$ columns of $G^i$. The equation of embedding network $\mathcal{N}$ can be found at `deepmd.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[str]]`sel[i] specifies the maximum 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: $y = x + dt \ast \phi(Wx + b)$

- `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 “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”, “None”, “none”.

- `precision`
  The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

- `multi_task`
  If the model has multi fitting nets to train.

- `spin`
  The deepspin object.
References

[1]

Methods

<table>
<thead>
<tr>
<th><strong>call</strong>(*args, **kwargs)</th>
<th>Forward pass in NumPy implementation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>call(coord_ext, atype_ext, nlist)</td>
<td>Compute the descriptor.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cal_g</th>
<th>deserialize</th>
<th>serialize</th>
</tr>
</thead>
</table>

cal_g(ss, ll)

call(coord_ext, atype_ext, nlist)

Compute the descriptor.

Parameters

coord_ext

The extended coordinates of atoms. shape: nf x (nallx3)

atype_ext

The extended atom types. shape: nf x nall

nlist

The neighbor list. shape: nf x nloc x nnei

Returns

descriptor

The descriptor. shape: nf x nloc x ng x axis_neuron

classmethod deserialize(data: dict) → DescriptSeA

serialize() → dict

deepmd_utils.utils package

Submodules

deepmd_utils.utils.argcheck module

class deepmd_utils.utils.argcheck.ArgsPlugin

Bases: object
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_all_argument</code></td>
<td>Get all arguments.</td>
</tr>
<tr>
<td><code>register</code></td>
<td>Register a descriptor argument plugin.</td>
</tr>
</tbody>
</table>

**get_all_argument** (exclude_hybrid: *bool* = False) → List[Argument]

Get all arguments.

Parameters

- **exclude_hybrid**
  - [bool] exclude hybrid descriptor to prevent circular calls

Returns

- List[Argument] all arguments

**register** (name: *str*, alias: Optional[List[str]] = None) → Callable([], List[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**

```python
>>> some_plugin = ArgsPlugin()
>>> @some_plugin.register("some_descrpt")
  def descrpt_some_descrpt_args():
    return []
```

depend_utils.utils.argcheck.descrpt_hybrid_args()
depend_utils.utils.argcheck.descrpt_local_frame_args()
depend_utils.utils.argcheck.descrpt_se_a_args()
depend_utils.utils.argcheck.descrpt_se_a_ebd_v2_args()
depend_utils.utils.argcheck.descrpt_se_a_mask_args()
depend_utils.utils.argcheck.descrpt_se_a_tpe_args()
depend_utils.utils.argcheck.descrpt_se_atten_args()
depend_utils.utils.argcheck.descrpt_se_atten_common_args()
depend_utils.utils.argcheck.descrpt_se_atten_v2_args()
DeePMD-kit

deepmd_utils.utils.argcheck.descrpt_se_r_args()
deepmd_utils.utils.argcheck.descrpt_se_t_args()
deepmd_utils.utils.argcheck.descrpt_variant_type_args(exclude_hybrid: bool = False) → Variant
deepmd_utils.utils.argcheck.fitting_dipole()
deepmd_utils.utils.argcheck.fitting_dos()
deepmd_utils.utils.argcheck.fitting_ener()
deepmd_utils.utils.argcheck.fitting_polar()
deepmd_utils.utils.argcheck.fitting_variant_type_args()
deepmd_utils.utils.argcheck.frozen_model_args() → Argument
deepmd_utils.utils.argcheck.gen_args(**kwargs) → List[Argument]
deepmd_utils.utils.argcheck.gen_doc(*, make_anchor=True, make_link=True, **kwargs)
deepmd_utils.utils.argcheck.gen_json(**kwargs)
deepmd_utils.utils.argcheck.learning_rate_args()
deepmd_utils.utils.argcheck.learning_rate_dict_args()
deepmd_utils.utils.argcheck.learning_rate_exp()
deepmd_utils.utils.argcheck.learning_rate_variant_type_args()
deepmd_utils.utils.argcheck.limit_pref(item)
deepmd_utils.utils.argcheck.linear_ener_model_args() → Argument
deepmd_utils.utils.argcheck.list_to_doc(xx)
deepmd_utils.utils.argcheck.loss_args()
deepmd_utils.utils.argcheck.loss_dict_args()
deepmd_utils.utils.argcheck.loss_dos()
deepmd_utils.utils.argcheck.loss_ener()
deepmd_utils.utils.argcheck.loss_ener_spin()
deepmd_utils.utils.argcheck.loss_tensor()
deepmd_utils.utils.argcheck.loss_variant_type_args()
deepmd_utils.utils.argcheck.make_index(keys)
deepmd_utils.utils.argcheck.make_link(content, ref_key)
deepmd_utils.utils.argcheck.mixed_precision_args()
deepmd_utils.utils.argcheck.model_args(exclude_hybrid=False)
deepmd_utils.utils.argcheck.model_compression()
deepmd_utils.utils.argcheck.model_compression_type_args()
deepmd_utils.utils.argcheck.modifier_dipole_charge()
deepmd_utils.utils.argcheck.modifier_variant_type_args()
deepmd_utils.utils.argcheck.multi_model_args() \rightarrow \text{Argument}
deepmd_utils.utils.argcheck.normalize(data)
deepmd_utils.utils.argcheck.normalize_data_dict(data_dict)
deepmd_utils.utils.argcheck.normalize_fitting_net_dict(fitting_net_dict)
deepmd_utils.utils.argcheck.normalize_fitting_weight(fitting_keys, data_keys, fitting_weight=None)
deepmd_utils.utils.argcheck.normalize_learning_rate_dict(fitting_keys, learning_rate_dict)
deepmd_utils.utils.argcheck.normalize_learning_rate_dict_with_single_learning_rate(fitting_keys, learning_rate)
deepmd_utils.utils.argcheck.normalize_loss_dict(fitting_keys, loss_dict)
deepmd_utils.utils.argcheck.normalize_multi_task(data)
deepmd_utils.utils.argcheck.pairtab_model_args() \rightarrow \text{Argument}
deepmd_utils.utils.argcheck.pairwise_dprc() \rightarrow \text{Argument}
deepmd_utils.utils.argcheck.spin_args()
deepmd_utils.utils.argcheck.standard_model_args() \rightarrow \text{Argument}
deepmd_utils.utils.argcheck.start_pref(item, label=None, abbr=None)
deepmd_utils.utils.argcheck.training_args()
deepmd_utils.utils.argcheck.training_data_args()
deepmd_utils.utils.argcheck.type_embedding_args()
deepmd_utils.utils.argcheck.validation_data_args()

\texttt{deepmd_utils.utils.argcheck_nvnmnd module}

deepmd_utils.utils.argcheck_nvnmnd.nvnmd_args()
class `deepmd_utils.utils.batch_size.AutoBatchSize` (initial_batch_size: int = 1024, factor: float = 2.0)

Bases: 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

Methods

- `execute(callable, start_index, natoms)` Execute a method with given batch size.
- `execute_all(callable, total_size, natoms, ...)` Execute a method with all given data.
- `is_gpu_available()` Check if GPU is available.
- `is_oom_error(e)` Check if the exception is an OOM error.

execute(callable: Callable, start_index: int, natoms: int) \[\rightarrow\] Tuple[int, tuple]

Execute 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
DeePMD-kit

natom

[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

execute_all(callable: Callable, total_size: int, natoms: int, *args, **kwargs) → Tuple[ndarray]
Execute a method with all given data.

Parameters

callable
[Callable] The method should accept *args and **kwargs as input and return the similar array.
total_size
[int] Total size
natom
[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.utils.compat module**

Module providing compatibility between 0.x.x and 1.x.x input versions.

```python
deepmd_utils.utils.compat.convert_input_v0_v1(jdata: Dict[str, Any], warning: bool = True, dump: Optional[Union[str, Path]] = 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

```python
deepmd_utils.utils.compat.convert_input_v1_v2(jdata: Dict[str, Any], warning: bool = True, dump: Optional[Union[str, Path]] = None) → Dict[str, Any]
```

```python
deepmd_utils.utils.compat.deprecate_numb_test(jdata: Dict[str, Any], warning: bool = True, dump: Optional[Union[str, Path]] = 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

```python
deepmd_utils.utils.compat.remove_decay_rate(jdata: Dict[str, Any])
```

Convert decay_rate to stop_lr.

**Parameters**

- **jdata**: Dict[str, Any]
  - input data
```python
deePMD-kit

deepmd_utils.utils.compat.update_deepmd_input(jdata: Dict[str, Any], warning: bool = True, dump: Optional[Union[str, Path]] = None) → Dict[str, Any]

deepmd_utils.utils.data module

class deepmd_utils.utils.data.DeepmdData(sys_path: str, set_prefix: str = 'set', shuffle_test: bool = True, type_map: Optional[List[str]] = None, optional_type_map: bool = True, modifier=None, trn_all_set: bool = False, sort_atoms: bool = True)

    Bases: object

    Class for a data system.
    It loads data from hard disk, and maintains 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 feded to descriptors except mixed types.

18.3. deepmd_utils package 587
```
**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>add</strong>(key, ndof[, atomic, must, high_prec, ...])</td>
<td>Add a data item that to be loaded.</td>
</tr>
<tr>
<td><strong>avg</strong>(key)</td>
<td>Return the average value of an item.</td>
</tr>
<tr>
<td><strong>check_batch_size</strong>(batch_size)</td>
<td>Check if the system can get a batch of data with batch_size frames.</td>
</tr>
<tr>
<td><strong>check_test_size</strong>(test_size)</td>
<td>Check if the system can get a test dataset with test_size frames.</td>
</tr>
<tr>
<td><strong>get_atom_type</strong>()</td>
<td>Get atom types.</td>
</tr>
<tr>
<td><strong>get_batch</strong>(batch_size)</td>
<td>Get a batch of data with batch_size frames.</td>
</tr>
<tr>
<td><strong>get_data_dict</strong>()</td>
<td>Get the data_dict.</td>
</tr>
<tr>
<td><strong>get_natoms</strong>()</td>
<td>Get number of atoms.</td>
</tr>
<tr>
<td><strong>get_natoms_vec</strong>(ntypes)</td>
<td>Get number of atoms and number of atoms in different types.</td>
</tr>
<tr>
<td><strong>get_numb_batch</strong>(batch_size, set_idx)</td>
<td>Get the number of batches in a set.</td>
</tr>
<tr>
<td><strong>get_numb_set</strong>()</td>
<td>Get number of training sets.</td>
</tr>
<tr>
<td><strong>get_sys_numb_batch</strong>(batch_size)</td>
<td>Get the number of batches in the data system.</td>
</tr>
<tr>
<td><strong>get_test</strong>(ntests)</td>
<td>Get the test data with ntests frames.</td>
</tr>
<tr>
<td><strong>get_type_map</strong>()</td>
<td>Get the type map.</td>
</tr>
<tr>
<td><strong>reduce</strong>(key_out, key_in)</td>
<td>Generate a new item from the reduction of another atom.</td>
</tr>
</tbody>
</table>

**add**(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: Optional[List[int]] = None, repeat: int = 1, default: float = 0.0, dtype: Optional[dtype] = None)

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

**avg**(key)

Return the average value of an item.

**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_atom_type**( ) → List[int]

Get atom types.

**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_data_dict**( ) → dict

Get the data_dict.

**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

**get_ntypes**( ) → int

Number of atom types in the system.

**get_numb_batch**(batch_size: int, set_idx: int) → int

Get the number of batches in a set.

**get_numb_set**( ) → int

Get number of training sets.

**get_sys_numb_batch**(batch_size: int) → int

Get the number of batches in the data system.

**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.
DeePMD-kit

get_type_map() → List[str]
Get the type map.

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

reset_get_batch()

deepmd_utils.utils.data_system module

class deepmd_utils.utils.data_system.DeepmdDataSystem(systems: List[str], batch_size: int,
test_size: int, rcut: Optional[float] = None,
set_prefix: str = 'set', shuffle_test: bool =
True, type_map: Optional[List[str]] =
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)
Bases: object
Class for manipulating many data systems.
It is implemented with the help of DeepmdData
Attributes

default_mesh
Mesh for each system.
Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(key, ndof[, atomic, must, high_prec, ...])</code></td>
<td>Add a data item that to be loaded.</td>
</tr>
<tr>
<td><code>add_dict(adict)</code></td>
<td>Add items to the data system by a dict.</td>
</tr>
<tr>
<td><code>get_batch([sys_idx])</code></td>
<td>Get a batch of data from the data systems.</td>
</tr>
<tr>
<td><code>get_batch_mixed()</code></td>
<td>Get a batch of data from the data systems in the mixed way.</td>
</tr>
<tr>
<td><code>get_batch_size()</code></td>
<td>Get the batch size.</td>
</tr>
<tr>
<td><code>get_batch_standard([sys_idx])</code></td>
<td>Get a batch of data from the data systems in the standard way.</td>
</tr>
<tr>
<td><code>get_nbatches()</code></td>
<td>Get the total number of batches.</td>
</tr>
<tr>
<td><code>get_nsystems()</code></td>
<td>Get the number of data systems.</td>
</tr>
<tr>
<td><code>get_ntypes()</code></td>
<td>Get the number of types.</td>
</tr>
<tr>
<td><code>get_sys(idx)</code></td>
<td>Get a certain data system.</td>
</tr>
<tr>
<td><code>get_sys_ntrials(sys_idx)</code></td>
<td>Get number of tests for the currently selected system, or one defined by sys_idx.</td>
</tr>
<tr>
<td><code>get_test([sys_idx, n_test])</code></td>
<td>Get test data from the the data systems.</td>
</tr>
<tr>
<td><code>get_type_map()</code></td>
<td>Get the type map.</td>
</tr>
<tr>
<td><code>reduce(key_out, key_in)</code></td>
<td>Generate a new item from the reduction of another atom.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>compute_energy_shift</code></td>
<td></td>
</tr>
<tr>
<td><code>get_data_dict</code></td>
<td></td>
</tr>
<tr>
<td><code>print_summary</code></td>
<td></td>
</tr>
<tr>
<td><code>set_sys_probs</code></td>
<td></td>
</tr>
</tbody>
</table>

`add(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: Optional[List[int]] = None, repeat: int = 1, default: float = 0.0)`

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

- **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.
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 explanation of the keys see add

compute_energy_shift(rcond=None, key='energy')

property default_mesh: List[ndarray]
Mesh for each system.

get_batch(sys_idx: Optional[int] = 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_mixed() → dict
Get a batch of data from the data systems in the mixed way.

Returns
    dict
        The batch data.

get_batch_size() → int
Get the batch size.

get_batch_standard(sys_idx: Optional[int] = 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_data_dict(ii: int = 0) → dict
get_nbatches() \rightarrow \text{int}

Get the total number of batches.

get_nsystems() \rightarrow \text{int}

Get the number of data systems.

get_ntypes() \rightarrow \text{int}

Get the number of types.

get_sys(idx: \text{int}) \rightarrow 
DeepmdData

Get a certain data system.

get_sys_n tests(sys_idx=None)

Get number of tests for the currently selected system, or one defined by sys_idx.

get_test(sys_idx: \text{Optional[int]} = \text{None}, n_test: \text{int} = -1)

Get test data from the data systems.

Parameters

sys_idx
The test data 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_type_map() \rightarrow \text{List[str]}

Get the type map.

print_summary(name)

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

set_sys_probs(sys_probs=None, auto_prob_style: \text{str} = \text{'prob_sys_size'})

deepmd_utils.utils.data_system.prob_sys_size_ext(keywords, nsystems, nbatch)

deepmd_utils.utils.data_system.process_sys_probs(sys_probs, nbatch)

deepmd_utils.utils.errors module

default exception deepmd_utils.utils.errors.OutOfMemoryError

Bases: Exception

This error is caused by out-of-memory (OOM).
DeepMD-kit

**deepmd_utils.utils.model_stat module**

`deepmd_utils.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

  ```python
  all_stat[key][sys_idx][batch_idx][frame_idx]
  ```

  else merge_sys == True can be accessed by

  ```python
  all_stat[key][batch_idx][frame_idx]
  ```

`deepmd_utils.utils.model_stat.merge_sys_stat(all_stat)`

**deepmd_utils.utils.pair_tab module**

**class deepmd_utils.utils.pair_tab.PairTab(filename: str)**

**Bases:** `object`

Pairwise tabulated potential.

**Parameters**

- **filename**  
File name for the short-range tabulated potential. The table is a text data file with \((N_t + 1) \times N_t / 2 + 1\) columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

**Methods**

```python
get()  
Get the serialized table.
```  
```python
reinit(filename)  
Initialize the tabulated interaction.
```  
```python
get()  
Tuple[array, array]  
Get the serialized table.
```
reinit(filename: str) → None
Initialize the tabulated interaction.

Parameters

filename
File name for the short-range tabulated potential. The table is a text data file with
\((N_t + 1) \times N_t / 2 + 1\) columns. The first column is the distance between atoms.
The second to the last columns are energies for pairs of certain types. For example
we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and
1-1 correspondingly.

depm_utils.utils.path module

class deepmd_utils.utils.path.DPH5Path(path: str)
Bases: DPath
The path class to data system (DeepmdData) for HDF5 files.

Parameters

path
[str] path

Notes

OS - HDF5 relationship:
directory - Group file - Dataset

Methods

<table>
<thead>
<tr>
<th>glob(pattern)</th>
<th>Search path using the glob pattern.</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_dir()</td>
<td>Check if self is directory.</td>
</tr>
<tr>
<td>is_file()</td>
<td>Check if self is file.</td>
</tr>
<tr>
<td>load_numpy()</td>
<td>Load NumPy array.</td>
</tr>
<tr>
<td>load_txt(dtype)</td>
<td>Load NumPy array from text.</td>
</tr>
<tr>
<td>rglob(pattern)</td>
<td>This is like calling DPath.glob() with **/added in front of the given relative pattern.</td>
</tr>
</tbody>
</table>

glob(pattern: str) → List[DPPath]
Search path using the glob pattern.

Parameters

pattern
[ str ] glob pattern

Returns

List[DPPath]
list of paths

is_dir() → bool
Check if self is directory.
**Class**: deepmd_util.utils.path.DPOSPath

**Bases**: DPath

The OS path class to data system (DeepmdData) for real directories.

**Parameters**

- path [str] path

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>glob(pattern)</code></td>
<td>Search path using the glob pattern.</td>
</tr>
<tr>
<td><code>is_dir()</code></td>
<td>Check if self is directory.</td>
</tr>
<tr>
<td><code>is_file()</code></td>
<td>Check if self is file.</td>
</tr>
<tr>
<td><code>load_numpy()</code></td>
<td>Load NumPy array.</td>
</tr>
<tr>
<td><code>load_txt(**kwargs)</code></td>
<td>Load NumPy array from text.</td>
</tr>
<tr>
<td><code>rglob(pattern)</code></td>
<td>This is like calling DPath.glob() with **/ added in front of the given relative pattern.</td>
</tr>
</tbody>
</table>

**Class Methods**

- `glob(pattern)` → List[DPPath]
  - Search path using the glob pattern.
  - **Parameters**
    - pattern [str] glob pattern
  - Returns
    - List[DPPath] list of paths
Returns

List[DPath]
list of paths

is_dir() → bool
Check if self is directory.

is_file() → bool
Check if self is file.

load_numpy() → ndarray
Load NumPy array.

Returns

np.ndarray
loaded NumPy array

load_txt(**kwargs) → ndarray
Load NumPy array from text.

Returns

np.ndarray
loaded NumPy array

rglob(pattern: str) → List[DPath]
This is like calling DPath.glob() with /** added in front of the given relative pattern.

Parameters

pattern
[str] glob pattern

Returns

List[DPath]
list of paths

class deepmd_utils.utils.path.DPath(path: str)
Bases: ABC
The path class to data system (DeepmdData).

Parameters

path
[str] path

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>glob(pattern)</td>
<td>Search path using the glob pattern.</td>
</tr>
<tr>
<td>is_dir()</td>
<td>Check if self is directory.</td>
</tr>
<tr>
<td>is_file()</td>
<td>Check if self is file.</td>
</tr>
<tr>
<td>load_numpy()</td>
<td>Load NumPy array.</td>
</tr>
<tr>
<td>load_txt(**kwargs)</td>
<td>Load NumPy array from text.</td>
</tr>
<tr>
<td>rglob(pattern)</td>
<td>This is like calling DPath.glob() with /** added in front of the given relative pattern.</td>
</tr>
</tbody>
</table>
abstract glob(pattern: str) → List[DPPath]
Search path using the glob pattern.
Parameters
pattern
  [str] glob pattern
Returns
List[DPPath]
  list of paths

abstract is_dir() → bool
Check if self is directory.

abstract is_file() → bool
Check if self is file.

abstract load_numpy() → ndarray
Load NumPy array.
Returns
np.ndarray
  loaded NumPy array

abstract load_txt(**kwargs) → ndarray
Load NumPy array from text.
Returns
np.ndarray
  loaded NumPy array

abstract rglob(pattern: str) → List[DPPath]
This is like calling DPath.glob() with **/ added in front of the given relative pattern.
Parameters
pattern
  [str] glob pattern
Returns
List[DPPath]
  list of paths

deepmd_utils.utils.plugin module

Base of plugin systems.

class deepmd_utils.utils.plugin.Plugin
Bases: object

A class to register and restore plugins.
Examples

```python
>>> plugin = Plugin()
>>> @plugin.register("xx")
    def xxx():
        pass
>>> print(plugin.plugins['xx'])
```

Attributes

- `plugins`: `Dict[str, object]` plugins

Methods

- `get_plugin(key)`: Visit a plugin by key.
  - **Parameters**
    - `key`: `str` key of the plugin
  - **Returns**
    - `object` the plugin

- `register(key)`: Register a plugin.
  - **Parameters**
    - `key`: `str` key of the plugin
  - **Returns**
    - `Callable[[object], object]` decorator

```python
class deepmd_utils.utils.plugin.PluginVariant(*args, **kwargs)
    Bases: object
    A class to remove type from input arguments.

class deepmd_utils.utils.plugin.VariantABCMeta
    Bases: VariantMeta, ABCMeta
```

18.3. deepmd_utils package 599
Methods

```python
__call__(*args, **kwargs) Remove type and keys that starts with underline.
mro() Return a type's method resolution order.
register(subclass) Register a virtual subclass of an ABC.
```

class deepmd_utils.utils.plugin.VariantMeta
    Bases: object

Methods

```python
__call__(*args, **kwargs) Remove type and keys that starts with underline.
```

deepmd_utils.utils.random module

deepmd_utils.utils.random.choice(a: ndarray, p: Optional[ndarray] = None)
    Generates a random sample from a given 1-D array.
    Parameters
    a  [np.ndarray] A random sample is generated from its elements.
    p  [np.ndarray] The probabilities associated with each entry in a.
    Returns
    np.ndarray arrays with results and their shapes

deepmd_utils.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.utils.random.seed(val: Optional[int] = None)
    Seed the generator.
    Parameters
    val  [int] Seed.
```python
def deepmd_utils.utils.random.shuffle(x: ndarray):
    Modify a sequence in-place by shuffling its contents.
    Parameters
    x
        [np.ndarray] The array or list to be shuffled.
```

**deepmd_utils.utils.weight_avg module**

```python
def deepmd_utils.utils.weight_avg.weighted_average(errors: List[Dict[str, Tuple[float, float]]]) -> Dict
    Compute weighted 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
```

### 18.3.2 Submodules

### 18.3.3 deepmd_utils.common module

```python
def deepmd_utils.common.add_data_requirement(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: Optional[bool] = None, repeat: int = 1, default: float = 0.0, dtype: Optional[dtype] = None)
    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 keyword 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
```

18.3.  deepmd_utils package
repeat
[int, optional] if specify repeat 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

deepmd_utils.common.expand_sys_str(root_dir: Union[str, 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_utils.common.get_np_precision(precision: _PRECISION) → dtype
Get numpy precision constant from string.

Parameters
precision
[_PRECISION] string name of numpy constant or default

Returns
np.dtype
numpy precision constant

Raises
RuntimeError
if string is invalid

deepmd_utils.common.j_loader(filename: Union[str, 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_utils.common.j_must_have(jdata: Dict[str, _DICT_VAL], key: str, deprecated_key: List[str] = []) → _DICT_VAL
Assert that supplied dictionary contains specified key.

Returns
_DICT_VAL
value that was store unde supplied key

Raises

RuntimeError
if the key is not present

depmd_utils.common.make_default_mesh(pbc: bool, mixed_type: bool) \rightarrow 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

depmd_utils.common.select_idx_map(atom_types: ndarray, select_types: ndarray) \rightarrow ndarray
Build map of indices for element supplied element types from all atoms list.

Parameters

atom_types

[np.ndarray] array specifying 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

18.3.4 deepmd_utils.env module

depmd_utils.env.GLOBAL_ENER_FLOAT_PRECISION
alias of float64

depmd_utils.env.GLOBAL NP_FLOAT_PRECISION
alias of float64
18.3.5 deepmd_utils.main module

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.

class deepmd_utils.main.RawTextArgumentDefaultsHelpFormatter (prog, indent_increment=2, max_help_position=24, width=None)

Bases: RawTextHelpFormatter, ArgumentDefaultsHelpFormatter

This formatter is used to print multile-line help message with default value.

Methods

format_help()

start_section(heading)

depemd_utils.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

depemd_utils.main.main()

DeePMD-kit new entry point.

Raises
RuntimeError
if no command was input

depemd_utils.main.main_parser() → ArgumentParser

DeePMD-Kit commandline options argument parser.

Returns
argparse.ArgumentParser
main parser of DeePMD-kit
deepmd_utils.main.parse_args(args: Optional[List[str]] = None) → 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
19.1 op_module

Python wrappers around TensorFlow ops.

This file is MACHINE GENERATED! Do not edit.

```python
deepmd.env.op_module.AddFltNvmmd(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`.

```python
class deepmd.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.
- Nested `Annotated` are flattened:

  ```python
  Annotated[Annotated[T, Ann1, Ann2], Ann3] == Annotated[T, Ann1, Ann2, Ann3]
  ```

- Instantiating an annotated type is equivalent to instantiating the underlying type:

  ```python
  Annotated[C, Ann1](5) == C(5)
  ```

- `Annotated` can be used as a generic type alias:
DeePMD-kit

```python
Optimized = Annotated[T, runtime.Optimize()]
Optimized[int] == Annotated[int, runtime.Optimize()]

OptimizedList = Annotated[List[T], runtime.Optimize()]
OptimizedList[int] == Annotated[List[int], runtime.Optimize()]
```

deepmd.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.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.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.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.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.
• \texttt{davg} – A Tensor. Must have the same type as \texttt{coord}.
• \texttt{dstd} – A Tensor. Must have the same type as \texttt{coord}.
• \texttt{rcut\_a} – A float.
• \texttt{rcut\_r} – A float.
• \texttt{rcut\_r\_smth} – A float.
• \texttt{sel\_a} – A list of ints.
• \texttt{sel\_r} – A list of ints.
• \texttt{name} – A name for the operation (optional).

Returns

A tuple of Tensor objects (\texttt{descrpt}, \texttt{descrpt\_deriv}, \texttt{rij}, \texttt{nlist}).

\texttt{deepmd.env.op_module.DescrptSeAEf}(\texttt{coord}, \texttt{type}, \texttt{natoms}, \texttt{box}, \texttt{mesh}, \texttt{ef}, \texttt{davg}, \texttt{dstd}, \texttt{rcut\_a}, \texttt{rcut\_r},
\texttt{rcut\_r\_smth}, \texttt{sel\_a}, \texttt{sel\_r}, \texttt{name}=\texttt{None})

\textbf{TODO: add doc.}

Parameters

• \texttt{coord} – A Tensor. Must be one of the following types: float32, float64.
• \texttt{type} – A Tensor of type int32.
• \texttt{natoms} – A Tensor of type int32.
• \texttt{box} – A Tensor. Must have the same type as \texttt{coord}.
• \texttt{mesh} – A Tensor of type int32.
• \texttt{ef} – A Tensor. Must have the same type as \texttt{coord}.
• \texttt{davg} – A Tensor. Must have the same type as \texttt{coord}.
• \texttt{dstd} – A Tensor. Must have the same type as \texttt{coord}.
• \texttt{rcut\_a} – A float.
• \texttt{rcut\_r} – A float.
• \texttt{rcut\_r\_smth} – A float.
• \texttt{sel\_a} – A list of ints.
• \texttt{sel\_r} – A list of ints.
• \texttt{name} – A name for the operation (optional).

Returns

A tuple of Tensor objects (\texttt{descrpt}, \texttt{descrpt\_deriv}, \texttt{rij}, \texttt{nlist}).

depthmd.env.op_module.DescrptSeAEfPara(\texttt{coord}, \texttt{type}, \texttt{natoms}, \texttt{box}, \texttt{mesh}, \texttt{ef}, \texttt{davg}, \texttt{dstd}, \texttt{rcut\_a}, \texttt{rcut\_r},
\texttt{rcut\_r\_smth}, \texttt{sel\_a}, \texttt{sel\_r}, \texttt{name}=\texttt{None})

\textbf{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.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).

dscrpt: 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.

depmd.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).

dscrpt: 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.

depmd.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).

dscrpt: 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.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.

deeper.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.

deeper.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.

deeper.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.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.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.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.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.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`.
- `name` – A name for the operation (optional).

Returns
A Tensor. Has 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`.

depmd.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`.

depmd.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`.

depmd.env.op_module.MapFltNvmmd(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`.

depmd.env.op_module.MatmulFitnetNvmmd(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 $x$.
• $\text{nbit}_x$ – An int.
• $\text{nbit}_w$ – An int.
• $\text{norm}_w$ – An int.
• $\text{name}$ – A name for the operation (optional).

Returns
A Tensor. Has the same type as $x$.

deepmind.env.op_module.MatmulFlt2fixNvmmd($x$, $w$, $\text{nbit}$, $\text{name}=\text{None}$)
 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 $x$.
• $\text{nbit}$ – An int.
• $\text{name}$ – A name for the operation (optional).

Returns
A Tensor. Has the same type as $x$.

deepmind.env.op_module.MatmulFltNvmmd($x$, $w$, $\text{norm}_x$, $\text{norm}_w$, $\text{name}=\text{None}$)
 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 $x$.
• $\text{norm}_x$ – An int.
• $\text{norm}_w$ – An int.
• $\text{name}$ – A name for the operation (optional).

Returns
A Tensor. Has the same type as $x$.

deepmind.env.op_module.MulFltNvmmd($x$, $w$, $\text{name}=\text{None}$)
 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 $x$.
• $\text{name}$ – A name for the operation (optional).

Returns
A Tensor. Has the same type as $x$.

deepmind.env.op_module.NeighborStat($\text{coord}$, $\text{type}$, $\text{natoms}$, $\text{box}$, $\text{mesh}$, $\text{rcut}$, $\text{name}=\text{None}$)
 TODO: add doc.

Parameters
• $\text{coord}$ – 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.

```python
deepmd.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.

```python
deepmd.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`.

```python
deepmd.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 $R^i$ can be constructed as follows

$$
(R^i)_j = \begin{bmatrix}
\frac{s(r_{ji})}{s(r_{ji})r_{ji}} \\
\frac{s(r_{ji})y_{ji}}{s(r_{ji})r_{ji}} \\
\frac{s(r_{ji})z_{ji}}{s(r_{ji})r_{ji}}
\end{bmatrix}
$$

In the above equation, $R_{ji} = R_j - R_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \|R_{ji}\|$ is its norm. The switching function $s(r)$ is defined as:

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

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]`: 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. 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.
- **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 maximum 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).

desrcript: A Tensor. Has the same type as coord. The environment matrix.
desrpt_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.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).

desrcript: 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.
type: A Tensor of type int32. The corresponding atom types in nlist. nmask: A Tensor
of type bool. The atom mask in nlist.
DeePMD-kit

deepmd.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, 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.

depmd.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).

descript: 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.

```python
def deepmd.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).

descript: 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.

```python
def deepmd.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.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.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.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.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.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.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.
DeePMD-kit

- **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.

depmd.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.

depmd.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.

depmd.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.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.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.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.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.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.

def deepmd.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.

def deepmd.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.

def deepmd.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.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.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.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.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.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.

```python
deepmd.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.

```python
deepmd.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.

```python
deepmd.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.

```python
deepmd.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.

```python
deepmd.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.

```python
deepmd.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-kit

deepmd.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.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.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.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.env.op_module.add_flt_nvmmd**(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.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.env.op_module.copy_flt_nvmmd**(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 \( x \).


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.env.op_module.descrpt_norot**

```python
```

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.

depemd.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.

depemd.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.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.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.env.op_module.descript_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.


TODO: add doc.

Parameters
DeePMD-kit

- `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.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.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.env.op_module.ewald_recpcod: 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.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.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.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.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.

depm.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.

depm.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.

depm.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.

depm.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.
• \texttt{nlist} – A Tensor of type int32.
• \texttt{natoms} – A Tensor of type int32.
• \texttt{n_a Sel} – An int.
• \texttt{n_r Sel} – An int.
• \texttt{name} – A name for the operation (optional).

Returns
A Tensor. Has the same type as \texttt{aparam}.

\begin{verbatim}
depdeepmd.env.op_module.map_flt_nvnmd(x: Any, table: Any, table_grad: Any, table_info: Any, name=None) → Any

todo: add doc.
\end{verbatim}

Parameters
\begin{itemize}
  \item \texttt{x} – A Tensor. Must be one of the following types: float32, float64.
  \item \texttt{table} – A Tensor. Must have the same type as \texttt{x}.
  \item \texttt{table_grad} – A Tensor. Must have the same type as \texttt{x}.
  \item \texttt{table_info} – A Tensor. Must have the same type as \texttt{x}.
  \item \texttt{name} – A name for the operation (optional).
\end{itemize}

Returns
A Tensor. Has the same type as \texttt{x}.

\begin{verbatim}
depdeepmd.env.op_module.matmul_fitnet_nvnmd(x: Any, w: Any, nbitx: int, nbitw: int, normw: int, name=None) → Any

todo: add doc.
\end{verbatim}

Parameters
\begin{itemize}
  \item \texttt{x} – A Tensor. Must be one of the following types: float32, float64.
  \item \texttt{w} – A Tensor. Must have the same type as \texttt{x}.
  \item \texttt{nbitx} – An int.
  \item \texttt{nbitw} – An int.
  \item \texttt{normw} – An int.
  \item \texttt{name} – A name for the operation (optional).
\end{itemize}

Returns
A Tensor. Has the same type as \texttt{x}.

\begin{verbatim}
depdeepmd.env.op_module.matmul_flt2fix_nvnmd(x: Any, w: Any, nbit: int, name=None) → Any

todo: add doc.
\end{verbatim}

Parameters
\begin{itemize}
  \item \texttt{x} – A Tensor. Must be one of the following types: float32, float64.
  \item \texttt{w} – A Tensor. Must have the same type as \texttt{x}.
  \item \texttt{nbit} – An int.
  \item \texttt{name} – A name for the operation (optional).
\end{itemize}

Returns
A Tensor. Has the same type as \texttt{x}.
DeePMD-kit

\texttt{deepmd.env.op_module.matmul_flt_nvnmd}(x: \text{Any}, w: \text{Any}, normx: \text{int}, normw: \text{int}, name=None) \rightarrow \text{Any}

TODO: add doc.

Parameters

\begin{itemize}
  \item \textit{x} – A Tensor. Must be one of the following types: float32, float64.
  \item \textit{w} – A Tensor. Must have the same type as \textit{x}.
  \item \textit{normx} – An int.
  \item \textit{normw} – An int.
  \item \textit{name} – A name for the operation (optional).
\end{itemize}

Returns

A Tensor. Has the same type as \textit{x}.

\texttt{deepmd.env.op_module.mul_flt_nvnmd}(x: \text{Any}, w: \text{Any}, name=None) \rightarrow \text{Any}

TODO: add doc.

Parameters

\begin{itemize}
  \item \textit{x} – A Tensor. Must be one of the following types: float32, float64.
  \item \textit{w} – A Tensor. Must have the same type as \textit{x}.
  \item \textit{name} – A name for the operation (optional).
\end{itemize}

Returns

A Tensor. Has the same type as \textit{x}.

\texttt{deepmd.env.op_module.neighbor_stat}(coord: \text{Any}, type: \text{Any}, natoms: \text{Any}, box: \text{Any}, mesh: \text{Any}, rcut: \text{float}, name=None)

TODO: add doc.

Parameters

\begin{itemize}
  \item \textit{coord} – A Tensor. Must be one of the following types: float32, float64.
  \item \textit{type} – A Tensor of type int32.
  \item \textit{natoms} – A Tensor of type int32.
  \item \textit{box} – A Tensor. Must have the same type as \textit{coord}.
  \item \textit{mesh} – A Tensor of type int32.
  \item \textit{rcut} – A float.
  \item \textit{name} – A name for the operation (optional).
\end{itemize}

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 \textit{coord}.

\texttt{deepmd.env.op_module.pair_tab}(table_info: \text{Any}, table_data: \text{Any}, type: \text{Any}, rij: \text{Any}, nlist: \text{Any}, natoms: \text{Any}, scale: \text{Any}, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

\begin{itemize}
  \item \textit{table_info} – A Tensor of type float64.
\end{itemize}
• **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.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.

```
```

Compute the environment matrix for descriptor se_e2_a.

Each row of the environment matrix $\mathcal{R}^i$ can be constructed as follows
\[
(R^i)_j = \frac{s(r_{ji})}{s(r_{ji})s(r_{ji})s(r_{ji})s(r_{ji})}
\]

In the above equation, \( R_{ji} = R_j - R_i = (x_{ji}, y_{ji}, z_{ji}) \) is the relative coordinate and \( r_{ji} = \| R_{ji} \| \) is its norm. The switching function \( s(r) \) is defined as:

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

Note that the environment matrix is normalized by \( \text{davg} \) and \( \text{dstd} \).

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 \( N_{\text{types}} + 2 \). \( \text{natoms}[0] \): number of local atoms. \( \text{natoms}[1] \): total number of atoms held by this processor. \( \text{natoms}[i] \): \( 2 \leq i < N_{\text{types}} + 2 \), 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. For historical reasons, only the length of the Tensor matters. If size of mesh == 6, \( \text{pbc} \) is assumed. If size of mesh == 0, no-\( \text{pbc} \) 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 maximum 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 (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. The environment matrix. descript_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.
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 (descript, descript_deriv, rij, nlist, ntype, nmask).

• **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.


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-kit

To be added.

**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.

To be added.

DeepMD-kit

To be added.

**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-kit

To be added.

**Parameters**

DeepMD-kit

To be added.
• `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`.

```python
deepmd.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`.

```python
deepmd.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`.

```python
deepmd.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.env.op_module.prod_virial`

```python
```

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.env.op_module.prod_virial_norot`

```python
deepmd.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.

```python
deepmd.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.

```python
deepmd.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.

```python
deepmd.env.op_module.quantize_nvmd(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.

depm.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.

depm.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.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.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.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.


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.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.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).

```python
depmod.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.

```python
depmod.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.

```python
deepmd.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.

```python
```

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.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.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.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.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.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.

depmd.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.

depdeepmd.env.op_module.tanh4_flt_nvmd(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.

depdeepmd.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.

depdeepmd.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.

depdeepmd.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.

depm.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.

19.2 op_grads_module

Python wrappers around TensorFlow ops.

This file is MACHINE GENERATED! Do not edit.

class depm.env.op_grads_module.Annotated(*args, **kwargs)
    Bases: object

Add context specific metadata to a type.

Example: Annotated[int, runtime_checkUnsigned] indicates to the hypothetical runtime_check mod-
ule 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.
- Nested Annotated are flattened:

\[
\text{Annotated[Annotated[T, Ann1, Ann2], Ann3]} == \text{Annotated[T, Ann1, Ann2, Ann3]}
\]

- Instantiating an annotated type is equivalent to instantiating the
underlying type:
Annotated[C, Ann1](5) == C(5)

- Annotated can be used as a generic type alias:

```python
Optimized = Annotated[T, runtime.Optimize()]
Optimized[int] == Annotated[int, runtime.Optimize()]

OptimizedList = Annotated[List[T], runtime.Optimize()]
OptimizedList[int] == Annotated[List[int], runtime.Optimize()]
```

deepmd.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.
- **natsoms** – 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.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.
- **natsoms** – 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.env.op_grads_module.ProdForceSeAMaskGrad(\(\text{grad}, \text{net\_deriv}, \text{in\_deriv}, \text{mask}, \text{nlist}, \text{total\_atom\_num}, \text{name}=\text{None}\))

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.env.op_grads_module.ProdForceSeRGrad(\(\text{grad}, \text{net\_deriv}, \text{in\_deriv}, \text{nlist}, \text{natoms}, \text{name}=\text{None}\))

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.env.op_grads_module.ProdVirialGrad(\(\text{grad}, \text{net\_deriv}, \text{in\_deriv}, \text{rij}, \text{nlist}, \text{axis}, \text{natoms}, \text{n\_a\_sel}, \text{n\_r\_sel}, \text{name}=\text{None}\))

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.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.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.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.

```python
deepmd.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.

```python
```

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.
The `prod_force_se_a_grad` function takes several 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).

The return type is `Any`.

The `prod_force_se_a_mask_grad` function has similar parameters and returns a Tensor of the same type as `grad`.

The `prod_force_se_r_grad` function also has similar parameters and returns a Tensor of the same type as `grad`.

The `add_doc` function is a placeholder for adding documentation.
Returns
A Tensor. Has the same type as grad.


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.


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.

depmd.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.


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.


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-kit

Chapter 19. OP API
20.1 Class Hierarchy

- Namespace deepmd
  - Struct deepmd_exception
  - Struct NeighborListData
  - Struct tf_exception
  - Class AtomMap
  - Class DeepPot
  - Class DeepPotBase
  - Class DeepPotModelDevi
  - Class DeepPotTF
  - Class DeepTensor
  - Class DeepTensorBase
  - Class DeepTensorTF
  - Class DipoleChargeModifier
  - Class DipoleChargeModifierBase
  - Class DipoleChargeModifierTF
  - Enum DPBackend

20.2 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_DataModifier.h
20.3 Full API

20.3.1 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 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_compatible
- 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(std::vector<VT>&, const std::vector<VT>&, const std::vector<int>&, const int&, const int&, const int&, const int&)
- 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_inv(std::vector<VT>&, const std::vector<VT>&, const std::vector<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_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<VATYPE>&, const int&, const std::vector<int>&, const std::vector<VATYPE>&, const double&, const std::vector<VATYPE>&, const std::vector<VATYPE>&, const deepmd::AtomMap&, const std::string, const bool)
- Template Function deepmd::session_input_tensors(std::vector<std::pair<std::string, tensorflow::Tensor>>&, const std::vector<VATYPE>&, const int&, const std::vector<int>&, const std::vector<VATYPE>&, const InputNlist&, const std::vector<VATYPE>&, const deepmd::AtomMap&, const int, const int, const std::string, const bool)
- Template Function deepmd::session_input_tensors_mixed_type
Typedefs

- Typedef deepmd::ENERGYTYPE
- Typedef deepmd::STRINGTYPE

Namespace tensorflow

20.3.2 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)
**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 VALUE_TYPE>
void forward(typename std::vector<VALUE_TYPE>::iterator out, const typename std::vector<VALUE_TYPE>::const_iterator in, const int stride = 1, const int nframes = 1, const int null = 0) const

template<typename VALUE_TYPE>
void backward(typename std::vector<VALUE_TYPE>::iterator out, const typename std::vector<VALUE_TYPE>::const_iterator in, const int stride = 1, const int nframes = 1, const int null = 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 VALUE_TYPE>
void compute(ENERGY_TYPE &ener, std::vector<VALUE_TYPE> &force,
std::vector<VALUE_TYPE> &virial, const std::vector<VALUE_TYPE> &coord, const std::vector<int> &atype, const std::vector<VALUE_TYPE> &box, const std::vector<VALUE_TYPE> &fparam = std::vector<VALUE_TYPE>(), const std::vector<VALUE_TYPE> &aparam = std::vector<VALUE_TYPE>())

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.

```cpp
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**

```cpp
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(VALUETYPE &ener, std::vector<VALUETYPE> &force,
             std::vector<ENERGYTYPE> &virial, const std::vector<ENERGYTYPE> &coord,
             const std::vector<int> &atype, const std::vector<ENERGYTYPE> &box, const int nghost,
             const InputNlist &inlist, const int &ago, const std::vector<ENERGYTYPE> &fparam = std::vector<ENERGYTYPE>(),
             const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>(0))
```

### Unnamed Group

```
template<typename VALUETYPE>
void compute(ENERGYTYPE &ener, std::vector<VALUETYPE> &force,
             std::vector<ENERGYTYPE> &virial, const std::vector<ENERGYTYPE> &atom_energy,
             std::vector<ENERGYTYPE> &atom_virial, const std::vector<ENERGYTYPE> &coord,
             const std::vector<int> &atype, const std::vector<ENERGYTYPE> &box, const std::vector<ENERGYTYPE> &fparam = std::vector<ENERGYTYPE>(),
             const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>(0))
```

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.
- `atom_energy` – [out] The atomic energy.
- `coord` – [in] The coordinates of atoms. The array should be of size `nframes x natoms x 3`.
- `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(ENERGYTYPE &ener, std::vector<VALUETYPE> &force,
             std::vector<ENERGYTYPE> &virial, const std::vector<ENERGYTYPE> &atom_energy,
             std::vector<ENERGYTYPE> &atom_virial, const std::vector<ENERGYTYPE> &coord,
             const std::vector<int> &atype, const std::vector<ENERGYTYPE> &box, const std::vector<ENERGYTYPE> &fparam = std::vector<ENERGYTYPE>(),
             const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>(0))
```
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>(0))

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>(0))
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>())

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.

```cpp
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 string &model, const int &gpu_rank = 0, const 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 string &model, const int &gpu_rank = 0, const string &file_content = "")**

Initialize the DP.

Parameters
DeePMD-kit

- **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.

```cpp
void print_summary(const std::string &pre) const
    Print the DP summary to the screen.
Parameters
    pre – [in] The prefix to each line.
```

```cpp
double cutoff() const
    Get the cutoff radius.
Returns
    The cutoff radius.
```

```cpp
int numb_types() const
    Get the number of types.
Returns
    The number of types.
```

```cpp
int numb_types_spin() const
    Get the number of types with spin.
Returns
    The number of types with spin.
```

```cpp
int dim_fparam() const
    Get the dimension of the frame parameter.
Returns
    The dimension of the frame parameter.
```

```cpp
int dim_aparam() const
    Get the dimension of the atomic parameter.
Returns
    The dimension of the atomic parameter.
```

```cpp
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.
```

```cpp
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 Type**

- `public deepmd::DeepPotTF (Class DeepPotTF)`

**Class Documentation**

class **DeepPotBase**

Deep Potential.

Subclassed by `deepmd::DeepPotTF`

**Unnamed Group**

virtual void **compute**

```cpp
class DeepPotBase

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,
std::vector<int> &atype, const std::vector<double> &box,
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 compute(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 compute(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 compute(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<int>& atype, const std::vector<float>& coord, 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 compute_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<float>& coord, const std::vector<double>& box, const std::vector<float>& fparam = std::vector<float>(), const std::vector<float>& aparam = std::vector<float>()) = 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.
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
• 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
• 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 contructor.

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 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.
- box – [in] The cell of the region. The array should be of size nframes x 9.
- 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.

```cpp
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.
```

```cpp
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.
```
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.

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

- 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 VALUEPE eps, 
    const int &stride)
    
Compute the relative standard deviation of vectors.

Parameters

- 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


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.

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 **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
- `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 \texttt{init} (\texttt{\textbf{const std::string \\&model, const int \\&gpu_rank = 0, const std::string \\&file_content = "")}

Initialize the DP.

Parameters

\begin{itemize}
\item \textbf{model} – [in] The name of the frozen model file.
\item \textbf{gpu_rank} – [in] The GPU rank. Default is 0.
\item \textbf{file_content} – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.
\end{itemize}

inline virtual double \texttt{cutoff()} const

Get the cutoff radius.

Returns

The cutoff radius.

inline virtual int \texttt{numb_types()} const

Get the number of types.

Returns

The number of types.

inline virtual int \texttt{numb_types_spin()} const

Get the number of types with spin.

Returns

The number of types with spin.

inline virtual int \texttt{dim_fparam()} const

Get the dimension of the frame parameter.

Returns

The dimension of the frame parameter.

inline virtual int \texttt{dim_aparam()} const

Get the dimension of the atomic parameter.

Returns

The dimension of the atomic parameter.

virtual void \texttt{get_type_map(\texttt{std::string \\&type_map})}

Get the type map (element name of the atom types) of this model.

Parameters

\begin{itemize}
\item \textbf{type_map} – [out] The type map of this model.
\end{itemize}

inline virtual bool \texttt{is_aparam_nall()} const

Get whether the atom dimension of aparam is nall instead of fparam.

Parameters

\begin{itemize}
\item \textbf{aparam_nall} – [out] whether the atom dimension of aparam is nall instead of fparam.
\end{itemize}

virtual void \texttt{compute(\texttt{std::vector<double \\&ener, std::vector<double \\&force, std::vector<double \\&virial, std::vector<double \\&atominviral, const std::vector<double \\&coord, const std::vector<int \\&atype, const std::vector<double \\&box, const std::vector<double \\&fp\texttt{param = std::vector<double \\>()}, const std::vector<double \\&aparam = std::vector<double \\>()})}}

Chapter 20. \textit{C++ API}
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.

```cpp
virtual void computeD(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>())
```

```cpp
virtual void computeD(std::vector<double>& ener, std::vector<double>& force, std::vector<double>& virial, std::vector<double>& atom_energy, std::vector<double>& atom_virial, std::vector<int>& intlist, const int intago, 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>())

define 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. 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. Then all frames are assumed to be provided with the same aparam.

```cpp
virtual void compute_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 evaluate, 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 evaluate, 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 evaluate.
• 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.
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 evaluate.
- **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 evaluate.
- **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 evaluate.
• **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()` { 
  Get the cutoff radius.
  
  Returns
  The cutoff radius.
}

int `numb_types()` { 
  Get the number of types.
  
  Returns
  The number of types.
}

int `output_dim()` { 
  Get the output dimension.
  
  Returns
  The output dimension.
}

const std::vector<int>& `sel_types()` { 
  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 compute(float& global_tensor, float& force,
                    float& virial, float& atom_tensor,
                    float& atom_virial, const float& coord,
                    const float& atype, const float& box, const bool request_deriv) = 0

Evaluate the global tensor and component-wise force and virial.

Parameters

- global_tensor – [out] The global tensor to evaluate.
- force – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- atom_tensor – [out] The atomic tensor value of the model, size natoms x odim.
- coord – [in] The coordinates of atoms. The array should be of size natoms x 3.
- request_deriv – [in] Whether to request the derivative of the global tensor, including force and virial.
virtual void compute(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 evaluate.
- **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 compute(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

• 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 evaluate.
- **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 evaluate.
• **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 compute<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**

DeePMDF::DeepTensorTF()  
Deep Tensor constructor without initialization.

~DeePMDF::DeepTensorTF()  
Deep Tensor constructor with initialization.

```
DeePMDF::DeepTensorTF(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")
```

Initialize the Deep Tensor.

```
virtual void init(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")
```

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
  • gpu_rank – [in] The GPU rank. Default is 0.
  • name_scope – [in] The name scope.
DeePMD-kit

-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> &datatype_, 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 natoms x 3.
- **datatype_** – [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.

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.

std::vector<int> **sel_types**(const)
Get the list of sel types.
DeePMD-kit

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 compute($\text{std::vector<double> \&dfcorr_}$, $\text{std::vector<double> \&dvcorr_}$, $\text{const std::vector<double> \&dcoord_}$, $\text{const std::vector<int> \&datatype_}$, $\text{const std::vector<double> \&dbox}$, $\text{const std::vector<std::pair<int,int>> \&pairs}$, $\text{const std::vector<double> \&delef_}$, $\text{const int nghost}$, $\text{const InputNlist \&lmp_list}$) = 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.
- **datatype_** - [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 compute($\text{std::vector<float> \&dfcorr_}$, $\text{std::vector<float> \&dvcorr_}$, $\text{const std::vector<float> \&dcoord_}$, $\text{const std::vector<int> \&datatype_}$, $\text{const std::vector<float> \&dbox}$, $\text{const std::vector<std::pair<int,int>> \&pairs}$, $\text{const std::vector<float> \&delef_}$, $\text{const int nghost}$, $\text{const InputNlist \&lmp_list}$) = 0
Public Functions

```cpp
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
• 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
• 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 std::vector<int> sel_types() const = 0
Get the list of sel types.
Returns
The list of sel types.
```
Class \texttt{DipoleChargeModifierTF}

- Defined in file \texttt{source_api_cc/include/DataModifierTF.h}

Inheritance Relationships

Base Type

- \texttt{public deepmd::DipoleChargeModifierBase (Class DipoleChargeModifierBase)}

Class Documentation

class \texttt{DipoleChargeModifierTF : public deepmd::DipoleChargeModifierBase}

Dipole charge modifier.

Public Functions

\texttt{DipoleChargeModifierTF()}

Dipole charge modifier without initialization.

\texttt{DipoleChargeModifierTF(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")}

Dipole charge modifier without initialization.

Parameters

- \texttt{model} – [in] The name of the frozen model file.
- \texttt{gpu_rank} – [in] The GPU rank. Default is 0.
- \texttt{name_scope} – [in] The name scope.

\texttt{-DipoleChargeModifierTF()}

virtual void \texttt{init(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")}

Initialize the dipole charge modifier.

Parameters

- \texttt{model} – [in] The name of the frozen model file.
- \texttt{gpu_rank} – [in] The GPU rank. Default is 0.
- \texttt{name_scope} – [in] The name scope.

\texttt{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 std::vector<int> sel_types() const
Get the list of sel types.
    Returns
    The list of sel types.
virtual void compute(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<std::pair<int, int>>&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 compute(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<std::pair<int, int>>&delef_, const int nghost, const InputNlist&lmp_list)

20.3.3 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*

### 20.3.4 Functions

**Function deepmd::check_status**

- Defined in file `source_api.cc/include_common.h`

**Function Documentation**

```cpp
def check_status(const tensorflow::Status& status)
    Check TensorFlow status. Exit if not OK.
    Parameters
```

**Function deepmd::convert_pbtxt_to_pb**

- Defined in file `source_api.cc/include_common.h`

**Function Documentation**

```cpp
def convert_pbtxt_to_pb(std::string fn_pb_txt, std::string fn_pb)
    Convert .pbtxt to .pb.
    Parameters
```
Function deepmd::get_env_nthreads

- Defined in file_source_api_cc_include_common.h

**Function Documentation**

```cpp
def 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 TF_INTRA_OP_PARALLELISM_THREADS.
- `num_inter_nthreads` - [out] The number of inter threads. Read from TF_INTER_OP_PARALLELISM_THREADS.

Function deepmd::load_op_library

- Defined in file_source_api_cc_include_common.h

**Function Documentation**

```cpp
def load_op_library()
```
Dynamically load OP library. This should be called before loading graphs.

Function deepmd::model_compatible

- Defined in file_source_api_cc_include_common.h

**Function Documentation**

```cpp
def model_compatible(std::string& model_version)
```
Check if the model version is supported.

**Parameters**

**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.

**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> &datatype_, const int &nghost, const std::vector<int> &sel_type_)`  

Get forward and backward map of selected atoms by atom types.

Parameters
DeePMD-kit

- **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.
- **nghost** – [in] The number of ghost atoms.

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.
- **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&, 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.
• 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_common.h

Function Documentation

int deepmd::session_get_dtype(tensordflow::Session *session, const std::string name, const std::string scope = "")

Get the type of a tensor.

Parameters

- name – [in] The name of the tensor.

Returns

The type of the tensor as int.
**Template Function deepmd::session_get_scalar**

- Defined in file _source_api_cc_include_common.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
- `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_common.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.
- `name` - [in] The name of the tensor.
- `scope` - [in] The scope of the tensor.

**Template Function deepmd::session_input_tensors**

- Defined in file _source_api_cc_include_common.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> &datatype_, 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.
- **ntypes** – [in] Number of atom types.
- **dbox** – [in] Box matrix.
- **cell_size** – [in] Cell size.
- **scope** – [in] The scope of the tensors.
- **aparam_nall** – [in] Whether the atomic dimension of atomic parameters is nall.

Template Function

```
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>& datatype_, 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)
```

- Defined in file source_api_cc_include_common.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> &datatype_, const std::vector<VALUETYPE> &dbox, const double &cell_size, 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.
- **ntypes** – [in] Number of atom types.
- **dlist** – [in] Neighbor list.
- **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_common.h`

Function Documentation

```cpp
template<typename MODELTYPE, typename VALUETYPE>
int deepmd::session_input_tensors_mixed_type
(const std::vector<std::pair<std::string, tensorflow::Tensor>>& input_tensors,
 const int nframes,
 const std::vector<VALUETYPE>& dcoord_,
 const int ntypes,
 const std::vector<VALUETYPE>& datatype_,
 const std::vector<VALUETYPE>& dbox, const double& cell_size,
 const std::vector<VALUETYPE>& fparam_,
 const std::vector<VALUETYPE>& aparam_,
 const deepmd::AtomMap& atommap,
 const string scope = "",
 const bool aparam_nall = false)
```

Get input tensors for mixed type.

Parameters

- **input_tensors** – [out] Input tensors.
- **nframes** – [in] Number of frames.
- **ntypes** – [in] Number of atom types.
- **dlist** – [in] Neighbor list.
• nghost – [in] Number of ghost atoms.
• ago – [in] Update the internal neighbour list if ago is 0.
• aparam_nall – [in] Whether the atomic dimension of atomic parameters is nall.

### 20.3.5 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
21.1 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

21.2 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
21.3 Full API

21.3.1 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
Namespace std

21.3.2 Classes and Structs

Struct deepmd_exception

- Defined in file_source_api_c_include_deepmd.hpp

Inheritance Relationships

Base Type

- public std::runtime_error

Struct Documentation

struct deepmd_exception : public std::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_)
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

depdeepmd::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

depdeepmd::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


inline void init(const std::string &model, const int &gpu_rank = 0, const std::string &file_content ="
    Initialize the DP.

Parameters


template<typename VALUETYPE, typename ENERGYVTYPE>
inline void compute\(\text{ENERGYVTYPE} &\text{ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &\text{virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &\text{atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &\text{fparam = std::vector<VALUETYPE>()}, const std::vector<VALUETYPE> &\text{aparam = std::vector<VALUETYPE>()}}}\)

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.

\[\text{template<typename VALUETYPE, typename ENERGYVTYPE>}
\[
\text{inline void compute(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())}\]

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.

```cpp
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.
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: Natomsshould 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 \times 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 \times 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 \times 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 \times 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 \( \times \) natoms \( \times 3 \).
• **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 \( \times 9 \) (PBC) or empty (no PBC).
• **fparam** – [in] The frame parameter. The array can be of size \( nframes \times \text{dim}_fparam \). 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 \( nframes \times \text{natoms} \times \text{dim}_aparam \). natoms \( \times \text{dim}_aparam \). Then all frames are assumed to be provided with the same aparam.

```cpp
template<typename VALUETYPE, typename ENERGYVTYPE>
inline void compute_mixed_type(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force,
                                std::vector<ENERGYVTYPE> &eneral, std::vector<VALUETYPE> &atom_energy,
                                std::vector<ENERGYVTYPE> &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 \( \times \) natoms \( \times 3 \).
• **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 \( \times 9 \) (PBC) or empty (no PBC).
• **fparam** – [in] The frame parameter. The array can be of size \( nframes \times \text{dim}_fparam \). 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

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.

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 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).

template<

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).

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**

```cpp
class DeepTensor

Deep Tensor.

**Public Functions**

```cpp
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.

```cpp
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.

```cpp
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).

```cpp
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.
- 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)

Evaluate the global tensor, force and virial by using this Deep Tensor.

Parameters

- force – [out] The force on each atom.
- coord – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- 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

- force – [out] The force on each atom.
- atom_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> &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 num_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

DeePMD-kit

• **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> &datatype_,
                    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 natoms x 3.

• **datatype_** – [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.

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.
21.3.3 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_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**

```cpp
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**

```cpp
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**

```cpp
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 *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, double *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
DeePMD-kit

**Function Documentation**

```c
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

```c
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 central 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

```c
inline void deepmd::hpp::convert_pbtxt_to_pb(std::string fn_pb_txt, std::string fn_pb)
```

Convert pbtxt to pb.

**Parameters**


**Function deepmd::hpp::read_file_to_string**

- Defined in file_source_api_c_include_deepmd.hpp

```c
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>& datatype_, 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.
- **nghost** – [in] The number of ghost atoms.

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.
- **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**

```c
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**

```c
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**

```c
void DP_DeepPotCompute(DP_DeepPot *dp, 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.
• **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**

```c
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.
- **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_DeepPotCompute**

*Defined in file source_api_c/include/c_api.h*

**Function Documentation**

```c
void DP_DeepPotCompute(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.
- **natom** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **cell** – [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

```c
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

- `nframes` – [in] The number of frames.
- `coord` – [in] The coordinates of atoms. The array should be of size natoms x 3.
- `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.
- `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.
- **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.
• 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**

```c
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.
- **natom** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
• 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.
• 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.
• \textit{fparam} – [in] The frame parameters. The array can be of size nframes x dim\_fp.
• \textit{aparam} – [in] The atom parameters. The array can be of size nframes x dim\_ap.
• \textit{energy} – [out] Output energy.
• \textit{force} – [out] Output force. The array should be of size natoms x 3.
• \textit{virial} – [out] Output virial. The array should be of size 9.
• \textit{atomic\_energy} – [out] Output atomic energy. The array should be of size natoms.
• \textit{atomic\_virial} – [out] Output atomic virial. The array should be of size natoms x 9.

Function \texttt{DP\_DeepPotGetCutoff}

• Defined in file\_source\_api\_c\_include\_c\_api.h

Function Documentation

\begin{verbatim}
double DP_DeepPotGetCutoff(DP_DeepPot *dp)
    Get the type map of a DP.
    Parameters
    Returns
        The cutoff radius.
\end{verbatim}

Function \texttt{DP\_DeepPotGetDimAParam}

• Defined in file\_source\_api\_c\_include\_c\_api.h

Function Documentation

\begin{verbatim}
int DP_DeepPotGetDimAParam(DP_DeepPot *dp)
    Get the dimension of atomic parameters of a DP.
    Parameters
    Returns
        The dimension of atomic parameters of the DP.
\end{verbatim}

Function \texttt{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
    Returns
        The dimension of frame parameters of the DP.

Function Documentation

int DP_DeepPotGetNumbTypes(DP_DeepPot *dp)
    Get the number of types of a DP.
    Parameters
    Returns
        The number of types of the DP.

Function Documentation

int DP_DeepPotGetNumbTypesSpin(DP_DeepPot *dp)
    Get the number of types with spin of a DP.
    Parameters
    Returns
        The number of types with spin of the DP.

Function Documentation

- Defined in file_source_api_c_include_c_api.h
**Function Documentation**

```c
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**

```c
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
    false the atomic dimension of atomic parameters is nloc
```

**Function DP_DeepPotModelDeviCheckOK**

- Defined in file_source_api_c_include_c_api.h

**Function Documentation**

```c
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_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

- coord – [in] The coordinates of atoms. The array should be of size natoms x 3.
- box – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- ago – [in] Update the internal neighbour list if ago is 0.
- 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.
• 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

```c
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.
- **natom** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natom x 3.
- **atype** – [in] The atom types. The array should contain natom 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 natom 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 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

- nframes – [in] The number of frames. Only support 1 for now.
- coord – [in] The coordinates of atoms. The array should be of size natoms x 3.
- box – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- 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.
- 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**

```c
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**

```c
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**

```c
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

```c
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

```c
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

```c
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
- false the atomic dimension of atomic parameters is nloc
Function **DP_DeepTensorCheckOK**

- Defined in file `source_api_c/include/c_api.h`

Function Documentation

```c
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

```c
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
- **natom** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natom x 3.
- **atype** – [in] The atom types. The array should contain natom ints.
- **cell** – [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 natom 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 natom.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natom 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

```c
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

- **natom** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size `natom x 3`.
- **atype** – [in] The atom types. The array should contain `natom` 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 `natom 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 `natom`.
- **atomic_virial** – [out] Output atomic virial. The array should be of size `natom 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

```c
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

- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **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

```c
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

- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **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**

```c
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**

- **natom** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **cell** – [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**

```c
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**

- **natom** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **cell** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
DeePMD-kit

- **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**

```c
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**

- **natom** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **cell** – [in] The box 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**

```c
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**

- **natom** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
• **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**

declares DP_DeepTensorGetCutoff(DP_DeepTensor *dt)

Get the type map of a Deep Tensor.

Parameters

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

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

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

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

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
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_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
DeePMD-kit

**Function Documentation**

```c
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.
- **natom** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **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 Documentation**

```c
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.
- **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_DipoleChargeModifierGetNumSelTypes**

- 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

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

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

Returns
   The sel types

Function DP_NewDeepPot

- Defined in file_source_api_c_include_c_api.h
Function Documentation

**DP_DeepPot** *(const char *c_model)*

DP constructor with initialization.

Parameters

Returns
- A pointer to the deep potential.

**Function DP_NewDeepPotModelDevi**

- Defined in `file_source_api_c_include_c_api.h`

Function Documentation

**DP_DeepPotModelDevi** *(const char **c_models, int n_models)*

DP model deviation constructor with initialization.

Parameters
- `n_models` – [in] The number of models.

**Function DP_NewDeepPotModelDeviWithParam**

- Defined in `file_source_api_c_include_c_api.h`

Function Documentation

**DP_DeepPotModelDevi** *(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
- `n_models` – [in] The number of models.
- `gpu_rank` – [in] The rank of the GPU.

Returns
- DP_DeepPotModelDevi* A pointer to the deep potential model deviation.
DeePMD-kit

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.

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

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

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**

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_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
    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
- 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
- nsel_type – [in] The number of 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**

```c
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**

- **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.

### 21.3.4 Defines

**Define DP_CHECK_OK**

- Defined in file_source_api_c_include_deepmd.hpp

**Define Documentation**

```c
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**

```c
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)

21.3.5 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.
22.1 Class Hierarchy

- Namespace deepmd
  - Struct deepmd_exception
  - Struct deepmd_exception_oom
  - Template Struct EwaldParameters
  - Struct InputNlist
  - Template Struct Region
- Template Struct DescriptSeRGPUEXecuteFunctor
- 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
22.2 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
22.3 Full API

22.3.1 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 *&, const std::vector<FPTYPE>&&)
- Template Function deepmd::malloc_device_memory(FPTYPE *&, const int)
- Template Function deepmd::malloc_device_memory(FPTYPE *&, std::vector<FPTYPE>&&)

Chapter 22. Core API
• Template Function deepmd::malloc_device_memory_sync(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 *&, 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 std::vector<FPTYPE>&)
• Template Function deepmd::memcpy_host_to_device(FPTYPE *, const FPTYPE *, const int)
• 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_nvmd_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)
• 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_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::ElectrostaticConversion`

Namespace std

### 22.3.2 Classes and Structs

**Struct `deepmd_exception`**

- Defined in `file_source_lib_include_errors.h`

**Inheritance Relationships**

**Base Type**

- `public std::runtime_error`

**Derived Type**

- `public deepmd::deepmd_exception_oom` ([Struct `deepmd_exception_oom`])

**Struct Documentation**

```cpp
struct deepmd_exception : public std::runtime_error

   General DeePMD-kit exception. Throw if anything doesn't work.

   Subclassed by `deepmd::deepmd_exception_oom`
```

**Public Functions**

```cpp
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

```cpp
struct deepmd_exception_oom : public deepmd::deepmd_exception
```

#### Public Functions

```cpp
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

```cpp
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 ()

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.

Template Struct Region

• Defined in file_source_lib_include_region.h

Struct Documentation

template< typename FPTYPE >
struct Region
DeePMD-kit

Public Functions

Region()
Region(FPTYPE *extern_boxt, FPTYPE *extern_rec_boxt)
~Region()

Public Members

FPTYPE *boxt
FPTYPE *rec_boxt

Template Struct DesrptSeRGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename FPTYPE>
struct DesrptSeRGPUExecuteFunctor

Public Functions

void operator() (const FPTYPE *coord, const int *type, const int *ilist, const int *jrage, 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 *ndescript, 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
DeePMD-kit

Public Functions

```cpp
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

```cpp
template<typename FPTYPE>
struct ProdForceSeRGPUExecuteFunctor
```

Public Functions

```cpp
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

```cpp
template<typename FPTYPE>
struct ProdVirialSeAGPUExecuteFunctor
```

Public Functions

```cpp
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(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, const VALUETYPE &dx, const VALUETYPE &dy, const 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 &shift_x, VALUETYPE &shift_y, VALUETYPE &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 &shift_x, VALUETYPE &shift_y, VALUETYPE &shift_z, int &shift_x, int &shift_y, int &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
22.3.3 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

22.3.4 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` defined in `file_source_lib_include_neighbor_list.h`

```c
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 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 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` defined in `file_source_lib_include_ComputeDescriptor.h`

```c
void 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 std::vector<int>&, const int, const int, const int, const int)
```

Function Documentation

void `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)
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 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, 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 std::vector<double>& rfield, 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 double&rmin, const double&rmax)

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, 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 std::vector<double>& rfield, 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 double&rmin, const double&rmax)
Function compute_descriptor_se_a_ef_vert

- Defined in file_source_lib/include_ComputeDescriptor.h

Function Documentation

```cpp
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<int>& 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

```cpp
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<int>& 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:

```cpp
- 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

---

Function compute_dRdT_2

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

---

Function copy_coord

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

```cpp
void copy_coord(const std::vector<double>& out_c, const std::vector<int>& out_t, const std::vector<int>& mapping, const std::vector<int>& ncell, const std::vector<int>& ngcell, const std::vector<double>& in_c, const std::vector<double>& in_t, const SimulationRegion& 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

```cpp
void deepmd::convert_nlist_gpu_device(InputNlist &gpu_nlist, InputNlist &cpu_nlist, int *&gpu_memory, const int &max_nbor_size)
```

Convert the 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

```cpp
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

```cpp
template<typename FPTYPE>
void deepmd::convert_to_inter_gpu(FPTYPE *ri, const Region<FPTYPE> &region, const FPTYPE *rp)
```
DeePMD-kit

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

```cpp
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

```cpp
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

```cpp
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

```cpp
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`
DeePMD-kit

**Function Documentation**

```cpp
template<typename FTYPE>
void deepmd::delete_device_memory(FTYPE *device)
```

**Template Function deepmd::dot1**

- Defined in file_source_lib_include_utilities.h

**Function Documentation**

```cpp
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**

```cpp
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**

```cpp
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**

```cpp
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

\[
\text{template<typename } \text{TYPE}\text{>}
\]
\[
\text{inline void deepmd::dotmv3(TYPE *vec_o, const TYPE *tensor, const TYPE *vec_i)}
\]

Function deepmd::DPGetDeviceCount

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

\[
\text{inline void deepmd::DPGetDeviceCount(int &gpu_num)}
\]

Function deepmd::dprc_pairwise_map_cpu

- Defined in file_source_lib_include_pairwise.h

Function Documentation

\[
\text{void deepmd::dprc_pairwise_map_cpu(std::vector<int> &forward_qm_map, std::vector<int> &backward_qm_map, std::vector<int> &forward_qmmm_map, std::vector<int> &backward_qmmm_map, int &nloc_qm, int &nloc_qmmm, int &nall_qm, int &nall_qmmm, const std::vector<std::vector<int>>& fragments, const int nloc, const int nall)}
\]

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**

```cpp
cudaError_t deepmd::DPSetDevice(int rank)
```

**Template Function deepmd::env_mat_a_cpu**

- Defined in file_source_lib_include_env_mat.h

**Function Documentation**

```cpp
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**

```cpp
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**

```cpp
template<typename FPTYPE>
void deepmd::env_mat_nbor_update(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 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**

```cpp
template<typename FTYPE>
void deepmd::gelu_cpu(FTYPE *out, const FTYPE *xx, const int_64 size)
```

**Template Function deepmd::gelu_gpu**

- Defined in file `source_lib/include/gelu.h`

**Function Documentation**

```cpp
template<typename FTYPE>
void deepmd::gelu_cpu(FTYPE *out, const FTYPE *xx, const int_64 size)
```

**Template Function deepmd::gelu_grad_cpu**

- Defined in file `source_lib/include/gelu.h`

**Function Documentation**

```cpp
template<typename FTYPE>
void deepmd::gelu_grad_cpu(FTYPE *out, const FTYPE *xx, const FTYPE *dy, const int_64 size)
```

**Template Function deepmd::gelu_grad_gpu**

- Defined in file `source_lib/include/gelu.h`

**Function Documentation**

```cpp
template<typename FTYPE>
void deepmd::gelu_grad_gpu(FTYPE *out, const FTYPE *xx, const FTYPE *dy, const int_64 size)
```

**Template Function deepmd::gelu_grad_grad_cpu**

- Defined in file `source_lib/include/gelu.h`

**Function Documentation**

```cpp
template<typename FTYPE>
void deepmd::gelu_grad_grad_cpu(FTYPE *out, const FTYPE *xx, const FTYPE *dy, const FTYPE *dy_2, const int_64 size)
```
Template Function `deepmd::gelu_grad_grad_gpu`

- Defined in file `source_lib/include/gelu.h`

**Function Documentation**

```cpp
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**

```cpp
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**

```cpp
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_device_memory(FPTYPE *, std::vector<FPTYPE>&)
• Defined in file_source_lib_include_gpu_rocm.h

Function Documentation

template<typename FPTYPE>
void deepmd::memcpy_device_memory(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 *nats, 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

```cpp
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 null, 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

```cpp
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 null, 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

```cpp
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 null, 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**

```c
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**

```c
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**

- Defined in file_source_lib_include_prod_force.h

**Function Documentation**

```c
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.
• 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.
• 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**

definition

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**

definition

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**

definition

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**

definition

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 nei)

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 nei)

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 nei)

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 nei)
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, const 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, const 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, const 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 \texttt{deepmd::spline3\_switch}

- Defined in file \texttt{source\_lib\_include\_switcher.h}

Function Documentation

\texttt{inline void deepmd::spline3\_switch(double \&vv, double \&dd, const double \&xx, const double \&rmin, const double \&rmax)}

Template Function \texttt{deepmd::spline5\_switch}

- Defined in file \texttt{source\_lib\_include\_switcher.h}

Function Documentation

\texttt{template<typename FPTYPE> inline void deepmd::spline5\_switch(FPTYPE \&vv, FPTYPE \&dd, const FPTYPE \&xx, const float \&rmin, const float \&rmax)}

Template Function \texttt{deepmd::tabulate\_fusion\_se\_a\_cpu}

- Defined in file \texttt{source\_lib\_include\_tabulate.h}

Function Documentation

\texttt{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 \texttt{deepmd::tabulate\_fusion\_se\_a\_gpu}

- Defined in file \texttt{source\_lib\_include\_tabulate.h}

Function Documentation

\texttt{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

```cpp
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

```cpp
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

```cpp
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 *out, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em, const int nloc, const int nnei, const int last_layer_size)
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

```cpp
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

```cpp
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

```cpp
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

```cpp
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

```cpp
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

```cpp
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

```c
void env_mat_a(std::vector<double>& descrpt_a, std::vector<double>& descrpt_a_deriv,
               std::vector<double>& rrij_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

```c
void env_mat_r(std::vector<double>& descrpt_r, std::vector<double>& descrpt_r_deriv,
               std::vector<double>& rrij_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_nvmd.h

Function Documentation

```c
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_nvmd.h

Function Documentation

```c
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_nvmd

- Defined in file_source_lib_include_env_mat_nvmd.h

Function Documentation

template<class T>
void mul_flt_nvmd(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

```cpp
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

```cpp
int omp_get_num_threads()
```

Function `omp_get_thread_num`

- Defined in file `source_lib/include/ewald.h`

Function Documentation

```cpp
int omp_get_thread_num()
```

Template Function `split_flt`

- Defined in file `source_lib/include/env_mat_nvnmmd.h`

Function Documentation

```cpp
template<class T>
void split_flt(T x, int64_t &sign, int64_t &expo, int64_t &mant)
```

### 22.3.5 Variables

Variable `deepmd::ElectrostaticConversion`

- Defined in file `source_lib/include/ewald.h`
Variable Documentation

const double deepmd::ElectrostaticConversion = 14.39964535475696995031

### 22.3.6 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 Documentation


gpuMemset

Define Documentation


gpuMemset

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_nvmd.h

Define Documentation


NBIT_CUTF

Define NBIT_FLTF

• Defined in file_source_lib_include_env_mat_nvmd.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

22.3.7 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
The project DeePMD-kit is licensed under GNU LGPLv3.0.
24.1 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 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,
24.2 Package Contributors

- AngelJia
- AnguseZhang
- Anurag Kumar Singh
- 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
• YWolfeee
• Ye Ding
• Yifan Li
• Yingze Wang
• Yixiao Chen
• 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
• sigbjobo
• tuoping
• wsyxbcl
• ziyao
24.3 Other Credits

- Zhang ZiXuan for designing the Deepmodeling logo.
- Everyone on the Deepmodeling mailing list for contributing to many discussions and decisions!
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


DeePMD-kit


<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>backend</td>
<td>217</td>
</tr>
<tr>
<td>backend.dynamic_metadata</td>
<td>217</td>
</tr>
<tr>
<td>backend.find_tensorflow</td>
<td>217</td>
</tr>
<tr>
<td>backend.read_env</td>
<td>218</td>
</tr>
<tr>
<td>deepmd</td>
<td>219</td>
</tr>
<tr>
<td>deepmd.cluster</td>
<td>225</td>
</tr>
<tr>
<td>deepmd.cluster.local</td>
<td>225</td>
</tr>
<tr>
<td>deepmd.cluster.slurm</td>
<td>226</td>
</tr>
<tr>
<td>deepmd.common</td>
<td>554</td>
</tr>
<tr>
<td>deepmd.descriptor</td>
<td>226</td>
</tr>
<tr>
<td>deepmd.descriptor.descriptor</td>
<td>278</td>
</tr>
<tr>
<td>deepmd.descriptor.hybrid</td>
<td>285</td>
</tr>
<tr>
<td>deepmd.descriptor.loc_frame</td>
<td>291</td>
</tr>
<tr>
<td>deepmd.descriptor.se</td>
<td>295</td>
</tr>
<tr>
<td>deepmd.descriptor.se_a</td>
<td>297</td>
</tr>
<tr>
<td>deepmd.descriptor.se_a_ebd</td>
<td>303</td>
</tr>
<tr>
<td>deepmd.descriptor.se_a_ebd_v2</td>
<td>305</td>
</tr>
<tr>
<td>deepmd.descriptor.se_a_ef</td>
<td>306</td>
</tr>
<tr>
<td>deepmd.descriptor.se_a_mask</td>
<td>312</td>
</tr>
<tr>
<td>deepmd.descriptor.se_atten</td>
<td>316</td>
</tr>
<tr>
<td>deepmd.descriptor.se_atten_v2</td>
<td>321</td>
</tr>
<tr>
<td>deepmd.descriptor.se_r</td>
<td>323</td>
</tr>
<tr>
<td>deepmd.descriptor.se_t</td>
<td>328</td>
</tr>
<tr>
<td>deepmd.entrypoints</td>
<td>332</td>
</tr>
<tr>
<td>deepmd.entrypoints.compress</td>
<td>337</td>
</tr>
<tr>
<td>deepmd.entrypoints.convert</td>
<td>338</td>
</tr>
<tr>
<td>deepmd.entrypoints.doc</td>
<td>338</td>
</tr>
<tr>
<td>deepmd.entrypoints.freeze</td>
<td>338</td>
</tr>
<tr>
<td>deepmd.entrypoints.gui</td>
<td>338</td>
</tr>
<tr>
<td>deepmd.entrypoints.ipi</td>
<td>339</td>
</tr>
<tr>
<td>deepmd.entrypoints.main</td>
<td>339</td>
</tr>
<tr>
<td>deepmd.entrypoints.neighbor_stat</td>
<td>340</td>
</tr>
<tr>
<td>deepmd.entrypoints.test</td>
<td>340</td>
</tr>
<tr>
<td>deepmd.entrypoints.train</td>
<td>341</td>
</tr>
<tr>
<td>deepmd.entrypoints.transfer</td>
<td>342</td>
</tr>
<tr>
<td>deepmd.env</td>
<td>559</td>
</tr>
<tr>
<td>deepmd.env.op_grads_module</td>
<td>659</td>
</tr>
<tr>
<td>deepmd.env.op_module</td>
<td>607</td>
</tr>
<tr>
<td>deepmd.fit</td>
<td>343</td>
</tr>
<tr>
<td>deepmd.fit.dipole</td>
<td>357</td>
</tr>
<tr>
<td>deepmd.fit.dos</td>
<td>360</td>
</tr>
<tr>
<td>deepmd.fit.ener</td>
<td>363</td>
</tr>
<tr>
<td>deepmd.fit.fitting</td>
<td>366</td>
</tr>
<tr>
<td>deepmd.fit.polar</td>
<td>368</td>
</tr>
<tr>
<td>deepmd.infer</td>
<td>372</td>
</tr>
<tr>
<td>deepmd.infer.data_modifier</td>
<td>392</td>
</tr>
<tr>
<td>deepmd.infer.deep_dipole</td>
<td>394</td>
</tr>
<tr>
<td>deepmd.infer.deep_dos</td>
<td>395</td>
</tr>
<tr>
<td>deepmd.infer.deep_eval</td>
<td>398</td>
</tr>
<tr>
<td>deepmd.infer.deep_linear</td>
<td>402</td>
</tr>
<tr>
<td>deepmd.infer.deep_pot</td>
<td>405</td>
</tr>
<tr>
<td>deepmd.infer.deep_tensor</td>
<td>409</td>
</tr>
<tr>
<td>deepmd.infer.deep_wfc</td>
<td>412</td>
</tr>
<tr>
<td>deepmd.infer.ewald_recp</td>
<td>413</td>
</tr>
<tr>
<td>deepmd.infer.model_devi</td>
<td>414</td>
</tr>
<tr>
<td>deepmd.lmp</td>
<td>559</td>
</tr>
<tr>
<td>deepmd.loggers</td>
<td>418</td>
</tr>
<tr>
<td>deepmd.loggers.loggers</td>
<td>419</td>
</tr>
<tr>
<td>deepmd.loss</td>
<td>420</td>
</tr>
<tr>
<td>deepmd.loss.dos</td>
<td>426</td>
</tr>
<tr>
<td>deepmd.loss.ener</td>
<td>427</td>
</tr>
<tr>
<td>deepmd.loss.loss</td>
<td>432</td>
</tr>
<tr>
<td>deepmd.loss.tensor</td>
<td>433</td>
</tr>
<tr>
<td>deepmd.model</td>
<td>434</td>
</tr>
<tr>
<td>deepmd.model.dos</td>
<td>446</td>
</tr>
<tr>
<td>deepmd.model.ener</td>
<td>449</td>
</tr>
<tr>
<td>deepmd.model.frozen</td>
<td>452</td>
</tr>
<tr>
<td>deepmd.model.linear</td>
<td>455</td>
</tr>
<tr>
<td>deepmd.model.model</td>
<td>458</td>
</tr>
<tr>
<td>deepmd.model.model_stat</td>
<td>465</td>
</tr>
<tr>
<td>deepmd.model.multi</td>
<td>466</td>
</tr>
<tr>
<td>deepmd.model.pairtab</td>
<td>469</td>
</tr>
<tr>
<td>deepmd.model.pairwise_dprc</td>
<td>472</td>
</tr>
<tr>
<td>deepmd.model.tensor</td>
<td>475</td>
</tr>
<tr>
<td>deepmd.nvnmd</td>
<td>480</td>
</tr>
<tr>
<td>deepmd.nvnmd.data</td>
<td>480</td>
</tr>
<tr>
<td>deepmd.nvnmd.data.data</td>
<td>481</td>
</tr>
<tr>
<td>deepmd.nvnmd.descriptor</td>
<td>481</td>
</tr>
<tr>
<td>deepmd.nvnmd.descriptor.se_a</td>
<td>482</td>
</tr>
<tr>
<td>deepmd.nvnmd.descriptor.se_atten</td>
<td>482</td>
</tr>
<tr>
<td>Module</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>deepmd.nvnmd.entrypoints</td>
<td>483</td>
</tr>
<tr>
<td>deepmd.nvnmd.entrypoints.freeze</td>
<td>487</td>
</tr>
<tr>
<td>deepmd.nvnmd.entrypoints.mapt</td>
<td>488</td>
</tr>
<tr>
<td>deepmd.nvnmd.entrypoints.train</td>
<td>490</td>
</tr>
<tr>
<td>deepmd.nvnmd.entrypoints.wrap</td>
<td>491</td>
</tr>
<tr>
<td>deepmd.nvnmd.fit</td>
<td>492</td>
</tr>
<tr>
<td>deepmd.nvnmd.fit.ener</td>
<td>492</td>
</tr>
<tr>
<td>deepmd.nvnmd.utils</td>
<td>493</td>
</tr>
<tr>
<td>deepmd.nvnmd.utils.argcheck</td>
<td>497</td>
</tr>
<tr>
<td>deepmd.nvnmd.utils.config</td>
<td>497</td>
</tr>
<tr>
<td>deepmd.nvnmd.utils.encode</td>
<td>500</td>
</tr>
<tr>
<td>deepmd.nvnmd.utils.fio</td>
<td>502</td>
</tr>
<tr>
<td>deepmd.nvnmd.utils.network</td>
<td>504</td>
</tr>
<tr>
<td>deepmd.nvnmd.utils.op</td>
<td>505</td>
</tr>
<tr>
<td>deepmd.nvnmd.utils.weight</td>
<td>505</td>
</tr>
<tr>
<td>deepmd.op</td>
<td>507</td>
</tr>
<tr>
<td>deepmd.train</td>
<td>507</td>
</tr>
<tr>
<td>deepmd.train.run_options</td>
<td>507</td>
</tr>
<tr>
<td>deepmd.train.trainer</td>
<td>508</td>
</tr>
<tr>
<td>deepmd.utils</td>
<td>510</td>
</tr>
<tr>
<td>deepmd.utils.argcheck</td>
<td>519</td>
</tr>
<tr>
<td>deepmd.utils.batch_size</td>
<td>519</td>
</tr>
<tr>
<td>deepmd.utils.compat</td>
<td>520</td>
</tr>
<tr>
<td>deepmd.utils.compress</td>
<td>521</td>
</tr>
<tr>
<td>deepmd.utils.convert</td>
<td>521</td>
</tr>
<tr>
<td>deepmd.utils.data</td>
<td>523</td>
</tr>
<tr>
<td>deepmd.utils.data_system</td>
<td>526</td>
</tr>
<tr>
<td>deepmd.utils.errors</td>
<td>529</td>
</tr>
<tr>
<td>deepmd.utils.finetune</td>
<td>530</td>
</tr>
<tr>
<td>deepmd.utils.graph</td>
<td>530</td>
</tr>
<tr>
<td>deepmd.utils.learning_rate</td>
<td>535</td>
</tr>
<tr>
<td>deepmd.utils.multi_init</td>
<td>536</td>
</tr>
<tr>
<td>deepmd.utils.neighbor_stat</td>
<td>537</td>
</tr>
<tr>
<td>deepmd.utils.network</td>
<td>538</td>
</tr>
<tr>
<td>deepmd.utils.pair_tab</td>
<td>539</td>
</tr>
<tr>
<td>deepmd.utils.parallel_op</td>
<td>540</td>
</tr>
<tr>
<td>deepmd.utils.path</td>
<td>541</td>
</tr>
<tr>
<td>deepmd.utils.plugin</td>
<td>545</td>
</tr>
<tr>
<td>deepmd.utils.random</td>
<td>546</td>
</tr>
<tr>
<td>deepmd.utils.sess</td>
<td>547</td>
</tr>
<tr>
<td>deepmd.utils.spin</td>
<td>547</td>
</tr>
<tr>
<td>deepmd.utils.tabulate</td>
<td>548</td>
</tr>
<tr>
<td>deepmd.utils.type_embed</td>
<td>550</td>
</tr>
<tr>
<td>deepmd.utils.weight_avg</td>
<td>552</td>
</tr>
<tr>
<td>deepmd_utils</td>
<td>560</td>
</tr>
<tr>
<td>deepmd_utils.common</td>
<td>601</td>
</tr>
<tr>
<td>deepmd_utils.entrypoints</td>
<td>560</td>
</tr>
<tr>
<td>deepmd_utils.entrypoints.doc</td>
<td>560</td>
</tr>
<tr>
<td>deepmd_utils.entrypoints.gui</td>
<td>560</td>
</tr>
<tr>
<td>deepmd_utils.env</td>
<td>603</td>
</tr>
<tr>
<td>deepmd_utils.loggers</td>
<td>561</td>
</tr>
<tr>
<td>deepmd_utils.loggers.loggers</td>
<td>562</td>
</tr>
<tr>
<td>deepmd_utils.main</td>
<td>604</td>
</tr>
<tr>
<td>deepmd_utils.model_format</td>
<td>563</td>
</tr>
<tr>
<td>deepmd_utils.model_format.common</td>
<td>571</td>
</tr>
<tr>
<td>deepmd_utils.model_format.env_mat</td>
<td>571</td>
</tr>
<tr>
<td>deepmd_utils.model_format.network</td>
<td>572</td>
</tr>
<tr>
<td>deepmd_utils.model_format.output_def</td>
<td>576</td>
</tr>
<tr>
<td>deepmd_utils.model_format.se_e2_a</td>
<td>578</td>
</tr>
<tr>
<td>deepmd_utils.utils</td>
<td>580</td>
</tr>
<tr>
<td>deepmd_utils.utils.argcheck</td>
<td>580</td>
</tr>
<tr>
<td>deepmd_utils.utils.argcheck_nvnmd</td>
<td>583</td>
</tr>
<tr>
<td>deepmd_utils.utils.batch_size</td>
<td>583</td>
</tr>
<tr>
<td>deepmd_utils.utils.compat</td>
<td>586</td>
</tr>
<tr>
<td>deepmd_utils.utils.data</td>
<td>587</td>
</tr>
<tr>
<td>deepmd_utils.utils.data_system</td>
<td>590</td>
</tr>
<tr>
<td>deepmd_utils.utils.errors</td>
<td>593</td>
</tr>
<tr>
<td>deepmd_utils.utils.model_stat</td>
<td>594</td>
</tr>
<tr>
<td>deepmd_utils.utils.pair_tab</td>
<td>594</td>
</tr>
<tr>
<td>deepmd_utils.utils.path</td>
<td>595</td>
</tr>
<tr>
<td>deepmd_utils.utils.plugin</td>
<td>598</td>
</tr>
<tr>
<td>deepmd_utils.utils.random</td>
<td>600</td>
</tr>
<tr>
<td>deepmd_utils.utils.weight_avg</td>
<td>601</td>
</tr>
</tbody>
</table>
Symbols

_DP_Dee\textit{p}t\textit{c}ompute (C++ function), 739
_DP_Dee\textit{p}t\textit{c}ompute\langle\texttt{double}\rangle (C++ function), 739
_DP_Dee\textit{p}t\textit{c}ompute\langle\texttt{float}\rangle (C++ function), 739
_DP_Dee\textit{p}t\textit{c}ompute\textit{MixedType} (C++ function), 740
_DP_Dee\textit{p}t\textit{c}ompute\textit{MixedType}\langle\texttt{double}\rangle (C++ function), 740
_DP_Dee\textit{p}t\textit{c}ompute\textit{MixedType}\langle\texttt{float}\rangle (C++ function), 740
_DP_Dee\textit{p}t\textit{c}ompute\textit{NList} (C++ function), 741
_DP_Dee\textit{p}t\textit{c}ompute\textit{NList}\langle\texttt{double}\rangle (C++ function), 741
_DP_Dee\textit{p}t\textit{c}ompute\textit{NList}\langle\texttt{float}\rangle (C++ function), 741
_DP_Dee\textit{p}t\textit{c}ompute\textit{Tensor}\textit{NList} (C++ function), 744
_DP_Dee\textit{p}t\textit{c}ompute\textit{Tensor}\textit{NList}\langle\texttt{double}\rangle (C++ function), 744
_DP_Dee\textit{p}t\textit{c}ompute\textit{Tensor}\textit{NList}\langle\texttt{float}\rangle (C++ function), 744
_DP_Dee\textit{p}t\textit{c}ompute\textit{Tensor} (C++ function), 745
_DP_Dee\textit{p}t\textit{c}ompute\textit{Tensor}\langle\texttt{double}\rangle (C++ function), 745
_DP_Dee\textit{p}t\textit{c}ompute\textit{Tensor}\langle\texttt{float}\rangle (C++ function), 745
_DP_Dee\textit{p}t\textit{c}ompute\textit{Tensor}\textit{NList} (C++ function), 745
_DP_Dee\textit{p}t\textit{c}ompute\textit{Tensor}\textit{NList}\langle\texttt{double}\rangle (C++ function), 746
_DP_Dee\textit{p}t\textit{c}ompute\textit{Tensor}\textit{NList}\langle\texttt{float}\rangle (C++ function), 746
_DP_Dipole\textit{c}harge\textit{m}odifier\textit{c}ompute\textit{NList} (C++ function), 746
_DP_Dipole\textit{c}harge\textit{m}odifier\textit{c}ompute\textit{NList}\langle\texttt{double}\rangle (C++ function), 747
_DP_Dipole\textit{c}harge\textit{m}odifier\textit{c}ompute\textit{NList}\langle\texttt{float}\rangle (C++ function), 747
_DP_Get\_\textit{E}nergie\_\textit{P}ointer (C++ function), 747, 748

A

activation\_function:

model[type\_embedding/activation\_function](Argument), 90
model[standard]/descriptor[se\_a\_ebd\_v2]/activation\_function (Argument), 108
model[standard]/descriptor[se\_a\_mask]/activation\_function (Argument), 110
model[standard]/descriptor[se\_a\_tpe]/activation\_function (Argument), 98
model[standard]/descriptor[se\_atten\_v2]/activation\_function (Argument), 105
model[standard]/descriptor[se\_atten]/activation\_function (Argument), 103
model[standard]/descriptor[se\_e2\_a]/activation\_function (Argument), 95
model[standard]/descriptor[se\_e2\_r]/activation\_function (Argument), 100
model[standard]/descriptor[se\_e3]/activation\_function (Argument), 97
model[standard]/fitting\_net[dipole]/activation\_function (Argument), 116
model[standard]/fitting\_net[dos]/activation\_function (Argument), 113
model[standard]/fitting\_net[ener]/activation\_function (Argument), 112
model[standard]/fitting\_net[polar]/activation\_function (Argument), 115
add() (deepmd.utils.data.DeepmdDatamethod), 524
add() (deepmd.utils.data_system.DeepmdDataSystem method), 527
add() (deepmd.utils.DeepmdData method), 511
add() (deepmd.utils.DeepmdDataSystem method), 867
add() (deepmd_utils.utils.data.DeepmdData method), 588
add() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 591
add_data_requirement() (in module deepmd.common), 554
add_data_requirement() (in module deepmd_utils.common), 591
add_dict() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 554
add_dict() (deepmd_utils.utils.data.DeepmdData method), 512
add_flt_nvnmd (C++ function), 805
add_flt_nvnmd() (in module deepmd.env.op_module), 633
AddFltNvnmd() (in module deepmd.env.op_module), 607
Annotated (class in deepmd.env.op_grads_module), 659
Annotated (class in deepmd.env.op_module), 607
ArgsPlugin (class in deepmd_utils.utils.argcheck), 580
atom_ener: model[standard]/fitting_net[ener]/atom_ener (Argument), 113
attn: model[standard]/descriptor[se_atten_v2]/attn (Argument), 106
model[standard]/descriptor[se_atten]/attn (Argument), 102
attn_dotr: model[standard]/descriptor[se_atten_v2]/attn_dotr (Argument), 107
model[standard]/descriptor[se_atten]/attn_dotr (Argument), 104
attn_layer: model[standard]/descriptor[se_atten_v2]/attn_layer (Argument), 106
model[standard]/descriptor[se_atten]/attn_layer (Argument), 104
attn_mask: model[standard]/descriptor[se_atten_v2]/attn_mask (Argument), 107
model[standard]/descriptor[se_atten]/attn_mask (Argument), 104
auto_prob: training/validation_data/auto_prob (Argument), 127
training/validation_data/auto_prob (Argument), 127
AutoBatchSize (class in deepmd_utils.utils.batch_size), 584
Avg (class in deepmd_utils.utils.argcheck), 570
axis_neuron: model[standard]/descriptor[se_a_ebd_v2]/axis_neuron (Argument), 108
model[standard]/descriptor[se_a_mask]/axis_neuron (Argument), 110
model[standard]/descriptor[se_a_tpe]/axis_neuron (Argument), 105
model[standard]/descriptor[se_atten_v2]/axis_neuron (Argument), 102
model[standard]/descriptor[se_atten]/axis_neuron (Argument), 104
axis_rule: model[standard]/descriptor[loc_frame]/axis_rule (Argument), 93
backend module, 217
backend.dynamic_metadata module, 217
backend.find_tensorflow module, 217
backend.read_env module, 218
batch_size: training/validation_data/batch_size (Argument), 125
build() (deepmd.descriptor.descriptor.Descriptor method), 227
build() (deepmd.descriptor.DescriptorHybrid method), 234
build() (deepmd.descriptor.DescrptLocFrame method), 240
bin2hex() (deepmd.nvnmd.utils.Encode method), 493
bin2hex() (deepmd.nvnmd.utils.encode.Encode method), 500
bin2hex_str() (deepmd.nvnmd.utils.Encode method), 493
bin2hex_str() (deepmd.nvnmd.utils.encode.Encode method), 500
build() (deepmd.descriptor.Descriptor method), 279
build() (deepmd.descriptor.DescrptHybrid method), 240
build() (deepmd.descriptor.DescrptLocFrame method), 240
DeePMD-kit

(build.embedding_net(), 489)
(build_grad(), 490)
(build_u2s(), 485)
(build_u2s_grad(), 485)
(byte2hex(), 493)
(cal_coef4(), 490)
(cal_g(), 564)
(calc_model_devi(), 391)
(calc_model_devi_e(), 414)
(calc_model_devi_f(), 415)
(calc_model_devi_v(), 416)
(call(), 554)
(call(), 571)
(call(), 564)
(call(), 565)
(call(), 567)
(call(), 568)
(call(), 573)
call() (deepmd_utils.model_format.se_e2_a.DescrptSeA method), 568
check_type_consistency() (deepmd_utils.model_format.network.NativeLayer method), 574
check_var() (in module deepmd_utils.model_format.output_def), 578
choose() (in module deepmd_utils.utils.random), 546
clear_session() (in module deepmd.common), 555
compress() (in module deepmd_common.entrypoints), 332
compress() (in module deepmd_common.entrypoints.compress), 337
check_dec() (deepmd.nvnmd.utils.Encode method), 494
check_shape() (in module deepmd_utils.model_format.output_def), 578
check_shape_consistency() (deepmd_utils.model_format.NativeLayer method), 567
check_shape_consistency() (deepmd_utils.model_format.network.NativeLayer method), 574
check_switch_range() (in module deepmd.nvnmd.descriptor.se_a), 482
check_switch_range() (in module deepmd.nvnmd.descriptor.se_atten), 482
check_test_size() (deepmd_utils.data.DeepmdData method), 525
check_test_size() (deepmd_utils.data.DeepmdData method), 514
check_test_size() (deepmd_utils.utils.data.DeepmdData system method), 592
check_input_stats() (deepmd.descriptor.Descriptor method), 229
check_input_stats() (deepmd.descriptor.Descriptor method), 281
check_input_stats() (deepmd.descriptor.Descriptor method), 235
check_input_stats() (deepmd.descriptor.DescriptHybrid method), 241
check_input_stats() (deepmd.descriptor.DescriptSeA method), 246
check_input_stats() (deepmd.descriptor.DescriptSeAEf method), 255
check_input_stats() (deepmd.descriptor.DescriptSeAEfLower method), 258
DeePMD-kit

compute_input_stats()
(deepmd.descriptor.DescrptSeAMask method), 261

compute_input_stats()
(deepmd.descriptor.DescrptSeAtten method), 266

compute_input_stats()
(deepmd.descriptor.DescrptSeR method), 272

compute_input_stats()
(deepmd.descriptor.DescrptSeT method), 276

compute_input_stats()
(deepmd.descriptor.hybrid.DescrptHybrid method), 287

compute_input_stats()
(deepmd.descriptor.loc_frame.DescrptLocFrame method), 293

compute_input_stats()
(deepmd.descriptor.se_a.DescrptSeA method), 300

compute_input_stats()
(deepmd.descriptor.se_a_ef.DescrptSeAEf method), 309

compute_input_stats()
(deepmd.descriptor.se_a_ef.DescrptSeAEf method), 312

compute_input_stats()
(deepmd.descriptor.se_a_mask.DescrptSeAMask method), 315

compute_input_stats()
(deepmd.descriptor.se_atten.DescrptSeAtten method), 320

compute_input_stats()
(deepmd.descriptor.se_r.DescrptSeR method), 326

compute_input_stats()
(deepmd.descriptor.se_t.DescrptSeT method), 330

compute_input_stats()
(deepmd.fit.dos.DOSFitting method), 361

compute_input_stats()
(deepmd.fit.DOSFitting method), 344

compute_output_stats()
(deepmd.fit.dos.DOSFitting method), 365

compute_output_stats()
(deepmd.fit.ener.EnerFitting method), 350

compute_output_stats()
(deepmd.fit.ener.EnerFitting method), 350

compute_output_stats()
(training/mixed_precision/compute_prec (Argument), 128

compute_smooth_weight()
(in module deepmd_utils.model_format.env_mat), 572

config_file:

convert() (in module deepmd.entrypoints), 338

convert() (in module deepmd.entrypoints.convert), 338

convert_012_to_21() (in module deepmd_utils.convert), 521

convert_10_to_21() (in module deepmd_utils.convert), 521

convert_12_to_21() (in module deepmd_utils.convert), 521

convert_13_to_21() (in module deepmd_utils.convert), 521

convert_20_to_21() (in module deepmd_utils.convert), 521

convert_dp012_to_dp10() (in module deepmd_utils.convert), 522

convert_dp10_to_dp11() (in module deepmd_utils.convert), 522

convert_dp12_to_dp13() (in module deepmd_utils.convert), 522

convert_dp13_to_dp20() (in module deepmd_utils.convert), 522

convert_dp20_to_dp21() (in module deepmd_utils.convert), 522

convert_forward_map() (in module deepmd.env.op_module), 633

convert_input_v0_v1() (in module deepmd_utils.compat), 520

convert_input_v0_v1() (in module deepmd_utils.compat), 520

convert_input_v1_v2() (in module deepmd_utils.compat), 520

convert_input_v1_v2() (in module deepmd_utils.compat), 586

convert_pb_to_pbtxt() (in module deepmd_utils.compat), 522

convert_pbtxt_to_pb() (in module deepmd_utils.compat), 586
DeePMD-kit

convert_to_21() (in module deepmd.utils.convert), 523

ConvertForwardMap() (in module deepmd.env.op_module), 608
copy_coord (C++ function), 809
copy_flt_nvmd() (in module deepmd.env.op_module), 633
CopyFltNvnmd() (in module deepmd.env.op_module), 608
Counter (class in deepmd_utils.model_format.network), 572
create_file_path() (deepmd.nvnmd.utils.Fio method), 502

data_bias_nsample: model/data_bias_nsample (Argument), 89
data_dict: training/data_dict (Argument), 129
data_stat() (deepmd.model.dos.DOSModel method), 448
data_stat() (deepmd.model.DOSModel method), 436
data_stat() (deepmd.model.ener.EenerModel method), 451
data_stat() (deepmd.model.EenerModel method), 440
data_stat() (deepmd.model.frozen.FrozenModel method), 454
data_stat() (deepmd.model.linear.LinearModel method), 457
data_stat() (deepmd.model.model.Model method), 462
data_stat() (deepmd.model.multi.MultiModel method), 468
data_stat() (deepmd.model.MultiModel method), 443
data_stat() (deepmd.model.pairtab.PairTabModel method), 471
data_stat() (deepmd.model.pairwise_dprc.PairwiseDprc method), 473
data_stat_nbatch: model/data_stat_nbatch (Argument), 89
data_stat_protect: model/data_stat_protect (Argument), 89
DatasetLoader (class in deepmd.train.trainer), 509
dec2bin() (deepmd.nvnmd.util.Encode method), 494
dec2bin() (deepmd.nvnmd.util.encode.Encode method), 500
decay_steps: learning_rate[exp]/decay_steps (Argument), 119
DeepDipole (class in deepmd.infer, 375
DeepDipole (class in deepmd.infer.deep_dipole), 394
DeepDOS (class in deepmd.infer), 372
DeepDOS (class in deepmd.infer.deep_dos), 395
DeepEval (class in deepmd), 219
DeepEval (class in deepmd.infer), 376
DeepEval (class in deepmd.infer.deep_eval), 398
DeepGlobalPolar (class in deepmd.infer), 380
DeepGlobalPolar (class in deepmd.infer.deep_polar), 402
depdeepmd
module, 219
deepmd.calculator
module, 552
depdeepmd.cluster
module, 225
depdeepmd.cluster.local
module, 225
depdeepmd.cluster.slurm
module, 226
depdeepmd.common
module, 554
depdeepmd.descriptor
module, 226
depdeepmd.descriptor.descriptor
module, 278
depdeepmd.descriptor.hybrid
module, 285
depdeepmd.descriptor.loc_frame
module, 291
depdeepmd.descriptor.se
module, 295
depdeepmd.descriptor.se_a
module, 297
depdeepmd.descriptor.se_a_ebd
module, 303
depdeepmd.descriptor.se_a_ebd_v2
module, 305
depdeepmd.descriptor.se_a_ef
module, 306
depdeepmd.descriptor.se_a_mask
module, 312
depdeepmd.descriptor.se_atten
module, 316
depdeepmd.descriptor.se_atten_v2
module, 321
depdeepmd.descriptor.se_r
module, 323
depdeepmd.descriptor.se_t
module, 328
depdeepmd.entrypoints
module, 332

Index 873
DeePMD-kit

deepmd.entrypoints.compress  module, 337
deepmd.entrypoints.convert  module, 338
deepmd.entrypoints.doc  module, 338
deepmd.entrypoints.freeze  module, 338
deepmd.entrypoints.gui  module, 339
deepmd.entrypoints.main  module, 339
deepmd.entrypoints.neighbor_stat  module, 340
deepmd.entrypoints.test  module, 340
deepmd.entrypoints.train  module, 341
deepmd.entrypoints.transfer  module, 342
deepmd.env  module, 559
deepmd.env.op_grads_module  module, 659
deepmd.env.op_module  module, 607
deepmd.fit  module, 343
deepmd.fit.dipole  module, 357
deepmd.fit.dos  module, 360
deepmd.fit.ener  module, 363
deepmd.fit.fitting  module, 366
deepmd.fit.polar  module, 368
deepmd.infer  module, 372
deepmd.infer.data_modifier  module, 392
deepmd.infer.deep_dipole  module, 394
deepmd.infer.deep_dos  module, 395
deepmd.infer.deep_eval  module, 398
deepmd.infer.deep_polar  module, 402
deepmd.infer.deep_pot  module, 405
deepmd.infer.deep_tensor  module, 409
deepmd.infer.deep_wfc  module, 412
deepmd.infer.ewald_recip  module, 413
deepmd.infer.model_deviation  module, 414
deepmd.lmp  module, 559
deepmd.loggers  module, 418
deepmd.loggers.loggers  module, 419
deepmd.loss  module, 420
deepmd.loss.dos  module, 426
deepmd.loss.ener  module, 427
deepmd.loss.loss  module, 432
deepmd.loss.tensor  module, 433
deepmd.model  module, 434
deepmd.model.dos  module, 446
deepmd.model.ener  module, 449
deepmd.model.frozen  module, 452
deepmd.model.linear  module, 455
deepmd.model.model  module, 458
deepmd.model.model_stat  module, 465
deepmd.model.multi  module, 466
deepmd.model.pairtab  module, 469
deepmd.model.pairwise_dprc  module, 472
deepmd.model.tensor  module, 475
deepmd.nvnmd  module, 480
deepmd.nvnmd.data  module, 480
deepmd.nvnmd.data.data  module, 481
deepmd.nvnmd.descriptor  module, 481
deepmd.nvnmd.descriptor.se_a
module, 482

deepmd.nvnmd.descriptor.se_atten
module, 482

deepmd.nvnmd.entrypoints
module, 483

deepmd.nvnmd.entrypoints.freeze
module, 487

deepmd.nvnmd.entrypoints.mapt
module, 488

deepmd.nvnmd.entrypoints.train
module, 490

deepmd.nvnmd.entrypoints.wrap
module, 491

deepmd.nvnmd.fit
module, 492

deepmd.nvnmd.fit.ener
module, 492

deepmd.nvnmd.utils
module, 493

deepmd.nvnmd.utils.argcheck
module, 497

deepmd.nvnmd.utils.config
module, 497

deepmd.nvnmd.utils.encode
module, 500

deepmd.nvnmd.utils.fio
module, 502

deepmd.nvnmd.utils.network
module, 504

deepmd.nvnmd.utils.op
module, 505

deepmd.nvnmd.utils.weight
module, 505

deepmd.op
module, 507

deepmd.train
module, 507

deepmd.train.run_options
module, 507

deepmd.train.trainer
module, 508

deepmd.utils
module, 510

deepmd.utils.argcheck
module, 519

deepmd.utils.batch_size
module, 519

deepmd.utils.compat
module, 520

deepmd.utils.compress
module, 521

deepmd.utils.convert
module, 521

deepmd.utils.data
module, 523

deepmd.utils.data_system
module, 526

deepmd.utils.errors
module, 529

deepmd.utils.finetune
module, 530

deepmd.utils.graph
module, 530

deepmd.utils.learning_rate
module, 535

deepmd.utils.multi_init
module, 536

deepmd.utils.neighbor_stat
module, 537

deepmd.utils.network
module, 538

deepmd.utils.pair_tab
module, 539

deepmd.utils.parallel_op
module, 540

deepmd.utils.path
module, 541

deepmd.utils.plugin
module, 545

deepmd.utils.random
module, 546

deepmd.utils.sess
module, 547

deepmd.utils.spin
module, 547

deepmd.utils.tabulate
module, 548

deepmd.utils.type_embed
module, 550

deepmd.utils.weight_avg
module, 552

deepmd::AtomMap (C++ class), 674

deepmd::AtomMap::AtomMap (C++ function), 674

deepmd::AtomMap::backward (C++ function), 674

deepmd::AtomMap::forward (C++ function), 674

deepmd::AtomMap::get_bkw_map (C++ function), 674

deepmd::AtomMap::get_fwd_map (C++ function), 674

deepmd::AtomMap::get_type (C++ function), 674

deepmd::build_nlist_cpu (C++ function), 810

deepmd::build_nlist_gpu (C++ function), 810

deepmd::check_status (C++ function), 708

deepmd::compute_cell_info (C++ function), 810

deepmd::convert_nlist (C++ function), 810

deepmd::convert_nlist_gpu_device (C++ function), 811
Index
DeePMD-kit

deepmd::DeepPotTF::cutoff (C++ function), 690
deepmd::DeepPotTF::DeepPotTF (C++ function), 689
deepmd::DeepPotTF::dim_aparam (C++ function), 690
deepmd::DeepPotTF::dim_fpam (C++ function), 690
deepmd::DeepPotTF::get_type_map (C++ function), 690
deepmd::DeepPotTF::init (C++ function), 689
deepmd::DeepPotTF::is_aparam_nall (C++ function), 690
deepmd::DeepPotTF::numb_types (C++ function), 690
deepmd::DeepPotTF::numb_types_spin (C++ function), 690
deepmd::DeepPotTF::output_dim (C++ function), 696
deepmd::DeepTensor (C++ class), 693
deepmd::DeepTensor::compute (C++ function), 694, 695
deepmd::DeepTensor::cutoff (C++ function), 696
deepmd::DeepTensor::DeepTensor (C++ function), 693
deepmd::DeepTensor::get_type_map (C++ function), 696
deepmd::DeepTensor::init (C++ function), 693
deepmd::DeepTensor::numb_types (C++ function), 696
deepmd::DeepTensor::output_dim (C++ function), 696
deepmd::DeepTensor::print_summary (C++ function), 693
deepmd::DeepTensor::sel_types (C++ function), 696
deepmd::DeepTensorBase (C++ class), 697
deepmd::DeepTensorBase::DeepTensorBase (C++ function), 698
deepmd::DeepTensorBase::compute (C++ function), 697, 698
deepmd::DeepTensorBase::cutoff (C++ function), 699
deepmd::DeepTensorBase::DeepTensorBase (C++ function), 698
deepmd::DeepTensorBase::get_type_map (C++ function), 699
deepmd::DeepTensorBase::init (C++ function), 698
deepmd::DeepTensorBase::numb_types (C++ function), 699
deepmd::DeepTensorBase::output_dim (C++ function), 699
deepmd::DeepTensorBase::sel_types (C++ function), 699

Index 877
DeePMD-kit

deepmd::DipoleChargeModifierTF::DipoleChargeModifierTF (C++ function), 706

deepmd::DipoleChargeModifierTF::compute (C++ function), 707

deepmd::DipoleChargeModifierTF::cutoff (C++ function), 706

deepmd::DipoleChargeModifierTF::DipoleChargeModifierTF (C++ function), 706

deepmd::DipoleChargeModifierTF::DipoleChargeModifierTF (C++ function), 820

deepmd::DipoleChargeModifierTF::init (C++ function), 706

deepmd::DipoleChargeModifierTF::numb_types (C++ function), 706

deepmd::DipoleChargeModifierTF::sel_types (C++ function), 706

deepmd::dpDot1 (C++ function), 814

deepmd::dpDot2 (C++ function), 814

deepmd::dpDot3 (C++ function), 814

deepmd::dpDot4 (C++ function), 814

deepmd::dpDotv3 (C++ function), 815

deepmd::DPBackend::Paddle (C++ enumerator), 708

deepmd::DPBackend::PyTorch (C++ enumerator), 708

deepmd::DPBackend::TensorFlow (C++ enumerator), 708

deepmd::DPBackend::Unknown (C++ enumerator), 708

deepmd::DPGetDeviceCount (C++ function), 815

deepmd::dprc_pairwise_map_cpu (C++ function), 816

deepmd::ElectrostaticConversion (C++ member), 845

deepmd::ENERGYTYPE (C++ type), 717

deepmd::env_mat_a_cpu (C++ function), 816

deepmd::env_mat_a_nvmd_quantize_cpu (C++ function), 816

deepmd::env_mat_nbor_update (C++ function), 817

deepmd::env_mat_r_cpu (C++ function), 817

deepmd::ewald_recp (C++ function), 817

deepmd::EwaldParameters (C++ struct), 796

deepmd::EwaldParameters::beta (C++ member), 796

deepmd::EwaldParameters::rcut (C++ member), 796

deepmd::EwaldParameters::spacing (C++ member), 796

deepmd::filter_ftype_gpu (C++ function), 817

deepmd::format_nbor_list_gpu (C++ function), 818

deepmd::format_nlist_cpu (C++ function), 818

deepmd::free_nlist_gpu_device (C++ function), 818

deepmd::gelu_cpu (C++ function), 819

deepmd::gelu_gpu (C++ function), 819

deepmd::gelu_grad_cpu (C++ function), 819

deepmd::gelu_grad_gpu (C++ function), 819

deepmd::gelu_grad_grad_cpu (C++ function), 820

deepmd::get_env_nthreads (C++ function), 709

deepmd::group_atoms_cpu (C++ function), 820

deepmd::hpp::convert_nlist (C++ function), 748

deepmd::hpp::convert_pbtxt_to_pb (C++ function), 748

deepmd::hpp::deepmd_exception (C++ struct), 721

deepmd::hpp::deepmd_exception::deepmd_exception (C++ function), 721

deepmd::hpp::DeepPot::DeepPot (C++ class), 725

deepmd::hpp::DeepPot::::DeepPot (C++ function), 725

deepmd::hpp::DeepPot::compute (C++ function), 725--727

deepmd::hpp::DeepPot::compute_avg (C++ function), 728, 729

deepmd::hpp::DeepPot::compute_mixed_type (C++ function), 730

deepmd::hpp::DeepPot::cutoff (C++ function), 730

deepmd::hpp::DeepPot::DeepPot (C++ function), 725

deepmd::hpp::DeepPot::dim_aparam (C++ function), 730

deepmd::hpp::DeepPot::dim_fparam (C++ function), 730

deepmd::hpp::DeepPot::get_type_map (C++ function), 730

deepmd::hpp::DeepPot::init (C++ function), 725

deepmd::hpp::DeepPot::numb_types (C++ function), 730

deepmd::hpp::DeepPot::numb_types_spin (C++ function), 730

deepmd::hpp::DeepPot::print_summary (C++ function), 730

deepmd::hpp::DeepPotModelDevi::DeepPotModelDevi (C++ class), 731

deepmd::hpp::DeepPotModelDevi::::DeepPotModelDevi (C++ function), 731

deepmd::hpp::DeepPotModelDevi::compute (C++ function), 731

deepmd::hpp::DeepPotModelDevi::compute_avg (C++ function), 732

deepmd::hpp::DeepPotModelDevi::compute_relative_std (C++ function), 733

deepmd::hpp::DeepPotModelDevi::compute_relative_std_f (C++ function), 733

deepmd::hpp::DeepPotModelDevi::compute_std (C++ function), 733

deepmd::hpp::DeepPotModelDevi::compute_std_f (C++ function), 733
DeePMD-kit

deepmd::hpp::DeepPotModelDevi::cutoff (C++ function), 732
deepmd::hpp::DeepPotModelDevi::DeepPotModelDevi (C++ function), 731

deepmd::hpp::DeepPotModelDevi::dim_aparam (C++ function), 732

deepmd::hpp::DeepPotModelDevi::dim_fparam (C++ function), 732

deepmd::hpp::DeepPotModelDevi::init (C++ function), 731

deepmd::hpp::DeepPotModelDevi::numb_types (C++ function), 732

deepmd::hpp::DeepPotModelDevi::numb_types_spin (C++ function), 732

deepmd::hpp::DeepTensor (C++ class), 734

deepmd::hpp::DeepTensor::~DeepTensor (C++ function), 734

deepmd::hpp::DeepTensor::compute (C++ function), 734–736

deepmd::hpp::DeepTensor::cutoff (C++ function), 737

deepmd::hpp::DeepTensor::DeepTensor (C++ function), 734

deepmd::hpp::DeepTensor::get_type_map (C++ function), 737

deepmd::hpp::DeepTensor::init (C++ function), 734

deepmd::hpp::DeepTensor::numb_types (C++ function), 737

deepmd::hpp::DeepTensor::output_dim (C++ function), 737

deepmd::hpp::DeepTensor::print_summary (C++ function), 737

deepmd::hpp::DeepTensor::sel_types (C++ function), 737

deepmd::hpp::DipoleChargeModifier (C++ class), 737

deepmd::hpp::DipoleChargeModifier::~DipoleChargeModifier (C++ function), 737

deepmd::hpp::DipoleChargeModifier::DipoleChargeModifier (C++ function), 737

deepmd::hpp::DipoleChargeModifier::compute (C++ function), 738

deepmd::hpp::DipoleChargeModifier::cutoff (C++ function), 738

deepmd::hpp::DipoleChargeModifier::DipoleChargeModifier (C++ function), 737

deepmd::hpp::DipoleChargeModifier::init (C++ function), 738

deepmd::hpp::DipoleChargeModifier::numb_types (C++ function), 738

deepmd::hpp::DipoleChargeModifier::print_summary (C++ function), 738

deepmd::hpp::DipoleChargeModifier::sel_types (C++ function), 738

deepmd::hpp::InputNlist (C++ struct), 721

deepmd::hpp::InputNlist::firstneigh (C++ member), 722

deepmd::hpp::InputNlist::ilist (C++ member), 722

deepmd::hpp::InputNlist::init (C++ function), 722

deepmd::hpp::InputNlist::inum (C++ member), 722

deepmd::hpp::InputNlist::nl (C++ member), 722

deepmd::hpp::InputNlist::numneigh (C++ member), 722

deepmd::hpp::InputNlist::read_file_to_string (C++ function), 748

deepmd::hpp::select_by_type (C++ function), 749

deepmd::hpp::select_map (C++ function), 749

deepmd::init_region_cpu (C++ function), 820

deepmd::InputNlist (C++ struct), 797

deepmd::InputNlist::~InputNlist (C++ function), 797

deepmd::InputNlist::firstneigh (C++ member), 797

deepmd::InputNlist::ilist (C++ member), 797

deepmd::InputNlist::InputNlist (C++ function), 797

deepmd::InputNlist::inum (C++ member), 797

deepmd::InputNlist::numneigh (C++ member), 797

deepmd::invsqrt (C++ function), 821

deepmd::invsqrt<double> (C++ function), 821

deepmd::invsqrt<float> (C++ function), 821

deepmd::load_op_library (C++ function), 709

deepmd::malloc_device_memory (C++ function), 821, 822

deepmd::malloc_device_memory_sync (C++ function), 822

deepmd::map_aparam_cpu (C++ function), 823

deepmd::map_aparam_cpu::numneigh (C++ function), 823

deepmd::memcpy_device_to_host (C++ function), 823, 824

deepmd::memcpy_host_to_device (C++ function), 824

deepmd::memset_device_memory (C++ function), 824

deepmd::model_compatable (C++ function), 709

deepmd::name_prefix (C++ function), 710

deepmd::neighbor_stat_gpu (C++ function), 825

deepmd::NeighborListData (C++ struct), 672

deepmd::NeighborListData::copy_from_nlist (C++ function), 672

deepmd::NeighborListData::firstneigh (C++ member), 673

deepmd::NeighborListData::ilist (C++ member), 673

Index 879
DeePMD-kit

depend::NeighborListData::jlist (C++ member), 673
depend::NeighborListData::make_inlist (C++ function), 672
depend::NeighborListData::numneigh (C++ member), 673
depend::NeighborListData::shuffle (C++ function), 672
depend::NeighborListData::shuffle_exclude_empty (C++ function), 672
depend::normalize_coord_cpu (C++ function), 825
depend::normalize_coord_gpu (C++ function), 825
depend::pair_tab_cpu (C++ function), 825
depend::print_summary (C++ function), 710
depend::prod_env_mat_a_cpu (C++ function), 826
depend::prod_env_mat_a_gpu (C++ function), 826
depend::prod_env_mat_a_nvnmd_quantize_cpu (C++ function), 826
depend::prod_env_mat_r_cpu (C++ function), 827
depend::prod_env_mat_r_gpu (C++ function), 827
depend::prod_force_a_cpu (C++ function), 827, 828
depend::prod_force_a_gpu (C++ function), 829
depend::prod_force_a_cpu (C++ function), 829
depend::prod_force_grad_a_cpu (C++ function), 829
depend::prod_force_grad_r_cpu (C++ function), 829
depend::prod_force_grad_r_cpu (C++ function), 830
depend::prod_force_r_cpu (C++ function), 830
depend::prod_force_r_gpu (C++ function), 830
depend::prod_virial_a_cpu (C++ function), 831
depend::prod_virial_a_cpu (C++ function), 831
depend::prod_virial_g_a_cpu (C++ function), 831
depend::prod_virial_g_r_cpu (C++ function), 831
depend::prod_virial_grad_a_cpu (C++ function), 831
depend::prod_virial_grad_r_cpu (C++ function), 831
depend::prod_virial_grad_r_cpu (C++ function), 831
depend::prod_virial_grad_r_cpu (C++ function), 832
depend::prod_virial_r_cpu (C++ function), 832
depend::prod_virial_r_cpu (C++ function), 832
depend::prod_virial_r_cpu (C++ function), 832
depend::read_file_to_string (C++ function), 710
depend::Region (C++ struct), 797
depend::Region::Region (C++ function), 798
depend::Region::box (C++ member), 798
depend::Region::rec_box (C++ member), 798
depend::Region::Region (C++ function), 798
depend::select_by_type (C++ function), 710
depend::select_map (C++ function), 711, 712
depend::select_map_inv (C++ function), 712
depend::select_real_atoms (C++ function), 713
depend::select_real_atoms_coord (C++ function), 713
depend::session_get_dtype (C++ function), 713
depend::session_get_scalar (C++ function), 714
depend::session_get_vector (C++ function), 714
depend::session_input_tensors (C++ function), 715
depend::session_input_tensors_mixed_type (C++ function), 716
depend::soft_min_switch_cpu (C++ function), 832
depend::soft_min_switch_force_cpu (C++ function), 833
depend::soft_min_switch_force_grad_cpu (C++ function), 833
depend::soft_min_switch_virial_cpu (C++ function), 833
depend::soft_min_switch_virial_grad_cpu (C++ function), 833
depend::spline3_switch (C++ function), 834
depend::spline5_switch (C++ function), 834
depend::STRINGTYPE (C++ type), 717
depend::tabulate_fusion_se_a_cpu (C++ function), 834
depend::tabulate_fusion_se_a_cpu (C++ function), 834
depend::tabulate_fusion_se_a_cpu (C++ function), 835
depend::tabulate_fusion_se_a_cpu (C++ function), 835
depend::tabulate_fusion_se_a_cpu (C++ function), 835
depend::tabulate_fusion_se_a_cpu (C++ function), 836
depend::tabulate_fusion_se_a_cpu (C++ function), 836
depend::tabulate_fusion_se_a_cpu (C++ function), 836
depend::tabulate_fusion_se_a_cpu (C++ function), 837
depend::tabulate_fusion_se_a_cpu (C++ function), 837
depend::tabulate_fusion_se_a_cpu (C++ function), 837
deepmd::tabulate_fusion_se_t_grad_cpu (C++ function), 838
deepmd::tabulate_fusion_se_t_grad_gpu (C++ function), 839
deepmd::tabulate_fusion_se_t_grad_cpu (C++ function), 839
deepmd::tabulate_fusion_se_t_grad_gpu (C++ function), 839
deepmd::test_encoding_decoding_nbor_info_gpu (C++ function), 840
deepmd::tf_exception (C++ struct), 673
deepmd::use_nei_info_cpu (C++ function), 840
deepmd::use_nei_info_gpu (C++ function), 840
deepmd::use_nlist_map (C++ function), 840
deepmd::volume_cpu (C++ function), 841
deepmd::volume_gpu (C++ function), 841
deepmd_utils module, 560
deepmd_utils.common module, 601
deepmd_utils.entrypoints module, 560
deepmd_utils.entrypoints.doc module, 560
deepmd_utils.entrypoints.gui module, 560
deepmd_utils.env module, 603
deepmd_utils.loggers module, 561
deepmd_utils.loggers.loggers module, 562
deepmd_utils.main module, 604
deepmd_utils.model_format module, 563
deepmd_utils.model_format.common module, 571
deepmd_utils.model_format.env_mat module, 571
deepmd_utils.model_format.network module, 572
deepmd_utils.model_format.output_def module, 576
deepmd_utils.model_format.se_e2_a module, 578
deepmd_utils.utils module, 580
deepmd_utils.utils.argcheck module, 580
deepmd_utils.utils.argcheck_nvnmmd module, 583

Index 881
DeePMD-kit

deserialize() (deepmd_utils.model_format.NativeLayer class method), 568
deserialize() (deepmd_utils.model_format.network.NativeLayer class method), 574
deserialize() (deepmd_utils.model_format.network.NetworkCollection class method), 575
deserialize() (deepmd_utils.model_format.network.NetworkCollection class method), 569
deserialize() (deepmd_utils.model_format.se_e2_a.DescrptSeA classmethod), 580
detect_model_version() (in module deepmd.utils.convert), 523
dim_in() (deepmd_utils.model_format.NativeLayer method), 568
dim_in() (deepmd_utils.model_format.network.NativeLayer method), 574
dim_out() (deepmd_utils.model_format.NativeLayer method), 568
dim_out() (deepmd_utils.model_format.network.NativeLayer method), 574
DipoleChargeModifier (class in deepmd), 223
DipoleChargeModifier (class in deepmd.infer), 388
DipoleFittingSeA (class in deepmd.fit), 345
DipoleFittingSeA (class in deepmd.fit.dipole), 357
DipoleModel (class in deepmd.model), 436
DipoleModel (class in deepmd.model.tensor), 475
disp_file: training/disp_file (Argument), 128
disp_freq: training/disp_freq (Argument), 128
disp_message() (deepmd.nvnmd.utils.config.NvnmdConfig method), 498
disp_training: training/disp/training (Argument), 128
display_if_exist() (deepmd.loss.loss.Loss static method), 432
do_derivative() (in module deepmd_utils.model_format.output_def), 578
do_reduce() (in module deepmd_utils.model_format.output_def), 578
doc_train_input() (in module deepmd.entrypoints), 333
doc_train_input() (in module deepmd.entrypoints.doc), 338
doc_train_input() (in module deepmd_utils.entrypoints.doc), 560
DOSFitting (class in deepmd.fit), 343
DOSFitting (class in deepmd.fit.dos), 360
DOSLoss (class in deepmd.loss), 420
DOSLoss (class in deepmd.loss.dos), 426
DOSModel (class in deepmd.model), 434
DOSModel (class in deepmd.model.dos), 446
DOSModel (class in deepmd.model.fit), 345
DOSModel (class in deepmd.model.fit.dos), 360
DOSModel (class in deepmd.model.tensor), 475
DOSModel (class in deepmd.model.fit.dipole), 357
DOSModel (class in deepmd.model.fit.dipole.tensor), 475
DOSModel (class in deepmd.model.fit.dipole.data_modifier), 392
DOSModel (class in deepmd.model.fit.dipole.data_modifier), 392
DOSModel (class in deepmd.fit), 343
DOSModel (class in deepmd.fit.dos), 360
DOSModel (class in deepmd.model), 436
DOSModel (class in deepmd.model.tensor), 475
dotmul_flt_nvnmd (C++ function), 841
dotmul_flt_nvnmd () (in module deepmd.env.op_module), 638
DotmulFltNvnmd () (in module deepmd.env.op_module), 612
DP (class in deepmd.calculator), 552
DP_CHECK_OK (C macro), 785
DP_ConvertPbtxtToPb (C++ function), 750
DP_DeepPot (C++ struct), 722
DP_DeepPot (C++ type), 786
DP_DeepPot::aparam_nall (C++ member), 722
DP_DeepPot::daparam (C++ member), 722
DP_DeepPot::dp (C++ member), 722
dp_DeepPot::dp (C++ member), 722
dp_DeepPot::dparam (C++ member), 722
DP_DeepPot::DP_DeepPot (C++ function), 722
DP_DeepPot::exception (C++ member), 722
DP_DeepPot::exception (C++ member), 722
DP_DeepPotCheckOK (C++ function), 750
DP_DeepPotCompute (C++ function), 750
DP_DeepPotCompute2 (C++ function), 751
DP_DeepPotComputeC (C++ function), 752
DP_DeepPotComputeC2 (C++ function), 753
dp_DeepPotComputeMixedType (C++ function), 754
dp_DeepPotComputeMixedType (C++ function), 755
dp_DeepPotComputeMixedType (C++ function), 755
DP_DeepPotComputeNList (C++ function), 755
DP_DeepPotComputeNList2 (C++ function), 756
DP_DeepPotComputeNList (C++ function), 757
DP_DeepPotComputeNList2 (C++ function), 758
dp_DeepPotGetCutOff (C++ function), 759
dp_DeepPotGetDimAParam (C++ function), 759
dp_DeepPotGetDimFParam (C++ function), 760
dp_DeepPotGetNumTypes (C++ function), 760
dp_DeepPotGetNumTypes (C++ function), 760
dp_DeepPotGetNumTypesSpin (C++ function), 760
dp_DeepPotGetNumbTypesMap (C++ function), 761
dp_DeepPotIsAParamNAll (C++ function), 761
dp_DeepPotModelDevi (C++ struct), 723
DP_DeepPotModelDevi (C++ type), 786
DP_DeepPotModelDevi::aparam_nall (C++ member), 723
DP_DeepPotModelDevi::daparam (C++ member), 723
dp_DeepPotModelDevi::dp (C++ member), 723
dp_DeepPotModelDevi::dparam (C++ member), 723
DP_DeepPotModelDevi::DP_DeepPotModelDevi (C++ function), 723
DP_DeepPotModelDevi::exception (C++ member), 723
dp_DeepPotModelDevi::exception (C++ member), 723
DP_DeepPotModelDevi::exception (C++ member), 761
dp_DeepPotModelDevi::DP_DeepPotModelDevi (C++ function), 762

Index 883
DeePMD-kit

DP_DeepPotModelDeviComputeNList2 (C++ function), 763
DP_DeepPotModelDeviComputeNListf (C++ function), 764
DP_DeepPotModelDeviComputeNListf2 (C++ function), 765
DP_DeepPotModelDeviGetCutoff (C++ function), 766
DP_DeepPotModelDeviGetDimAParam (C++ function), 766
DP_DeepPotModelDeviGetDimFParam (C++ function), 766
DP_DeepPotModelDeviGetNumbTypes (C++ function), 767
DP_DeepPotModelDeviGetNumbTypesSpin (C++ function), 767
DP_DeepPotModelDeviIsAParamNAll (C++ function), 767
DP_DeepTensor (C++ struct), 723
DP_DeepTensor (C++ type), 786
DP_DeepTensor::DP_DeepTensor (C++ function), 723
DP_DeepTensor::dt (C++ member), 724
DP_DeepTensor::exception (C++ member), 724
DP_DeepTensor::Compute (C++ function), 768
DP_DeepTensor::Computef (C++ function), 769
DP_DeepTensor::ComputeNList (C++ function), 769
DP_DeepTensor::ComputeNListf (C++ function), 770
DP_DeepTensor::ComputeTensor (C++ function), 771
DP_DeepTensor::ComputeTensorf (C++ function), 771
DP_DeepTensor::ComputeTensorNList (C++ function), 771
DP_DeepTensor::ComputeTensorNListf (C++ function), 772
DP_DeepTensor::GetCutoff (C++ function), 772
DP_DeepTensor::GetNumbSelTypes (C++ function), 773
DP_DeepTensor::GetNumbTypes (C++ function), 774
DP_DeepTensor::GetOutputDim (C++ function), 774
DP_DeepTensor::GetSelTypes (C++ function), 774
DP_DeepTensor::GetTypeMap (C++ function), 775
DP_DipoleChargeModifier (C++ struct), 724
DP_DipoleChargeModifier (C++ type), 787
DP_DipoleChargeModifier::dcm (C++ member), 724
DP_DipoleChargeModifier::DP_DipoleChargeModifier (C++ function), 724
DP_DipoleChargeModifier::exception (C++ member), 724
DP_DipoleChargeModifierCheckOK (C++ function), 775
DP_DipoleChargeModifierComputeNList (C++ function), 776
DP_DipoleChargeModifierComputeNListf (C++ function), 776
DP_DipoleChargeModifierGetCutoff (C++ function), 777
DP_DipoleChargeModifierGetNumbSelTypes (C++ function), 778
DP_DipoleChargeModifierGetNumbTypes (C++ function), 778
DP_DipoleChargeModifierGetSelTypes (C++ function), 778
dp_ipi () (in module deepmd.entrypoints.ipi), 339
DP_NewOK (C macro), 785
DP_NewDeepPot (C++ function), 779
DP_NewDeepPotModelDevi (C++ function), 779
DP_NewDeepPotModelDeviWithParam (C++ function), 779
DP_NewDeepPotWithParam (C++ function), 780
DP_NewDeepPotWithParam2 (C++ function), 780
DP_NewDeepTensor (C++ function), 781
DP_NewDeepTensorWithParam (C++ function), 781
DP_NewDipoleChargeModifier (C++ function), 781
DP_NewDipoleChargeModifierWithParam (C++ function), 782
DP_NewNlist (C++ function), 782
DP_Nlist (C++ struct), 724
DP_Nlist (C++ type), 787
DP_Nlist::DP_Nlist (C++ function), 725
DP_Nlist::exception (C++ member), 725
DP_Nlist::nl (C++ member), 725
DP_NlistCheckOK (C++ function), 783
DP_PrintSummary (C++ function), 783
DP_ReadFileToChar (C++ function), 783
DP_ReadFileToChar2 (C++ function), 784
DP_REQUIRES_OK (C macro), 786
DP_SelectByType (C++ function), 784
DP_SelectMapInt (C++ function), 785
DPAssert (C++ function), 841
DPErrcheck (C macro), 845
DPH5Path (class in deepmd.utils.path), 541
DPH5Path (class in deepmd.utils.path), 595
DPOSPath (class in deepmd.utils.path), 542
DPOSPath (class in deepmd.utils.path), 596
DPath (class in deepmd.utils.path), 543
DPath (class in deepmd.utils.path), 597
dprc_pairwise_idx () (in module deepmd.env.op_module), 638
dprcPairwiseIdx () (in module deepmd.env.op_module), 613
DP_Tabulate (class in deepmd.utils.tabulate), 548
DP_Trainer (class in deepmd.train.trainer), 508
dynamic_metadata () (in module backend.dynamic_metadata), 217

884 Index
embed_atom_type() (in module deepmd.utils.type_embed), 551
embedding_net() (in module deepmd.utils.network), 538
embedding_net_rand_seed_shift() (in module deepmd.utils.network), 539
EmbeddingNet (in module deepmd_utils.model_format), 565
EmbeddingNet (in module deepmd_utils.model_format.network), 573
enable: nvnmd-enable (Argument), 130
enable_atom_ener_coeff:
  loss[ener_spin]/enable_atom_ener_coeff (Argument), 123
  loss[ener]/enable_atom_ener_coeff (Argument), 121
enable_compression() (deepmd.descriptor.Descriptor method), 230
enable_compression() (deepmd.descriptor.descriptor.Descriptor method), 282
enable_compression() (deepmd.descriptor.DescrptHybrid method), 235
enable_compression() (deepmd.descriptor.DescrptSeA method), 246
enable_compression() (deepmd.descriptor.DescrptSeAtten method), 266
enable_compression() (deepmd.descriptor.DescrptSeR method), 272
enable_compression() (deepmd.descriptor.DescrptSeT method), 276
enable_compression() (deepmd.descriptor.hybrid.DescrptHybrid method), 287
enable_compression() (deepmd.descriptor.se_a.DescrptSeA method), 300
enable_compression() (deepmd.descriptor.se_attn.DescrptSeAtten method), 320
enable_compression() (deepmd.descriptor.se_r.DescrptSeR method), 326
enable_compression() (deepmd.descriptor.se_t.DescrptSeT method), 330
enable_compression() (deepmd.model.frozen.FrozenModel method), 454
enable_compression() (deepmd.model.linear.LinearModel method), 457
enable_compression() (deepmd.model.model.Model method), 462
enable_compression() (deepmd.model.model.StandardModel method), 464
enable_compression() (deepmd.model.pairtab.PairTabModel method), 471
enable_compression() (deepmd.model.pairwise_dprc.PairwiseDPRc method), 473
enable_mixed_precision() (deepmd.descriptor.Descriptor method), 230
enable_mixed_precision() (deepmd.descriptor.descriptor.Descriptor method), 282
enable_mixed_precision() (deepmd.descriptor.DescrptHybrid method), 236
enable_mixed_precision() (deepmd.descriptor.DescrptSeA method), 247
enable_mixed_precision() (deepmd.descriptor.hybrid.DescrptHybrid method), 288
enable_mixed_precision() (deepmd.descriptor.se_a.DescrptSeA method), 301
enable_mixed_precision() (deepmd.fit.dipole.DipoleFittingSeA method), 359
enable_mixed_precision() (deepmd.fit.DipoleFittingSeA method), 347
enable_mixed_precision() (deepmd.fit.dos.DOSFitting method), 362
enable_mixed_precision() (deepmd.fit.DOSFitting method), 345
enable_mixed_precision() (deepmd.fit.ener.EnerFitting method), 366
enable_mixed_precision() (deepmd.fit.EnerFitting method), 351
enable_mixed_precision()
enable_mixed_precision() (deepmd.fit.GlobalPolarFittingSeA method), 354
enable_mixed_precision() (deepmd.fit.polar.GlobalPolarFittingSeA method), 369
enable_mixed_precision() (deepmd.fit.polar.PolarFittingSeA method), 372
enable_mixed_precision() (deepmd.model.model.Model method), 462
enable_mixed_precision() (deepmd.model.model.StandardModel method), 464
enable_mixed_precision() (deepmd.model.multi.MultiModel method), 468
enable_mixed_precision() (deepmd.model.MultiModel method), 443
eval() (deepmd.infer.DeepGlobalPolar method), 381
eval() (deepmd.infer.DeepPot method), 384
eval() (deepmd.infer.DipoleChargeModifier method), 389
eval() (deepmd.infer.ewald_recp.EwaldRecp method), 414
eval() (deepmd.infer.EwaldRecp method), 421
eval() (deepmd.loss.dos.DOSLoss method), 427
eval() (deepmd.loss.DOSLoss method), 420
eval() (deepmd.loss.ener.EnerDipoleLoss method), 428
eval() (deepmd.loss.ener.EnerSpinLoss method), 429
eval() (deepmd.loss.ener.EnerStdLoss method), 431
eval() (deepmd.loss.EnerDipoleLoss method), 421
eval() (deepmd.loss.EnerSpinLoss method), 423
eval() (deepmd.loss.EnerStdLoss method), 425
eval() (deepmd.loss.loss.Loss method), 432

eval() (deepmd.loss.tensor.TensorLoss method), 426
eval_descriptor() (deepmd.infer.deep_dos.DeepDOS method), 397
eval_descriptor() (deepmd.infer.deep_pot.DeepPot method), 408
eval_descriptor() (deepmd.infer.DeepDOS method), 374
eval_descriptor() (deepmd.infer.DeepPot method), 385
eval_full() (deepmd.infer.deep_tensor.DeepTensor method), 411
eval_single_list() (deepmd.train.trainer.DPTrainer static method), 508
eval_typeebd() (deepmd.DeepEval method), 220
eval_typeebd() (deepmd.infer.deep_eval.DeepEval method), 400
eval_typeebd() (deepmd.infer.DeepEval method), 378
ewald_beta:
  model/modifier[dipole_charge]/ewald_beta (Argument), 91
  model/modifier[dipole_charge]/ewald_h (Argument), 92
  model/modifier[dipole_charge]/ewald_h (Argument), 92
  model/modifier[dipole_charge]/ewald_recp() (in module deepmd.env.op_module), 638
  EwaldRecp (class in deepmd.infer), 390
  EwaldRecp (class in deepmd.infer.ewald_recp), 413
  EwaldRecp() (in module deepmd.env.op_module), 613
exclude_types:
  model[standard]/descriptor[se_a_ebd_v2]/exclude_types (Argument), 109
  model[standard]/descriptor[se_a_mask]/exclude_types (Argument), 110

eval() (deepmd.DipoleChargeModifier method), 224
eval() (deepmd.infer.data_modifier.DipoleChargeModifier method), 393
eval() (deepmd.infer.deep_dos.DeepDOS method), 396
eval() (deepmd.infer.deep_polar.DeepGlobalPolar method), 403
eval() (deepmd.infer.deep_pot.DeepPot method), 407
eval() (deepmd.infer.deep_tensor.DeepTensor method), 410
eval() (deepmd.infer.DeepDOS method), 373
DeePMD-kit

model[standard]/descriptor[se_a_tpe]/exclude_types (Argument), 99
model[standard]/descriptor[se_atten_v2]/exclude_types (Argument), 106
model[standard]/descriptor[se_atten]/exclude_types (Argument), 103
model[standard]/descriptor[se_e2_a]/exclude_types (Argument), 96
model[standard]/descriptor[se_e2_r]/exclude_types (Argument), 101
execute() (deepmd_utils.utils.batch_size.AutoBatchSize method), 584
execute_all() (deepmd_utils.utils.batch_size.AutoBatchSize method), 585
exits() (deepmd.nvnmd.utils.fio.Fio method), 502
expand_sys_str() (in module deepmd.common), 555
expand_sys_str() (in module deepmd.nvnmd.utils.fio), 502
explicit_ntypes (deepmd.descriptor.Descriptor property), 230
explicit_ntypes (deepmd.descriptor.descriptor.Descriptor property), 282
explicit_ntypes (deepmd.descriptor.descriptor.DescrptHybrid property), 236
explicit_ntypes (deepmd.descriptor.DescrptSeA property), 247
explicit_ntypes (deepmd.descriptor.DescrptSeAtten property), 267
explicit_ntypes (deepmd.descriptor.hybrid.DescrptSeAtten property), 288
explicit_ntypes (deepmd.descriptor.se_a.DescrptSeA property), 301
explicit_ntypes (deepmd.descriptor.se_atten.DescrptSeAtten property), 321
extend_bin() (deepmd.nvnmd.utilsEncode method), 494
extend_bin() (deepmd.nvnmd.utils.encode.Encode method), 501
extend_hex() (deepmd.nvnmd.utils.encode.Encode method), 494
extend_hex() (deepmd.nvnmd.utils.encode.Encode method), 501
extend_list() (deepmd.nvnmd.utils.encode.Encode method), 494
extend_list() (deepmd.nvnmd.utils.encode.Encode method), 501
filter_GR2D() (in module deepmd.nvnmd.descriptor.se_a), 482
filter_GR2D() (in module deepmd.nvnmd.descriptor.se_atten), 482
find_max_expo() (C++ function), 842
find_max_expo() (deepmd.nvnmd.utils.Encode method), 501
find_max_expo() (deepmd.nvnmd.utils.encode.Encode method), 501
fitting_check_output() (in module deepmd_utils.model_format.output_def), 578
fitting_check_output() (in module deepmd_utils.model_format), 570
fitting_dipole() (in module deepmd_utils.utils.argcheck), 582
fitting_dos() (in module deepmd_utils.utils.argcheck), 582
fitting_ener() (in module deepmd_utils.utils.argcheck), 582
fitting_net: model[standard]/fitting_net[polar]/fit_diag (Argument), 115
fitting_net_dict: model[standard]/fitting_net (Argument), 111
fitting_polar() (in module deepmd_utils.utils.argcheck), 582
fitting_variant_type_args() (in module deepmd_utils.utils.argcheck), 582
fitting_weight: training/fitting_weight (Argument), 129
Fitting (class in deepmd.fit), 351
Fitting (class in deepmd.fit.fitting), 366
Fio (class in deepmd.nvnmd.utils.fio), 502
FioBin (class in deepmd.nvnmd.utils), 495
FioDic (class in deepmd.nvnmd.utils), 503
FioJsonDic (class in deepmd.nvnmd.utils), 504
FioNpyDic (class in deepmd.nvnmd.utils), 504
FioTxt (class in deepmd.nvnmd.utils), 504
fit_diag: model[standard]/fitting_net[polar]/fit_diag (Argument), 115
fitting_check_output() (in module deepmd_utils.model_format.output_def), 578
fitting_dipole() (in module deepmd_utils.utils.argcheck), 582
fitting_dos() (in module deepmd_utils.utils.argcheck), 582
fitting_ener() (in module deepmd_utils.utils.argcheck), 582
fitting_net: model[standard]/fitting_net (Argument), 111
fitting_net_dict: model[standard]/fitting_net (Argument), 111
fitting_polar() (in module deepmd_utils.utils.argcheck), 582
fitting_variant_type_args() (in module deepmd_utils.utils.argcheck), 582
fitting_weight: training/fitting_weight (Argument), 129
FittingNet (in module deepmd_utils.model_format), 566
FittingNet (in module deepmd_utils.model_format.network), 566
Index 887
DeePMD-kit

FittingOutputDef (class in deepmd_utils.model_format), 566
FittingOutputDef (class in deepmd_utils.model_format.output_def), 576
flt2bin() (deepmd.nvnmd.utils.Encode method), 494
flt2bin() (deepmd.nvnmd.utils.encode.Encode method), 501
flt2bin_one() (deepmd.nvnmd.utils.Encode method), 494
flt2bin_one() (deepmd.nvnmd.utils.encode.Encode method), 501
FLT_MASK (C macro), 845
flt_nvnmd() (in module deepmd.env.op_module), 639
FltNvnmd() (in module deepmd.env.op_module), 613
format_nlist_i_cpu (C++ function), 843
format_nlist_i_fill_a (C++ function), 843
freeze() (in module deepmd.entrypoints), 333
freeze() (in module deepmd.entrypoints.freeze), 338
frozen_model_args() (in module deepmd_utils.utils.argcheck), 582
FrozenModel (class in deepmd.model.frozen), 452
G

gather_placeholder() (in module deepmd.model.pairwise_dprc), 474
gelu() (in module deepmd.common), 556
Gelu() (in module deepmd.env.op_module), 614
gelu() (in module deepmd.env.op_module), 639
gelu_custom() (in module deepmd.env.op_module), 639
gelu_grad() (in module deepmd.env.op_module), 639
gelu_grad_custom() (in module deepmd.env.op_module), 639
gelu_grad_grad() (in module deepmd.env.op_module), 640
gelu_grad() (in module deepmd.env.op_module), 640
gelu_grad_custom() (in module deepmd.env.op_module), 640
gelu_tfp() (in module deepmd.common), 556
GeluCustom() (in module deepmd.env.op_module), 614
GeluGPUEXecuteFunctor (C++ struct), 798
GeluGPUEXecuteFunctor::operator() (C++ function), 799
GeluGrad() (in module deepmd.env.op_module), 614
GeluGradCustom() (in module deepmd.env.op_module), 615
GeluGradGPUEXecuteFunctor (C++ struct), 799
GeluGradGPUEXecuteFunctor::operator() (C++ function), 799
gen_args() (in module deepmd_utils.utils.argcheck), 519
gen_args() (in module deepmd_utils.utils.argcheck), 582
gen_doc() (in module deepmd_utils.utils.argcheck), 519
gen_doc() (in module deepmd_utils.utils.argcheck), 582
gen_json() (in module deepmd_utils.utils.argcheck), 519
gen_json() (in module deepmd_utils.utils.argcheck), 582
generate() (deepmd.utils.parallel_op.ParallelOp method), 541
get() (deepmd.env.op_module.fio.FioDict method), 503
get() (deepmd.env.op_module.fio.FioDict method), 495
g get() (deepmd.env.op_module.PairTab method), 539
get() (deepmd.env.op_module.PairTab method), 517
get() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 512
get() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 589
get_attention_layer_nodes_from_graph_def() (in module deepmd.env.op_module), 530
get_attention_layer_variables_from_graph_def() (in module deepmd.env.op_module), 530
get_batch() (deepmd.env.op_module.data.DeepmdData method), 525
get_batch() (deepmd.env.op_module.data.DeepmdData method), 528
get_batch() (deepmd.env.op_module.data.DeepmdData method), 512
get_batch() (deepmd.env.op_module.data.DeepmdData method), 589
get_batch() (deepmd.env.op_module.data_system.DeepmdDataSystem method), 525
get_batch() (deepmd.env.op_module.data_system.DeepmdDataSystem method), 528
get_batch() (deepmd.env.op_module.data_system.DeepmdDataSystem method), 592
DeePMD-kit

get_batch_mixed() (deepmd.utils.data_system.DeepmdDataSystem method), 528
get_batch_mixed() (deepmd.utils.DeepmdDataSystem method), 515
get_batch_mixed() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 592
get_batch_size() (deepmd.utils.data_system.DeepmdDataSystem method), 528
get_batch_size() (deepmd.utils.DeepmdDataSystem method), 515
get_batch_size() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 592
get_batch_standard() (deepmd.utils.data_system.DeepmdDataSystem method), 528
get_batch_standard() (deepmd.utils.DeepmdDataSystem method), 515
get_batch_standard() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 592
get_class_by_input() (deepmd.descriptor.Descriptor class method), 230
get_class_by_input() (deepmd.descriptor.descriptor.Descriptor class method), 282
get_class_by_input() (deepmd.model.model.Model class method), 462
get_constant_initializer() (in module deepmd.nvnmd.utils.weight), 505
get_data() (deepmd_utils.model_format.FittingOutputDef method), 566
get_data() (deepmd_utils.model_format.ModelOutputDef method), 566
get_data() (deepmd_utils.model_format.output_def.FittingOutputDef method), 576
get_data() (deepmd_utils.model_format.output_def.ModelOutputDef method), 577
get_data() (deepmd.train.trainer.DatasetLoader method), 509
get_data() (deepmd.utils.DeepmdData method), 525
get_data() (deepmd.utils.data_system.DeepmdData method), 528
get_data() (deepmd.utils.DeepmdData method), 512
get_data() (deepmd_utils.utils.data.DeepmdData method), 589
get_data() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 592
get_deriv_name() (in module deepmd_utils.model_format), 570
get_deriv_name() (in module deepmd_utils.model_format.output_def), 578
get_descriptor_type() (deepmd.infer.deep_pot.DeepPot method), 409
get_descriptor_type() (deepmd.infer.DeepPot method), 386
get_dim_aparam() (deepmd.infer.deep_dipole.DeepDipole method), 395
get_dim_aparam() (deepmd.infer.deep_dos.DeepDOS method), 398
get_dim_aparam() (deepmd.infer.deep_polar.DeepGlobalPolar method), 403
get_dim_aparam() (deepmd.infer.deep_polar.DeepPolar method), 405
get_dim_aparam() (deepmd.infer.deep_pot.DeepPot method), 409
get_dim_aparam() (deepmd.infer.deep_tensor.DeepTensor method), 412
get_dim_aparam() (deepmd.infer.deep_wfc.DeepWFC method), 413
get_dim_aparam() (deepmd.infer.DeepDipole method), 376
get_dim_aparam() (deepmd.infer.DeepDOS method), 375
get_dim_fparam() (deepmd.infer.deep_dipole.DeepDipole method), 395
get_dim_fparam() (deepmd.infer.deep_dos.DeepDOS method), 398
get_dim_fparam() (deepmd.infer.deep_polar.DeepGlobalPolar method), 404
get_dim_fparam() (deepmd.infer.deep_polar.DeepPolar method), 405
get_dim_fparam() (deepmd.infer.deep_pot.DeepPot method), 409
get_dim_fparam() (deepmd.infer.deep_tensor.DeepTensor method), 412
get_dim_fparam() (deepmd.infer.deep_wfc.DeepWFC method), 413
get_fparam() (deepmd.infer.deep_dipole.DeepDipole method), 376
get_fparam() (deepmd.infer.deep_dos.DeepDOS method), 375
get_fparam() (deepmd.infer.deep_polar.DeepGlobalPolar method), 381
get_fparam() (deepmd.infer.deep_polar.DeepPolar method), 383
get_fparam() (deepmd.infer.deep_pot.DeepPot method), 388
get_fparam() (deepmd.infer.deep_tensor.DeepTensor method), 395
get_fparam() (deepmd.infer.deep_wfc.DeepWFC method), 399
get_model() (deepmd.infer.DeepDipole method), 376
get_model() (deepmd.infer.DeepDOS method), 375
get_model() (deepmd.infer.deep_dipole.DeepDipole method), 395
get_model() (deepmd.infer.deep_dos.DeepDOS method), 398
get_model() (deepmd.infer.deep_polar.DeepGlobalPolar method), 404
get_model() (deepmd.infer.deep_polar.DeepPolar method), 405
get_model() (deepmd.infer.deep_pot.DeepPot method), 409
get_model() (deepmd.infer.deep_tensor.DeepTensor method), 412
get_model() (deepmd.infer.deep_wfc.DeepWFC method), 413
get_model() (deepmd.infer.DeepDipole method), 376

DeePMD-kit

get_dim_fparam() (deepmd.infer.DeepDOS method), 375
get_dim_fparam() (deepmd.infer.DeepGlobalPolar method), 382
get_dim_fparam() (deepmd.infer.DeepPolar method), 383
get_dim_fparam() (deepmd.infer.DeepPot method), 386
get_dim_fparam() (deepmd.infer.DeepWFC method), 388
get_dim_out() (deepmd.descriptor.Descriptor method), 230
get_dim_out() (deepmd.descriptor.descriptor.Descriptor method), 282
get_dim_out() (deepmd.descriptor.DescriptorSeA method), 247
get_dim_out() (deepmd.descriptor.DescriptorSeAEf method), 255
get_dim_out() (deepmd.descriptor.DescriptorSeR method), 277
get_dim_out() (deepmd.descriptor.DescriptorSeT method), 293
get_dim_out() (deepmd.descriptor.se_a.DescriptorSeA method), 301
get_dim_out() (deepmd.descriptor.se_a_ef.DescriptorSeAEf method), 309
get_dp_init_weights() (deepmd.nvnmd.utils.config.NvnmdConfig method), 498
get_dscp_jdata() (deepmd.nvnmd.utils.config.NvnmdConfig method), 498
get_embedding_net_nodes() (in module deepmd.utils.graph), 530
get_embedding_net_nodes_from_graph_def() (in module deepmd.utils.graph), 531
get_embedding_net_variables() (in module deepmd.utils.graph), 531
get_embedding_net_variables_from_graph_def() (in module deepmd.utils.graph), 531
get_env() (in module deepmd.utils.graph), 508
get_evaluation_results() (deepmd.train.trainer.DPTrainer method), 508
get_extra_embedding_net_suffix() (in module deepmd.utils.graph), 531
get_extra_embedding_net_variables_from_graph_def() (in module deepmd.utils.graph), 531
get_extra_side_embedding_net_variable() (in module deepmd.utils.graph), 521
get_feed_dict() (deepmd.model.model.Model method), 462
get_feed_dict() (deepmd.model.pairwise_dprc.PairwiseDPRc method), 473
get_feed_dict() (deepmd.train.trainer.DPTrainer method), 508
get_file_list() (deepmd.nvnmd.utils.fio.Fio method), 502
get_file_list() (deepmd.nvnmd.utils.fio.Fio method), 502
get_filter_type_weight() (in module deepmd.nvnmd.utils.weight), 505
get_filter_weight() (in module deepmd.nvnmd.utils.weight), 506
get_filter_weight() (in module deepmd.nvnmd.utils.weight), 506
get_fitn_jdata() (deepmd.nvnmd.utils.config.NvnmdConfig method), 498
get_fitnet_weight() (in module deepmd.nvnmd.utils.weight), 506
get_fitnet_weight() (in module deepmd.nvnmd.utils.weight), 506
get_fitting() (deepmd.model.frozen.FrozenModel method), 454
get_fitting() (deepmd.model.linear.LinearModel method), 457
get_fitting() (deepmd.model.model.Model method), 462
get_fitting() (deepmd.model.model.StandardModel method), 465
get_fitting() (deepmd.model.multi.MultiModel method), 468
Index 891

get_fitting() (deepmd.model.MultiModel method), 468
get_fitting() (deepmd.model.pairtab.PairTabModel method), 471
get_fitting() (deepmd.model.pairwise_dprc.PairwiseDPRc method), 474
get_fitting_net_nodes() (in module deepmd.utils.graph), 532
get_fitting_net_variables() (in module deepmd.utils.graph), 533
get_gpus() (in module deepmd.cluster.local), 225
get_loss() (deepmd.model.MultiModel method), 443
get_loss() (deepmd.model.pairtab.PairTabModel method), 471
get_loss() (deepmd.model.pairwise_dprc.PairwiseDPRc method), 474
get_loss_jdata() (deepmd.nvnmd.utils.config.NvnmdConfig method), 498
get_model_jdata() (deepmd.nvnmd.utils.config.NvnmdConfig method), 498
get_natoms() (deepmd.utils.data.DeepmdData method), 508
get_natoms() (deepmd.utils.data.DeepmdData method), 512
get_natoms() (deepmd.utils.data.DeepmdData method), 525
get_natoms() (deepmd.utils.data.DeepmdData method), 525
get_natoms_vec() (deepmd.utils.data.DeepmdData method), 512
get_natoms_vec() (deepmd.utils.data.DeepmdData method), 525
get_nlist() (deepmd.descriptor.descriptor.DescrptSeA method), 301
get_nlist() (deepmd.descriptor.descriptor.DescrptSeAEf method), 309
get_nlist() (deepmd.descriptor.descriptor.DescrptSeR method), 327
get_nlist() (deepmd.descriptor.descriptor.DescrptSeT method), 372
get_nlist() (deepmd.descriptor.descriptor.DescrptSeT.method), 372
get_nlist() (deepmd.descriptor.descriptor.DescrptSeT.method), 372
get_nlist() (deepmd.descriptor.descriptor.DescrptSeT.method), 372
DeePMD-kit

get_nlist() (deepmd.descriptor.se_t.DescrptSeT method), 331
get_nlist_i() (deepmd.descriptor.DescrptHybrid method), 236
get_nlist_i() (deepmd.descriptor.hybrid.DescrptHybrid method), 288
get_normalize() (in module deepmd.nvnmd.utils.weight), 506
get_np_precision() (in module deepmd.common), 556
get_np_precision() (in module deepmd_utils.common), 602
get_nsystems() (deepmd.utils.data_system.DeepmdDataSystem method), 529
get_nsystems() (deepmd.utils.DeepmdDataSystem method), 515
get_nsystems() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 593
get_ntypes() (deepmd.descriptor.Descriptor method), 231
get_ntypes() (deepmd.descriptor.descriptor.Descriptor method), 283
get_ntypes() (deepmd.descriptor.DescrptHybrid method), 247
get_ntypes() (deepmd.descriptor.DescrptLocFrame method), 247
get_ntypes() (deepmd.descriptor.DescrptSeA method), 255
get_ntypes() (deepmd.descriptor.DescrptSeAEf method), 305
get_ntypes() (deepmd.descriptor.se_a.DescrptSeA method), 301
get_ntypes() (deepmd.descriptor.se_a_ef.DescrptSeAEf method), 309
get_ntypes() (deepmd.descriptor.se_r.DescrptSeR method), 327
get_ntypes() (deepmd.descriptor.se_t.DescrptSeT method), 331
get_ntypes() (deepmd.infer.deep_dos.DeepDOS method), 398
get_ntypes() (deepmd.infer.deep_pot.DeepPot method), 409
get_ntypes() (deepmd.infer.deep_tensor.DeepTensor method), 412
get_ntypes() (deepmd.infer.DeepDOS method), 375
get_ntypes() (deepmd.infer.DeepPot method), 386
get_numb_aparam() (deepmd.fit.dos.DOSFitting method), 362
get_numb_aparam() (deepmd.fit.ener.EnerFitting method), 362
get_numb_aparam() (deepmd.fit.EnerFitting method), 351
get_numb_aparam() (deepmd.model.dos.DOSModel method), 448
get_numb_aparam() (deepmd.model.DOSModel method), 436
get_numb_aparam() (deepmd.model.linear.LinearModel method), 457
get_numb_aparam() (deepmd.model.frozen.FrozenModel method), 454
get_numb_aparam() (deepmd.model.model.StandardModel method), 463
get_numb_aparam() (deepmd.model.multi.MultiModel method), 468
get_numb_aparam() (deepmd.model.MultiModel method), 443
get_numb_aparam() (deepmd.model.pairtab.PairTabModel method), 471
get_numb_aparam() (deepmd.model.pairwise_dprc.PairwiseDPRc method), 474
get_numb_aparam() (deepmd.utils.data.DeepmdData method), 512
get_numb_aparam() (deepmd.utils.DeepmdDataSystem method), 515
get_numb_aparam() (deepmd.utils.data_system.DeepmdDataSystem method), 589
get_numb_aparam() (deepmd_utils.utils.data.DeepmdData method), 593
get_numb_aparam() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 593
get_numb_aparam() (deepmd.infer.deep_dos.DeepDOS method), 409
get_numb_aparam() (deepmd.infer.DeepDOS method), 386
get_numb_aparam() (deepmd.infer.DeepPot method), 386
get_numb_aparam() (deepmd.infer.EnerFitting method), 366
get_numb_aparam() (deepmd.fit.ener.EnerFitting method), 351
get_numb_aparam() (deepmd.model.dos.DOSModel method), 448

892 Index
DeePMD-kit

get_sys_ntest() (deepmd.utils.DeepmdDataSystem method), 515

get_sys_ntest() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 593

get_sys_numb_batch() (deepmd.utils.data.DeepmdData method), 526

get_sys_numb_batch() (deepmd_utils.utils.data.DeepmdData method), 512

get_sys_numb_batch() (deepmd_utils.utils.data.DeepmdData method), 515

get_tensor_by_name() (in module deepmd.utils.graph), 533

get_tensor_by_name_from_graph() (in module deepmd.utils.graph), 534

get_tensor_by_type() (in module deepmd.utils.graph), 534

get_tensor_names() (deepmd.descriptor.Descriptor method), 232

get_tensor_names() (deepmd.descriptor.descriptor.Descriptor method), 284

get_tensor_names() (deepmd.descriptor.hybrid.DescriptorHybrid method), 237

get_tensor_names() (deepmd.descriptor.hybrid.DescriptorHybrid method), 289

get_tensor_names() (deepmd.descriptor.se.DescriptorSe method), 296

get_test() (deepmd.utils.data.DeepmdData method), 526

get_test() (deepmd.utils.data_system.DeepmdDataSystem method), 529

get_test() (deepmd.utils.DeepmdData method), 512

get_test() (deepmd.utils.DeepmdData method), 515

get_test() (deepmd_utils.utils.data.DeepmdData method), 589

get_test() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 593

get_tf_requirement() (in module backend.find_tensorflow), 217

get_tf_version() (in module backend.find_tensorflow), 218

get_training_jdata() (deepmd.nvnmd.utils.config.NvnmdConfig method), 499

get_two_side_type_embedding() (in module deepmd.utils.compress), 521

get_type_embedding() (in module deepmd.utils.compress), 521

get_type_embedding_net_nodes_from_graph_def() get_virtual_len() (in module deepmd.utils.graph), 534

get_type_embedding_net_variables_from_graph_def() get_weight() (in module deepmd.utils.graph), 534

get_type_embedding_weight() (in module deepmd.nvnmd.utils.weight), 506

get_type_map() (deepmd.infer.deep_dos.DeepDOS method), 398

get_type_map() (deepmd.infer.deep_pot.DeepPot method), 409

get_type_map() (deepmd.infer.deep_tensor.DeepTensor method), 412

get_type_map() (deepmd.infer.DeepDOS method), 375

get_type_map() (deepmd.infer.DeepPot method), 386

get_type_map() (deepmd.model.dos.DOSModel method), 448

get_type_map() (deepmd.model.DOSModel method), 436

get_type_map() (deepmd.model.ener.EnerModel method), 451

get_type_map() (deepmd.model.EnerModel method), 440

get_type_map() (deepmd.model.frozen.FrozenModel method), 454

get_type_map() (deepmd.model.linear.LinearModel method), 457

get_type_map() (deepmd.model.model.Model method), 463

get_type_map() (deepmd.model.multi.MultiModel method), 468

get_type_map() (deepmd.model.MultiModel method), 444

get_type_map() (deepmd.model.tensor.TensorModel method), 479

get_type_map() (deepmd.utils.data.DeepmdData method), 526

get_type_map() (deepmd.utils.data_system.DeepmdDataSystem method), 529

get_type_map() (deepmd.utils.DeepmdData method), 513

get_type_map() (deepmd.utils.DeepmdDataSystem method), 516

get_type_map() (deepmd.utils.data.DeepmdData method), 590

get_type_map() (deepmd.utils.data_system.DeepmdDataSystem method), 593

get_type_weight() (in module deepmd.nvnmd.utils.weight), 506

get_use_spin() (deepmd.utils.spin.Spin method), 548

get_variables_from_graph_def_as_numpy_array() (in module deepmd.utils.graph), 535

get_virtual_len() (deepmd.utils.spin.Spin method), 548

get_weight() (in module deepmd.utils.graph), 534
DeePMD-kit

keys_outp() (deepmd_utils.model_format.ModelOutputDef
method), 567
keys_outp() (deepmd_utils.model_format.output_def.ModelOutputDef
method), 577
keys_redu() (deepmd_utils.model_format.ModelOutputDef
method), 567
keys_redu() (deepmd_utils.model_format.output_def.ModelOutputDef
method), 577

layer_name:
model[standard]/fitting_net[ener]/layer_name
(Argument), 113
learning_rate (Argument)
learning_rate: 118
learning_rate/scale_by_worker (Argument)
scale_by_worker: 118
learning_rate/type (Argument)
type: 118
learning_rate:
learning_rate (Argument), 118
learning_rate_args() (in module
deepmd_utils.utils.argcheck), 582
learning_rate_dict (Argument)
learning_rate_dict: 119
learning_rate_dict:
learning_rate_dict (Argument), 119
learning_rate_dict_args() (in module
deepmd_utils.utils.argcheck), 582
learning_rate_exp() (in module
deepmd_utils.utils.argcheck), 582
learning_rate_variant_type_args() (in module
deepmd_utils.utils.argcheck), 582
limit_pref() (class in deepmd.utils)
LimitRateExp (class in deepmd.utils)
LimitRateExp (class in deepmd.utils.learning_rate), 535
limit_pref_acdf:
loss[dos]/limit_pref_acdf (Argument), 124
limit_prefados:
loss[dos]/limit_prefados (Argument), 124
limit_prefae:
loss[ener_spin]/limit_prefae (Argument), 122
limit prefados (Argument), 120
limit_prefcdf:
loss[dos]/limit_prefcdf (Argument), 124

loss[ener]/limit_pref_e (Argument), 119
loss[ener]/limit pref e (Argument), 121
loss[ener]/limit pref e (Argument), 119
limit_pref_fm:
loss[ener_spin]/limit pref fm (Argument), 122
loss[ener]/limit pref fm (Argument), 122
limit pref gf:
loss[ener]/limit pref gf (Argument), 121
limit pref pf:
loss[ener]/limit pref pf (Argument), 123
loss[ener]/limit pref pf (Argument), 120
limit pref v:
loss[ener_spin]/limit pref v (Argument), 122
loss[ener]/limit pref v (Argument), 120
linear ener model args() (in module
deepmd_utils.utils.argcheck), 582
LinearEnergyModel (class in deepmd.model.linear), 455
LinearModel (class in deepmd.model.linear), 456

list:
model[standard]/descriptor[hybrid]/list
(Argument), 102
list_to_doc() (in module deepmd.utils.argcheck), 519
list_to_doc() (in module
deepmd_utils.utils.argcheck), 582
load() (deepmd.nvnmd.utils.fio.FioBin method), 503
load() (deepmd.nvnmd.utils.fio.FioDic method), 503
load() (deepmd.nvnmd.utils.FioJson method), 504
load() (deepmd.nvnmd.utils.fio.FioNpy method), 504
load() (deepmd.nvnmd.utils.fio.FioNpy method), 504
load() (deepmd.nvnmd.utils.FioBin method), 495
load() (deepmd.nvnmd.utils.FioDic method), 495
load() (deepmd.nvnmd.utils.FioTxt method), 496
load_dp_model() (in module
deepmd_utils.model_format), 570
load_dp_model() (in module
deepmd_utils.model_format.network), 575
load_graph_def() (in module deepmd.utils.graph), 535
load_numpy() (deepmd.utils.path.DPH5Path method), 542
load_numpy() (deepmd.utils.path.DPOSPath method), 543
load_numpy() (deepmd.utils.path.DPPath method), 544
load_numpy() (deepmd.utils.utils.path.DPH5Path method), 596
load_numpy() (deepmd.utils.utils.path.DPOSPath method), 597
load_numpy() (deepmd.utils.utils.path.DPPath method), 598
load_prefix (deepmd.DeepEval attribute), 221
load_prefix (deepmd.infer.deep_dos.DeepDOS attribute), 398
load_prefix (deepmd.infer.deep_eval.DeepEval attribute), 400
load_prefix (deepmd.infer.deep_pot.DeepPot attribute), 409
load_prefix (deepmd.infer.deep_wfc.DeepWFC attribute), 413
load_prefix (deepmd.infer.DeepDOS attribute), 375
load_prefix (deepmd.infer.DeepEval attribute), 378
load_prefix (deepmd.infer.DeepPot attribute), 386
load_txt() (deepmd.utils.path.DPH5Path method), 542
load_txt() (deepmd.utils.path.DPOSPath method), 543
load_txt() (deepmd.utils.path.DPPath method), 544
load_txt() (deepmd.utils.utils.path.DPH5Path method), 596
load_txt() (deepmd.utils.utils.path.DPOSPath method), 597
load_txt() (deepmd.utils.utils.path.DPPath method), 598

loss (Argument)
loss; 119
Loss (class in deepmd.loss.loss), 432
loss/type (Argument)
type; 119
loss:
  loss (Argument), 119
loss_args() (in module deepmd.utils.utils.argcheck), 582
loss_dict (Argument)
loss_dict; 125
loss_dict:
  loss_dict (Argument), 125
loss_dict_args() (in module deepmd.utils.utils.argcheck), 582
loss_dos() (in module deepmd.utils.utils.argcheck), 582
loss_ener() (in module deepmd.utils.utils.argcheck), 582
loss_ener_spin() (in module deepmd.utils.utils.argcheck), 582
loss_tensor() (in module deepmd.utils.utils.argcheck), 582
loss_variant_type_args() (in module deepmd.utils.utils.argcheck), 582
loss[dos]/limit_pref_acdf (Argument)
  limit_pref_acdf; 124
loss[dos]/limit_pref_ados (Argument)
  limit_pref_ados; 124
loss[dos]/limit_pref_cdf (Argument)
  limit_pref_cdf; 124
loss[dos]/limit_pref_dos (Argument)
  limit_pref_dos; 123
loss[dos]/start_pref_acdf (Argument)
  start_pref_acdf; 124
loss[dos]/start_pref_ados (Argument)
  start_pref_ados; 124
loss[dos]/start_pref_cdf (Argument)
  start_pref_cdf; 123
loss[dos]/start_pref_dos (Argument)
  start_pref_dos; 123
loss[ener_spin]/enable_atom_ener_coeff (Argument)
  enable_atom_ener_coeff; 121
loss[ener_spin]/limit_pref_ae (Argument)
  limit pref_ae; 121
loss[ener_spin]/limit_pref_e (Argument)
  limit pref_e; 121
loss[ener_spin]/limit_pref_fm (Argument)
  limit pref_fm; 122
loss[ener_spin]/limit_pref_fr (Argument)
  limit pref_fr; 122
loss[ener_spin]/limit pref pf (Argument)
  limit pref pf; 123
loss[ener_spin]/limit_pref_v (Argument)
  limit pref v; 122
loss[ener_spin]/relative_f (Argument)
  relative f; 123
loss[ener_spin]/start pref ae (Argument)
  start pref ae; 122
loss[ener_spin]/start pref e (Argument)
  start pref e; 121
loss[ener_spin]/start pref fm (Argument)
  start pref fm; 122
loss[ener_spin]/start pref fr (Argument)
  start pref fr; 121
loss[ener_spin]/start pref pf (Argument)
  start pref pf; 123
loss[ener_spin]/start pref v (Argument)
  start pref v; 122
loss[ener]/enable_atom_ener_coeff (Argument)
  enable_atom_ener_coeff; 121
loss[ener]/limit_pref_ae (Argument)
  limit_pref_ae; 120
loss[ener]/limit_pref_e (Argument)
  limit_pref_e; 119
loss[ener]/limit_pref_f (Argument)
  limit_pref_f; 119
loss[ener]/limit_pref_gf (Argument)
  limit_pref_gf; 121
loss[ener]/limit_pref_pf (Argument)
  limit_pref_pf; 120
loss[ener]/limit_pref_v (Argument)
  limit_pref_v; 120
loss[ener]/numb_generalized_coord (Argument)
  numb_generalized_coord; 121
loss[ener]/relative_f (Argument)
  relative_f; 120
loss[ener]/start_pref_ae (Argument)
  start_pref_ae; 120
loss[ener]/start_pref_e (Argument)
  start pref_e; 119
loss[ener]/start_pref_f (Argument)
  start pref_f; 119
loss[ener]/start_pref_gf (Argument)
  start pref_gf; 121
loss[ener]/start_pref_pf (Argument)
  start pref_pf; 120
loss[ener]/start_pref_v (Argument)
  start pref_v; 120
loss[tensor]/pref (Argument)
  pref; 124
loss[tensor]/pref_atomic (Argument)
  pref_atomic; 125

M
main() (in module deepmd.entrypoints.main), 339
main() (in module deepmd_utils.main), 604
main_parser() (in module deepmd.entrypoints.main), 339
main_parser() (in module deepmd_utils.main), 604
make_data() (in module deepmd_utils.compress), 521
make_default_mesh() (in module deepmd_common), 558
make_default_mesh() (in module deepmd_utils_common), 603
make_embedding_network() (in module deepmd_utils_model_format), 570
make_embedding_network() (in module deepmd_utils_model_format_network), 575
make_fitting_network() (in module deepmd_utils_model_format), 570
make_fitting_network() (in module deepmd_utils_model_format_network), 575
make_index() (in module deepmd_utils_utils.argcheck), 582
make_link() (in module deepmd_utils_utils.argcheck), 582
make_model_dev() (in module deepmd_entrypoints), 333
make_model_dev() (in module deepmd_infer_model_dev), 417
make_multilayer_network() (in module deepmd_utils_model_format), 570
make_multilayer_network() (in module deepmd_utils_model_format_network), 575
make_natoms_vec() (deepmd.DeepEval method), 221
make_natoms_vec() (deepmd.infer.deep_eval.DeepEval method), 400
make_natoms_vec() (deepmd.infer.DeepEval method), 378
make_stat_input() (in module deepmd_model_model_stat), 465
make_stat_input() (in module deepmd_utils_model_model_stat), 594
map_aparam() (in module deepmd_env_op_module), 640
map_file:
  nvnmd_map_file (Argument), 130
map_flt_nvnmd() (in module deepmd_env_op_module), 641
map_nvnmd() (in module deepmd_nvnmd_utils), 496
map_nvnmd() (in module deepmd_nvnmd_utils_op), 505
MapAparam() (in module deepmd_env_op_module), 615
MapFltNvnmd() (in module deepmd_env_op_module), 615
mapping() (deepmd_nvnmd_entrypoints_map_table.MapTable method), 490
mapping() (deepmd_nvnmd_entrypoints_map_table.MapTable method), 485
mapping2() (deepmd_nvnmd_entrypoints_map_table.MapTable method), 490
map2() (deepmd_nvnmd_entrypoints_map_table.MapTable method), 485
map() (in module deepmd_nvnmd_entrypoints_map), 490
MapTable (class in deepmd_nvnmd_entrypoints), 483
MapTable (class in deepmd_nvnmd_entrypoints_map), 488
matmul2_qq() (in module deepmd_nvnmd_utils_network), 504
matmul3_qq() (in module deepmd_nvnmd_utils_network), 504
matmul_fitnet_nvnmd() (in module deepmd_nvnmd_utils_network), 504
DeePMD-kit

matmul_flt2fix_nvnm() (in module deepmd.env.op_module), 641
matmul_flt_nvnm() (in module deepmd.env.op_module), 641
MatmulFitnetNvnmd() (in module deepmd.env.op_module), 615
MatmulFlt2fixNvnmd() (in module deepmd.env.op_module), 616
MatmulFltNvnmd() (in module deepmd.env.op_module), 616
max_nnei:
   nvnmd/max_nnei (Argument), 130
merge_bin() (deepmd.nvnmd.utils.Encode method), 494
merge_bin() (deepmd.nvnmd.utils.encode.Encode method), 501
merge_input_stats()
   (deepmd.descriptor.DescrptHybrid method), 237
merge_input_stats()
   (deepmd.descriptor.DescrptSeA method), 248
merge_input_stats()
   (deepmd.descriptor.DescrptSeR method), 273
merge_input_stats()
   (deepmd.descriptor.DescrptSeT method), 277
merge_input_stats()
   (deepmd.descriptor.hybrid.DescrptHybrid method), 289
merge_input_stats()
   (deepmd.descriptor.se_a.DescrptSeA method), 302
merge_input_stats()
   (deepmd.descriptor.se_r.DescrptSeR method), 327
merge_input_stats()
   (deepmd.descriptor.se_t.DescrptSeT method), 331
merge_sys_stat() (in module deepmd.model.model_stat), 465
merge_sys_stat() (in module deepmd.utils.utils.model_stat), 594
mixed_precision:
   training/mixed_precision (Argument), 127
mixed_precision_args() (in module deepmd.utils.utils.argcheck), 582
mkdir() (deepmd.nvnmd.utils.fio.Fio method), 502
MOASPNDIM (C macro), 849
model (Argument)
   model; 89
Model (class in deepmd.model.model), 458
model/compress (Argument)
   compress; 92
model/data_bias_nsample (Argument)
   data_bias_nsample; 89
model/data_stat_nbatch (Argument)
   data_stat_nbatch; 89
model/data_stat_protect (Argument)
   data_stat_protect; 89
model/modifier (Argument)
   modifier; 91
model/modifier/type (Argument)
   type; 91
model/modifier[dipole_charge]/ewald_beta
   (Argument)
   ewald_beta; 91
model/modifier[dipole_charge]/ewald_h
   (Argument)
   ewald_h; 92
model/modifier[dipole_charge]/model_charge_map
   (Argument)
   model_charge_map; 91
model/modifier[dipole_charge]/model_name
   (Argument)
   model_name; 91
model/modifier[dipole_charge]/sys_charge_map
   (Argument)
   sys_charge_map; 91
model/smin_alpha (Argument)
   smin_alpha; 89
model/spin (Argument)
   spin; 92
model/spin/spin_norm (Argument)
   spin_norm; 92
model/spin/use_spin (Argument)
   use_spin; 92
model/spin/virtual_len (Argument)
   virtual_len; 92
model/srtab_add_bias (Argument)
   srtab_add_bias; 90
model/sw_rmax (Argument)
   sw_rmax; 90
model/sw_rmin (Argument)
   sw_rmin; 90
model/type (Argument)
   type; 92
model/type_embedding (Argument)
   type_embedding; 90
model/type_embedding/activation_function
   (Argument)
   activation_function; 90
model/type_embedding/neuron (Argument)
   neuron; 90
model/type_embedding/precision (Argument)
   precision; 90
DeePMD-kit

model/type_embedding/resnet_dt
- resnet_dt: 90

model/type_embedding/seed
- seed: 91

model/type_embedding/trainable
- trainable: 91

model/type_map
- type_map: 89

model/use_srtab
- use_srtab: 89

model
- model: model() (Argument), 89

model_args() (in module deepmd_utils.utils.argcheck), 582

model_charge_map
- model_charge_map: model/modifier[dipole_charge]/model_charge_map (Argument), 91

model_check_output() (in module deepmd_utils.model_format), 570
- model check_output() (in module deepmd_utils.model_format.output_def), 578

model_compression() (in module deepmd_utils.utils.argcheck), 582
- model_compression_type_args() (in module deepmd_utils.utils.argcheck), 582

model_file
- model_file: model[frozen]/model_file (Argument), 117

model_name
- model_name: model/modifier[dipole_charge]/model_name (Argument), 91

model_type
- model_type: deepmd.DeepEvalproperty, 221
- model_type: deepmd.infer.deep_eval.DeepEval property, 400
- model_type: deepmd.infer.DeepEval property, 379
- model_type: deepmd.model.dos.DOSModel attribute, 448
- model_type: deepmd.model.DOSModel attribute, 436
- model_type: deepmd.model.ener.EnerModel attribute, 452
- model_type: deepmd.model.linear.LinearEnergyModel attribute, 456
- model_type: deepmd.model.multi.MultiModel attribute, 468
- model_type: deepmd.model.pairtab.PairTabModel attribute, 471
- model_type: deepmd.model.pairwise_dprc.PairwiseDPRc attribute, 474

model_version
- model_version: deepmd.DeepEvalproperty, 221
- model_version: deepmd.infer.deep_eval.DeepEval property, 401
- model_version: deepmd.infer.DeepEval property, 379

ModelOutputDef
- ModelOutputDef (class in deepmd_utils.model_format), 566
- ModelOutputDef (class in deepmd_utils.model_format.output_def), 576

models
- models: model[linear_ener]/models (Argument), 118
- models: model[frozen]/model_file (Argument)
- models: model[linear_ener]/models (Argument)
- models: model_file (Argument)
- models: model[linear_ener]/weights (Argument)
- models: descriptor (Argument)
- models: fitting_net_dict (Argument)
- models: rcut (Argument)
- models: sel (Argument)
- models: tab_file (Argument)
- models: qm_model (Argument)
- models: qmmm_model (Argument)
- models: descriptor (Argument)
- models: descriptor[type] (Argument)
- models: descriptor[hybrid]/list (Argument)
- models: descriptor[loc_frame]/axis_rule (Argument)
- models: descriptor[loc_frame]/rcut (Argument)
- models: descriptor[loc_frame]/sel_a (Argument)
- models: descriptor[loc_frame]/sel_r (Argument)
- models: descriptor[se_a_ebd_v2]/activation_function (Argument)

902 Index
model[standard]/descriptor[se_a_ebd_v2]/axis_neuron
(Argument) axis_neuron: 108
model[standard]/descriptor[se_a_ebd_v2]/exclude_types
(Argument) exclude_types: 109
model[standard]/descriptor[se_a_ebd_v2]/neuron
(Argument) neuron: 108
model[standard]/descriptor[se_a_ebd_v2]/precision
(Argument) precision: 108
model[standard]/descriptor[se_a_ebd_v2]/rcut
(Argument) rcut: 107
model[standard]/descriptor[se_a_ebd_v2]/rcut_smth
(Argument) rcut_smth: 107
model[standard]/descriptor[se_a_ebd_v2]/resnet_dt
(Argument) resnet_dt: 108
model[standard]/descriptor[se_a_ebd_v2]/seed
(Argument) seed: 109
model[standard]/descriptor[se_a_ebd_v2]/sel
(Argument) sel: 107
model[standard]/descriptor[se_a_ebd_v2]/set_davg_zero
(Argument) set_davg_zero: 109
model[standard]/descriptor[se_a_ebd_v2]/trainable
(Argument) trainable: 109
model[standard]/descriptor[se_a_mask]/activation_function
(Argument) activation_function: 98
model[standard]/descriptor[se_a_mask]/axis_neuron
(Argument) axis_neuron: 98
model[standard]/descriptor[se_a_mask]/exclude_types
(Argument) exclude_types: 99
model[standard]/descriptor[se_a_mask]/neuron
(Argument) neuron: 98
model[standard]/descriptor[se_a_mask]/numb_aparam
(Argument) numb_aparam: 100
model[standard]/descriptor[se_a_mask]/precision
(Argument) precision: 99
model[standard]/descriptor[se_a_mask]/resnet_dt
(Argument) resnet_dt: 98
model[standard]/descriptor[se_a_mask]/set_davg_zero
(Argument) set_davg_zero: 99
model[standard]/descriptor[se_a_mask]/trainable
(Argument) trainable: 99
model[standard]/descriptor[se_a_tpe]/activation_function
(Argument) activation_function: 98
model[standard]/descriptor[se_a_tpe]/axis_neuron
(Argument) axis_neuron: 99
model[standard]/descriptor[se_a_tpe]/exclude_types
(Argument) exclude_types: 97
model[standard]/descriptor[se_a_tpe]/neuron
(Argument) neuron: 99
model[standard]/descriptor[se_a_tpe]/type_nchanl
(Argument) type_nchanl: 99
model[standard]/descriptor[se_a_tpe]/set_davg_zero
(Argument) set_davg_zero: 99
model[standard]/descriptor[se_a_tpe]/trainable
(Argument) trainable: 99
DeePMD-kit

model[standard]/descriptor[se_attn]/type_nlayer (Argument)
   type_nlayer: 99
model[standard]/descriptor[se_attn]/type_one_side (Argument)
   type_one_side: 98
model[standard]/descriptor[se_atten_v2]/activation_function (Argument)
   activation_function: 105
model[standard]/descriptor[se_atten_v2]/attn (Argument)
   attn: 106
model[standard]/descriptor[se_atten_v2]/attn_dotr (Argument)
   attn_dotr: 107
model[standard]/descriptor[se_atten_v2]/attn_layer (Argument)
   attn_layer: 104
model[standard]/descriptor[se_atten_v2]/attn_mask (Argument)
   attn_mask: 107
model[standard]/descriptor[se_atten_v2]/axis_neuron (Argument)
   axis_neuron: 105
model[standard]/descriptor[se_atten_v2]/exclude_types (Argument)
   exclude_types: 106
model[standard]/descriptor[se_atten_v2]/neuron (Argument)
   neuron: 105
model[standard]/descriptor[se_atten_v2]/precision (Argument)
   precision: 103
model[standard]/descriptor[se_atten_v2]/rcut (Argument)
   rcut: 102
model[standard]/descriptor[se_atten_v2]/rcut_smth (Argument)
   rcut_smth: 105
model[standard]/descriptor[se_atten_v2]/resnet_dt (Argument)
   resnet_dt: 106
model[standard]/descriptor[se_atten_v2]/seed (Argument)
   seed: 103
model[standard]/descriptor[se_atten_v2]/sel (Argument)
   sel: 102
model[standard]/descriptor[se_atten_v2]/set_davg_zero (Argument)
   set_davg_zero: 104
model[standard]/descriptor[se_atten_v2]/smooth_type_embedding (Argument)
   smooth_type_embedding: 104
model[standard]/descriptor[se_atten_v2]/stripped_type_embedding (Argument)
   stripped_type_embedding: 104
model[standard]/descriptor[se_attn]/trainable (Argument)
  trainable: 103
model[standard]/descriptor[se_attn]/type_one_side (Argument)
  type_one_side: 103
model[standard]/descriptor[se_e2_a]/activation_function (Argument)
  activation_function: 95
model[standard]/descriptor[se_e2_a]/axis_neuron (Argument)
  axis_neuron: 95
model[standard]/descriptor[se_e2_a]/exclude_types (Argument)
  exclude_types: 96
model[standard]/descriptor[se_e2_a]/neuron (Argument)
  neuron: 94
model[standard]/descriptor[se_e2_a]/precision (Argument)
  precision: 95
model[standard]/descriptor[se_e2_a]/rcut (Argument)
  rcut: 94
model[standard]/descriptor[se_e2_a]/rcut_smth (Argument)
  rcut_smth: 94
model[standard]/descriptor[se_e2_a]/resnet_dt (Argument)
  resnet_dt: 95
model[standard]/descriptor[se_e2_a]/seed (Argument)
  seed: 95
model[standard]/descriptor[se_e2_a]/sel (Argument)
  sel: 94
model[standard]/descriptor[se_e2_a]/set_davg_zero (Argument)
  set_davg_zero: 96
model[standard]/descriptor[se_e2_r]/activation_function (Argument)
  activation_function: 100
model[standard]/descriptor[se_e2_r]/axis_neuron (Argument)
  axis_neuron: 100
model[standard]/descriptor[se_e2_r]/exclude_types (Argument)
  exclude_types: 101
model[standard]/descriptor[se_e2_r]/neuron (Argument)
  neuron: 100
model[standard]/descriptor[se_e2_r]/precision (Argument)
  precision: 101
model[standard]/descriptor[se_e2_r]/rcut (Argument)
  rcut: 100
model[standard]/descriptor[se_e2_r]/rcut_smth (Argument)
  rcut_smth: 100
model[standard]/descriptor[se_e2_r]/resnet_dt (Argument)
  resnet_dt: 101
model[standard]/descriptor[se_e2_r]/seed (Argument)
  seed: 101
model[standard]/descriptor[se_e2_r]/sel (Argument)
  sel: 100
model[standard]/descriptor[se_e2_r]/set_davg_zero (Argument)
  set_davg_zero: 101
model[standard]/descriptor[se_e3]/activation_function (Argument)
  activation_function: 97
model[standard]/descriptor[se_e3]/axis_neuron (Argument)
  axis_neuron: 96
model[standard]/descriptor[se_e3]/exclude_types (Argument)
  exclude_types: 97
model[standard]/descriptor[se_e3]/neuron (Argument)
  neuron: 96
model[standard]/descriptor[se_e3]/precision (Argument)
  precision: 97
model[standard]/descriptor[se_e3]/rcut (Argument)
  rcut: 96
model[standard]/descriptor[se_e3]/rcut_smth (Argument)
  rcut_smth: 96
model[standard]/descriptor[se_e3]/resnet_dt (Argument)
  resnet_dt: 97
model[standard]/descriptor[se_e3]/seed (Argument)
  seed: 97
model[standard]/descriptor[se_e3]/sel (Argument)
  sel: 96
model[standard]/descriptor[se_e3]/set_davg_zero (Argument)
  set_davg_zero: 97
DeePMD-kit

model[standard]/descriptor[se_e3]/trainable (Argument)
  trainable.: 97
model[standard]/fitting_net (Argument)
  fitting_net.: 111
model[standard]/fitting_net/type (Argument)
  type.: 111
model[standard]/fitting_net[dipole]/activation_function (Argument)
  activation_function.: 116
model[standard]/fitting_net[dipole]/neuron (Argument)
  neuron.: 116
model[standard]/fitting_net[dipole]/precision (Argument)
  precision.: 116
model[standard]/fitting_net[dipole]/resnet_dt (Argument)
  resnet_dt.: 116
model[standard]/fitting_net[dipole]/seed (Argument)
  seed.: 116
model[standard]/fitting_net[dipole]/sel_type (Argument)
  sel_type.: 116
model[standard]/fitting_net[dos]/activation_function (Argument)
  activation_function.: 113
model[standard]/fitting_net[dos]/atom_ener (Argument)
  atom_ener.: 113
model[standard]/fitting_net[dos]/layer_name (Argument)
  layer_name.: 113
model[standard]/fitting_net[dos]/neuron (Argument)
  neuron.: 112
model[standard]/fitting_net[dos]/numb_aparam (Argument)
  numb_aparam.: 111
model[standard]/fitting_net[dos]/numb_fparam (Argument)
  numb_fparam.: 111
model[standard]/fitting_net[dos]/precision (Argument)
  precision.: 114
model[standard]/fitting_net[dos]/rcond (Argument)
  rcond.: 112
model[standard]/fitting_net[dos]/resnet_dt (Argument)
  resnet_dt.: 112
model[standard]/fitting_net[dos]/seed (Argument)
  seed.: 112
model[standard]/fitting_net[dos]/use_aparam_as_mask (Argument)
  use_aparam_as_mask.: 113
model[standard]/fitting_net[polar]/activation_function (Argument)
  activation_function.: 115
model[standard]/fitting_net[polar]/fit_diag (Argument)
  fit_diag.: 115
model[standard]/fitting_net[polar]/neuron (Argument)
  neuron.: 114
model[standard]/fitting_net[polar]/precision (Argument)
  precision.: 115
model[standard]/fitting_net[polar]/resnet_dt (Argument)
  resnet_dt.: 115
model[standard]/fitting_net[polar]/scale (Argument)
scale: 115
call_model[standard]/fitting_net[polar]/seed
  (Argument)
seed: 115

call_model[standard]/fitting_net[polar]/sel_type
  (Argument)
sel_type: 115

call_model[standard]/fitting_net[polar]/shift_diag
  (Argument)
shift_diag: 115

modifier:
  model/modifier (Argument), 91
modifier_dipole_charge() (in module
deepmd_utils.utils.argcheck), 583
modifier_variant_type_args() (in module
deepmd_utils.utils.argcheck), 583
modify_data() (deepmd.DipoleChargeModifier
  method), 224
modify_data() (deepmd.infer.data_modifier.DipoleChargeModifier
  method), 393
modify_data() (deepmd.infer.DipoleChargeModifier
  method), 390

module
  backend, 217
  backend.dynamic_metadata, 217
  backend.find_tensorflow, 217
  backend.read_env, 218
deeplm, 219
deeplm.calculator, 552
deeplm.cluster, 225
deeplm.cluster.local, 225
deeplm.cluster.slurm, 226
deeplm.common, 554
deeplm.descriptor, 226
deeplm.descriptor.descriptor, 278
deeplm.descriptor.hybrid, 285
deeplm.descriptor.loc_frame, 291
deeplm.descriptor.se, 295
deeplm.descriptor.se_a, 297
deeplm.descriptor.se_a_ebd, 303
deeplm.descriptor.se_a_ebd_v2, 305
deeplm.descriptor.se_a_ef, 306
deeplm.descriptor.se_a_mask, 312
deeplm.descriptor.se_atten, 316
deeplm.descriptor.se_atten_v2, 321
deeplm.descriptor.se_r, 323
deeplm.descriptor.se_t, 328
deeplm.entrypoints, 332
deeplm.entrypoints.compress, 337
deeplm.entrypoints.convert, 338
deeplm.entrypoints.doc, 338
deeplm.entrypoints.freeze, 338
deeplm.entrypoints.gui, 338
deeplm.entrypoints.ipi, 339
deeplm.entrypoints.main, 339
deeplm.entrypoints.neighbor_stat, 340
deeplm.entrypoints.test, 340
deeplm.entrypoints.train, 341
deeplm.entrypoints.transfer, 342
deeplm.env, 559
deeplm.env.op_grads_module, 659
deeplm.env.op_module, 607
deeplm.fit, 343
deeplm.fit.dipole, 357
deeplm.fit.dos, 360
deeplm.fit.ener, 363
deeplm.fit.hess, 366
deeplm.fit.polar, 368
deeplm.infer, 372
deeplm.infer.data_modifier, 392
deeplm.infer.deep_dipole, 394
deeplm.infer.deep_dos, 395
deeplm.infer.deep_eval, 398
deeplm.infer.deep_polar, 402
deeplm.infer.deep_pot, 405
deeplm.infer.deep_tensor, 409
deeplm.infer.deep_wfc, 412
deeplm.infer.ewald_recp, 413
deeplm.infer.ml_model_dev, 414
deeplm.lmp, 559
deeplm.loggers, 418
deeplm.loggers.loggers, 419
deeplm.loss, 420
deeplm.loss.dos, 426
deeplm.loss.ener, 427
deeplm.loss.loss, 432
deeplm.loss.tensor, 433
deeplm.model, 434
deeplm.model.dos, 446
deeplm.model.ener, 449
deeplm.model.frozen, 452
deeplm.model.linear, 455
deeplm.model.model, 458
deeplm.model.model_stat, 465
deeplm.model.multi, 466
deeplm.model.pairtab, 469
deeplm.model.pairwise_dprc, 472
deeplm.model.tensor, 475
deeplm.nvnmd, 480
deeplm.nvnmd.data, 480
deeplm.nvnmd.data.data, 481
deeplm.nvnmd.descriptor, 481
deeplm.nvnmd.descriptor.se_a, 482
deeplm.nvnmd.descriptor.se_atten, 482
deeplm.nvnmd.entrypoints, 483
deeplm.nvnmd.entrypoints.freeze, 487
deeplm.nvnmd.entrypoints.mapt, 488
deeplm.nvnmd.entrypoints.train, 490

nborAssert (C++ function), 843, 844
nborErrcheck (C macro), 850
neighbor_stat() (in module deepmd.entrypoints), 334
neighbor_stat() (in module deepmd.entrypoints.neighbor_stat), 340
neighbor_stat() (in module deepmd.env.op_module), 642
NeighborStat (class in deepmd.utils.neighbor_stat), 537
NeighborStat() (in module deepmd.env.op_module), 616
net_size: nvnmd/net_size (Argument), 130
NETWORK_TYPE_MAP (deepmd_utils.model.format.NetworkCollection attribute), 575
NETWORK_TYPE_MAP (deepmd_utils.model_format.NetworkCollection attribute), 569
NetworkCollection (class in deepmd_utils.model_format), 568
NetworkCollection (class in deepmd_utils.model_format.network), 574
neuron:
  model/type_embedding/neuron (Argument), 90
  model[standard]/descriptor[se_a_ebd_v2]/numb_aparam
    (Argument), 108
  model[standard]/descriptor[se_a_mask]/neuron
    (Argument), 109
  model[standard]/descriptor[se_a_tpe]/neuron
    (Argument), 98
  model[standard]/descriptor[se_atten_v2]/neuron
    (Argument), 105
  model[standard]/descriptor[se_atten]/neuron
    (Argument), 102
  model[standard]/descriptor[se_e2_a]/numb_dos
    (Argument), 94
  model[standard]/descriptor[se_e2_r]/neuron
    (Argument), 100
  model[standard]/descriptor[se_e3]/neuron
    (Argument), 96
  model[standard]/fitting_net[dipole]/neuron
    (Argument), 116
  model[standard]/fitting_net[dos]/neuron
    (Argument), 113
  model[standard]/fitting_net[ener]/neuron
    (Argument), 112
  model[standard]/fitting_net[polar]/neuron
    (Argument), 114
modelist
  (deepmd.train.run_options.RunOptions attribute), 508
modelist (deepmd.nvnmd.run_options.RunOptions attribute), 508
norm_expo() (deepmd.nvnmd.utils.Encode method), 494
norm_expo() (deepmd.nvnmd.utils.Encode method), 501
normalize() (in module deepmd.utils.argcheck), 519
normalize() (in module deepmd_utils.utils.argcheck), 583
normalize() (in module deepmd_utils.utils.argcheck), 583
normalize_data_dict() (in module deepmd_utils.utils.argcheck), 583
normalize_fitting_net_dict() (in module deepmd_utils.utils.argcheck), 583
normalize_fitting_weight() (in module deepmd_utils.utils.argcheck), 583
normalize_learning_rate_dict() (in module deepmd_utils.utils.argcheck), 583
normalize_learning_rate_dict_with_single_learning_rate() (in module deepmd_utils.utils.argcheck), 583
normalize_loss_dict() (in module deepmd_utils.utils.argcheck), 583
normalize_multi_task() (in module deepmd_utils.utils.argcheck), 583
normalized_input() (in module deepmd.nvnmd.run_options.train), 490
normalized_input_qnn() (in module deepmd.nvnmd.run_options.train), 490
numb_aparam:
  model[standard]/descriptor[se_a_tpe]/numb_aparam
    (Argument), 100
  model[standard]/fitting_net[dos]/numb_aparam
    (Argument), 113
numb_btch:
  training/validation_data/numb_btch
    (Argument), 127
numb_dos:
  model[standard]/fitting_net[dos]/numb_dos
    (Argument), 114
numb_fparam:
  model[standard]/fitting_net[dos]/numb_fparam
    (Argument), 113
numb_generalized_coord:
  loss[ener]/numb_generalized_coord
    (Argument), 121
numb_steps:
  training/numb_steps (Argument), 128
nvnmd (Argument)
  nvnmd:, 129
  nvnmd/config_file (Argument)
  config_file:, 130
  nvnmd/enable (Argument)
  enable:, 130
DeePMD-kit

nvnmd/map_file (Argument)
  map_file: 130
nvnmd/max_nnei (Argument)
  max_nnei: 130
nvnmd/net_size (Argument)
  net_size: 130
nvnmd/quantize_descriptor (Argument)
  quantize_descriptor: 131
nvnmd/quantize_fitting_net (Argument)
  quantize_fitting_net: 131
nvnmd/restore_descriptor (Argument)
  restore_descriptor: 130
nvnmd/restore_fitting_net (Argument)
  restore_fitting_net: 130
nvnmd/version (Argument)
  version: 130
nvnmd/weight_file (Argument)
  weight_file: 130
nvnmd:
  nvnmd (Argument), 129
  nvnmd_args() (in module deepmd.nvnmd.utils), 497
  nvnmd_args() (in module deepmd.nvnmd.utils.argcheck), 497
  nvnmd_args() (in module deepmd.nvnmd.utils.argcheck.nvnmd),
  583
NvnmdConfig (class in deepmd.nvnmd.utils.config),
  497

O
omp_get_num_threads (C++ function), 844
omp_get_thread_num (C++ function), 844
one_layer() (in module deepmd.nvnmd.utils), 497
one_layer() (in module deepmd.nvnmd.utils.network), 505
one_layer() (in module deepmd.nvnmd.utils.tensor), 539
one_layer_nvnmd() (in module deepmd.nvnmd.fit.ener), 492
one_layer_rand_seed_shift() (in module deepmd.nvnmd.utils.tensor), 539
one_layer_t() (in module deepmd.nvnmd.utils.tensor), 505
one_layer_wb() (in module deepmd.nvnmd.utils.tensor), 505
OutOfMemoryError, 529, 593
output_prec:
  training/mixed_precision/output_prec (Argument), 127
OutputVariableDef (class in deepmd.utils.model_format), 569
OutputVariableDef (class in deepmd.utils.model_format.output_def),
  577
P
pair_tab() (in module deepmd.env.op_module), 642
PairTab (class in deepmd.utils.pair_tab), 539
PairTab (class in deepmd.utils.pair_tab, 594
PairTab() (in module deepmd.env.op_module), 617
pairtab_model_args() (in module deepmd.utils.utils.argcheck), 583
PairTabModel (class in deepmd.model.pairtab), 469
pairwise_dprc() (in module deepmd.utils.utils.argcheck), 583
PairwiseDPRc (class in deepmd.model.pairwise_dprc), 472
parallel_prod_force_se_a() (in module deepmd.env.op_module), 643
ParallelOp (class in deepmd.utils.parallel_op), 540
ParallelProdForceSeA() (in module deepmd.env.op_module), 617
parse_args() (in module deepmd.entrypoints.main), 339
parse_args() (in module deepmd.utils.main), 604
pass_tensors_from_frz_model() (deepmd.descriptor.Descriptor method),
  232
pass_tensors_from_frz_model() (deepmd.descriptor.descriptor.Descriptor method), 284
pass_tensors_from_frz_model() (deepmd.descriptor.DescrptHybrid method), 238
pass_tensors_from_frz_model() (deepmd.descriptor.hybrid.DescrptHybrid method), 290
pass_tensors_from_frz_model() (deepmd.descriptor.se.DescrptSe method), 296
plot_lines() (deepmd.nvnmd.entrypoints.mapt.MapTable method), 490
plot_lines() (deepmd.nvnmd.entrypoints.MapTable method), 485
Plugin (class in deepmd.utils), 517
Plugin (class in deepmd.utils.plugin), 545
Plugin (class in deepmd.utils.plugin, 598
PluginVariant (class in deepmd.utils), 518
PluginVariant (class in deepmd.utils.plugin), 546
PluginVariant (class in deepmd.utils.plugin), 599
PolarFittingSeA (class in deepmd.fit), 355
PolarFittingSeA (class in deepmd.fit.polar), 370
PolarModel (class in deepmd.model), 444
PolarModel (class in deepmd.model.tensor), 476
precision (deepmd.descriptor.se.DescrptSe property), 296
precision (deepmd.fit.Fitting property), 352
DeePMD-kit

precision: (deepmd.fit.fitting.Fitting property), 367
  model/type_embedding/precision
    (Argument), 90
  model[standard]/descriptor[se_ebd_v2]/precision
    (Argument), 108
  model[standard]/descriptor[se_a_mask]/precision
    (Argument), 110
  model[standard]/descriptor[se_a_tpe]/precision
    (Argument), 99
  model[standard]/descriptor[se_atten_v2]/precision
    (Argument), 106
  model[standard]/descriptor[se_atten]/precision
    (Argument), 103
  model[standard]/descriptor[se_e2_a]/precision
    (Argument), 95
  model[standard]/descriptor[se_e2_r]/precision
    (Argument), 101
  model[standard]/descriptor[se_e3]/precision
    (Argument), 97
  model[standard]/fitting_net[dipole]/precision
    (Argument), 116
  model[standard]/fitting_net[dos]/precision
    (Argument), 114
  model[standard]/fitting_net[ener]/precision
    (Argument), 112
  model[standard]/fitting_net[polar]/precision
    (Argument), 115
  pref:
    loss[tensor]/pref (Argument), 124
  pref_atomic:
    loss[tensor]/pref_atomic (Argument), 125
  print_header() (deepmd.loss.ener.EnerSpinLoss method), 429
  print_header() (deepmd.loss.ener.EnerSpinLoss method), 433
  print_on_training() (deepmd.train.trainer.DPTrainer static method), 509
  print_resource_summary()
    (deepmd.train.run_options.RunOptions method), 508
  print_summary() (deepmd.utils.data_system.DeepmdDataSystem method), 529
  print_summary() (deepmd.utils.data_system.DeepmdDataSystem method), 516
  print_summary() (deepmd.utils.data_system.DeepmdDataSystem method), 516
256 prod_force_virial() (deepmd.descriptor.DescrptSeAMask method), 261
prod_force_virial() (deepmd.descriptor.DescrptSeR method), 273
prod_force_virial() (deepmd.descriptor.DescrptSeT method), 278
prod_force_virial() (deepmd.descriptor.hybrid.DescrptHybrid method), 290
prod_force_virial() (deepmd.descriptor.loc_frame.DescrptLocFrame method), 294
prod_force_virial() (deepmd.descriptor.se_a.DescrptSeA method), 302
prod_force_virial() (deepmd.descriptor.se_a_ef.DescrptSeAEf method), 310
prod_force_virial() (deepmd.descriptor.se_a_mask.DescrptSeAMask method), 315
prod_force_virial() (deepmd.descriptor.se_r.DescrptSeR method), 327
prod_force_virial() (deepmd.descriptor.se_t.DescrptSeT method), 332
prod_virial() (in module deepmd.env.op_module), 649
prod_virial_grad() (in module deepmd.env.op_grads_module), 665
prod_virial_norot() (in module deepmd.env.op_module), 649
prod_virial_se_a() (in module deepmd.env.op_module), 650
prod_virial_se_a_grad() (in module deepmd.env.op_grads_module), 665
prod_virial_se_r() (in module deepmd.env.op_module), 650
prod_virial_se_r_grad() (in module deepmd.env.op_grads_module), 665
ProdEnvMatA() (in module deepmd.env.op_module), 618
ProdEnvMatAMix() (in module deepmd.env.op_module), 619
ProdEnvMatAMixNvmdQuantize() (in module deepmd.env.op_module), 619
ProdEnvMatANvmdQuantize() (in module deepmd.env.op_module), 620
ProdEnvMatR() (in module deepmd.env.op_module), 621
ProdForce() (in module deepmd.env.op_module), 621
ProdForceGrad() (in module deepmd.env.op_grads_module), 660
ProdForceNorot() (in module deepmd.env.op_module), 621
ProdForceSeA() (in module deepmd.env.op_module), 622
ProdForceSeAGPUExecuteFunctor (C++ struct), 799
ProdForceSeAGPUExecuteFunctor::operator() (C++ function), 800
ProdForceSeAGrad() (in module deepmd.env.op_grads_module), 660
ProdForceSeAMask() (in module deepmd.env.op_module), 622
ProdForceSeAMaskGrad() (in module deepmd.env.op_grads_module), 660
ProdForceSeR() (in module deepmd.env.op_module), 622
ProdForceSeRGPUExecuteFunctor (C++ struct), 800
ProdForceSeRGPUExecuteFunctor::operator() (C++ function), 800
ProdForceSeRGrad() (in module deepmd.env.op_grads_module), 660
ProdVirial() (in module deepmd.env.op_module), 623
ProdVirialGrad() (in module deepmd.env.op_grads_module), 661
ProdVirialNorot() (in module deepmd.env.op_module), 623
ProdVirialSeA() (in module deepmd.env.op_module), 624
ProdVirialSeAGPUExecuteFunctor (C++ struct), 800
ProdVirialSeAGPUExecuteFunctor::operator() (C++ function), 800
ProdVirialSeAGrad() (in module deepmd.env.op_grads_module), 662
ProdVirialSeR() (in module deepmd.env.op_module), 624
ProdVirialSeRGPUExecuteFunctor (C++ struct), 801
ProdVirialSeRGPUExecuteFunctor::operator() (C++ function), 801
ProdVirialSeRGrad() (in module deepmd.env.op_grads_module), 662
profiling: training/profiling (Argument), 129
profiling_file: training/profiling_file (Argument), 129
Q qc() (deepmd.nvnmd.utils.Encode method), 494
DeePMD-kit

qc() (deepmd.nvnmn.utils.encode.Encode method), 501
qf() (deepmd.nvnmn.utils.Encode method), 495
qf() (deepmd.nvnmn.utils.encode.Encode method), 501
qf() (in module deepmd.nvnmn.utils.network), 505
qm_model:
  model[pairwise_dprc]/qm_model (Argument), 118
qmmm_model:
  model[pairwise_dprc]/qmmm_model (Argument), 118
qr() (deepmd.nvnmn.utils.Encode method), 495
qr() (in module deepmd.nvnmn.utils.network), 505
quantize_descriptor:
  nvnmn/quantize_descriptor (Argument), 131
quantize_fitting_net:
  nvnmn/quantize_fitting_net (Argument), 131
quantize_nvnm() (in module deepmd.env.op_module), 650
QuantizeNvnmn() (in module deepmd.env.op_module), 624

R
r2s() (in module deepmd.nvnmn.utils.op), 505
random() (in module deepmd.utils.random), 546
random() (in module deepmd_utils.utils.random), 600
RawTextArgumentDefaultsHelpFormatter (class in deepmd_utils.main), 604
rcond:
  model[standard]/fitting_net[dos]/rcond (Argument), 114
  model[standard]/fitting_net[ener]/rcond (Argument), 112
rcut:
  model[pairtab]/rcut (Argument), 117
  model[standard]/descriptor[loc_frame]/rcut (Argument), 93
  model[standard]/descriptor[se_a_ebd_v2]/rcut (Argument), 107
  model[standard]/descriptor[se_a_tpe]/rcut (Argument), 98
  model[standard]/descriptor[se_atten_v2]/rcut (Argument), 105
  model[standard]/descriptor[se_atten]/rcut (Argument), 102
  model[standard]/descriptor[se_e2_a]/rcut (Argument), 94
  model[standard]/descriptor[se_e2_r]/rcut (Argument), 100
  model[standard]/descriptor[se_e3]/rcut (Argument), 96
rcut_smth:
  model[standard]/descriptor[se_a_ebd_v2]/rcut_smth (Argument), 107
  model[standard]/descriptor[se_a_tpe]/rcut_smth (Argument), 107
  model[standard]/descriptor[se_atten_v2]/rcut_smth (Argument), 105
  model[standard]/descriptor[se_atten]/rcut_smth (Argument), 102
  model[standard]/descriptor[se_e2_a]/rcut_smth (Argument), 94
  model[standard]/descriptor[se_e2_r]/rcut_smth (Argument), 100
  model[standard]/descriptor[se_e3]/rcut_smth (Argument), 96
reduce() (deepmd.utils.data.DeepmdData method), 526
reduce() (deepmd.utils.data_system.DeepmdDataSystem method), 529
reduce() (deepmd.utils.DeepmdData method), 513
reduce() (deepmd.utils.DeepmdDataSystem method), 516
reduce() (deepmd_utils.utils.data.DeepmdData method), 590
reduce() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 593
register() (deepmd.descriptor_Descriptor static method), 233
register() (deepmd.descriptor.descriptor.Descriptor static method), 285
register() (deepmd.fit.Fitting static method), 352
register() (deepmd.fit.fitting.Fitting static method), 367
register() (deepmd.utils.Plugin method), 518
register() (deepmd.utils.plugin.Plugin method), 545
register() (deepmd_utils.utils.argcheck_ArgsPlugin method), 581
register() (deepmd_utils.utils.plugin.Plugin method), 599
reinit() (deepmd.utils.pair_tab.PairTab method), 539
reinit() (deepmd.utils.PairTab method), 517
reinit() (deepmd_utils.utils.pair_tab.PairTab method), 594
relative_f:
  loss[ener_spin]/relative_f (Argument), 123
  loss[ener]/relative_f (Argument), 120
remove_decay_rate() (in module deepmd_utils.compat), 586
replace_model_params_with_frz_multi_model() (in module deepmd_utils.multi_init), 536
replace_model_params_with_pretrained_model() (in module deepmd.utils.finetune), 530
reset_default_tf_session_config() (in module deepmd.env), 559
reset_get_batch() (deepmd.utils.data.DeepmdData method), 526
reset_get_batch() (deepmd.utils.data.DeepmdData method), 513
reset_get_batch() (deepmd.utils.data.DeepmdData method), 505
reset_get_batch() (deepmd.utils.DeepmdData method), 526
reset_get_batch() (deepmd.utils.data.DeepmdData method), 590
resnet_dt:
    model/type_embedding/resnet_dt (Argument), 90
    model[standard]/descriptor[se_a_ebd_v2]/resnet_dt (Argument), 108
    model[standard]/descriptor[se_a_ebd_v2]/resnet_dt (Argument), 108
    model[standard]/descriptor[se_a_mask]/resnet_dt (Argument), 110
    model[standard]/descriptor[se_a_tpe]/resnet_dt (Argument), 98
    model[standard]/descriptor[se_atten_v2]/resnet_dt (Argument), 106
    model[standard]/descriptor[se_atten_v2]/resnet_dt (Argument), 106
    model[standard]/descriptor[se_atten_v2]/resnet_dt (Argument), 106
    model[standard]/descriptor[se_atten_v2]/resnet_dt (Argument), 106
    model[standard]/descriptor[se_e2_a]/resnet_dt (Argument), 95
    model[standard]/descriptor[se_e2_a]/resnet_dt (Argument), 95
    model[standard]/descriptor[se_e2_r]/resnet_dt (Argument), 101
    model[standard]/descriptor[se_e2_r]/resnet_dt (Argument), 101
    model[standard]/descriptor[se_e3]/resnet_dt (Argument), 97
    model[standard]/descriptor[se_e3]/resnet_dt (Argument), 97
    model[standard]/fitting_net[dipole]/resnet_dt (Argument), 116
    model[standard]/fitting_net[dipole]/resnet_dt (Argument), 116
    model[standard]/fitting_net[dos]/resnet_dt (Argument), 114
    model[standard]/fitting_net[dos]/resnet_dt (Argument), 114
    model[standard]/fitting_net[ener]/resnet_dt (Argument), 112
    model[standard]/fitting_net[ener]/resnet_dt (Argument), 112
    model[standard]/fitting_net[polar]/resnet_dt (Argument), 115
    model[standard]/fitting_net[polar]/resnet_dt (Argument), 115
restore_descriptor:
nvnmd/restore_descriptor (Argument), 130
restore_fitting_net:
nvnmd/restore_fitting_net (Argument), 130
reverse_bin() (deepmd.nvnmd.utils.Encode method), 495
reverse_bin() (deepmd.nvnmd.utils.encode.Encode method), 502
reverse_map() (deepmd.DeepEval static method), 221
reverse_map() (deepmd.infer.deep_eval.DeepEval static method), 401
reverse_map() (deepmd.infer.DeepEval static method), 379
rglob() (deepmd.utils.path.DPH5Path method), 542
rglob() (deepmd.utils.path.DPH5Path method), 542
rglob() (deepmd.utils.path.DPH5Path method), 544
rglob() (deepmd.utils.path.DPH5Path method), 546
rglob() (deepmd_utils.utils.path.DPH5Path method), 596
rglob() (deepmd_utils.utils.path.DPH5Path method), 597
rglob() (deepmd_utils.utils.path.DPH5Path method), 598
run_s2g() (deepmd.nvnmd.entrypoints.mapt.MapTable method), 490
run_s2g() (deepmd.nvnmd.entrypoints.mapt.MapTable method), 485
run_s2g() (deepmd.nvnmd.entrypoints.MapTable method), 485
run_t2g() (deepmd.nvnmd.entrypoints.mapt.MapTable method), 490
run_t2g() (deepmd.nvnmd.entrypoints.mapt.MapTable method), 485
run_u2s() (deepmd.nvnmd.entrypoints.mapt.MapTable method), 490
run_u2s() (deepmd.nvnmd.entrypoints.mapt.MapTable method), 485
RunOptions (class in deepmd.train.run_options), 507
safe_cast_tensor() (in module deepmd_common), 558
save() (deepmd.nvnmd.utils.config.NvnmdConfig method), 499
save() (deepmd.nvnmd.utils.config.NvnmdConfig method), 503
save() (deepmd.nvnmd.utils.config.FioNpyDic method), 504
save() (deepmd.nvnmd.utils.config.FioNpyDic method), 504
save() (deepmd.nvnmd.utils.FioNpyDic method), 504
save() (deepmd.nvnmd.utils.FioNpyDic method), 504
save() (deepmd.nvnmd.utils.FioNpyDic method), 504
save() (deepmd.nvnmd.utils.FioNpyDic method), 504
save() (deepmd.nvnmd.utils.FioNpyDic method), 504
save_checkpoint() (deepmd.train.trainer.DPTrainer method), 509
save_ckpt: training/save_ckpt (Argument), 128
save_compressed() (deepmd.train.trainer.DPTrainer method), 509
save_dp_model() (in module deepmd_utils.model_format), 570
save_dp_model() (in module deepmd_utils.model_format.network), 575
save_freq: training/save_freq (Argument), 128
save_weight() (in module deepmd.nvnmd.entrypoints), 486
save_weight() (in module deepmd.nvnmd.entrypoints.freeze), 487
scale:
  model[standard]/fitting_net[polar]/scale
    (Argument), 115
scale_by_worker:
  learning_rate/scale_by_worker (Argument), 115
seed() (in module deepmd.utils.random), 547
seed() (in module deepmd.utils.utils.random), 600
seed:
  model/type_embedding/seed (Argument), 91
  model[standard]/descriptor[se_a_ebd_v2]/seed
    select_idx_map() (in module deepmd.common), 558
  model[standard]/descriptor[se_a_mask]/seed
    select_idx_map() (in module deepmd.utils.utils.random), 603
  model[standard]/descriptor[se_a_tpe]/seed
  model[standard]/fitting_net[dipole]/seed
  model[standard]/fitting_net[dos]/seed
  model[standard]/fitting_net[ener]/seed
  model[standard]/fitting_net[polar]/seed
    (Argument), 112
training/seed (Argument), 128
sel:
  model[pairtab]/sel (Argument), 117
  model[standard]/descriptor[se_a_ebd_v2]/sel
    (Argument), 107
  model[standard]/descriptor[se_a_mask]/sel
    (Argument), 109
  model[standard]/descriptor[se_a_tpe]/sel
    (Argument), 97
  model[standard]/descriptor[se_atten_v2]/sel
    (Argument), 104
  model[standard]/descriptor[se_atten]/sel
    (Argument), 102
  model[standard]/descriptor[se_e2_a]/sel
    (Argument), 94
  model[standard]/descriptor[se_e2_r]/sel
    (Argument), 100
  model[standard]/descriptor[se_e3]/sel
    (Argument), 96
  model[standard]/descriptor[loc_frame]/sel_a
    (Argument), 93
  sel_r:
    model[standard]/descriptor[loc_frame]/sel_r
      (Argument), 93
  sel_type:
    model[standard]/fitting_net[dipole]/sel_type
      (Argument), 116
    model[standard]/fitting_net[polar]/sel_type
      (Argument), 115
set_davg_zero:
  model[standard]/descriptor[se_a_ebd_v2]/set_davg_zero
    (Argument), 109
  model[standard]/descriptor[se_a_tpe]/set_davg_zero
    (Argument), 99
  model[standard]/descriptor[se_atten_v2]/set_davg_zero
    (Argument), 107
  model[standard]/descriptor[se_atten]/set_davg_zero
    (Argument), 104
  model[standard]/descriptor[se_e2_a]/set_davg_zero
    (Argument), 96
  model[standard]/descriptor[se_e2_r]/set_davg_zero
    (Argument), 101
  model[standard]/descriptor[se_e3]/set_davg_zero
    (Argument), 97
set_log_handles() (in module deepmd.loggers), 418
set_log_handles() (in module deepmd.loggers.loggers), 419
set_log_handles() (in module deepmd.utils.loggers), 561
set_log_handles() (in module deepmd_utils.loggers.loggers), 562
set_ntype() (deepmd.nvnmd.utils.config.NvnmdConfig method), 499
set_prefix:
  training/training_data/set_prefix (Argument), 125
  training/validation_data/set_prefix (Argument), 126
set_scikit_build_env() (in module backend.read_env), 218
set_sys_probs() (deepmd.utils.data_system.DeepmdDataSystem method), 529
set_sys_probs() (deepmd.utils.DeepmdDataSystem method), 516
set_sys_probs() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 593
shift_diag:
  model[standard]/fitting_net[polar]/shift_diag (Argument), 115
shuffle() (in module deepmd.utils.random), 547
SimulationRegion (C++ class), 802
SimulationRegion::~SimulationRegion (C++ function), 802
SimulationRegion::affineTransform (C++ function), 802
SimulationRegion::backup (C++ function), 802
SimulationRegion::compactIndex (C++ function), 803
SimulationRegion::computeShiftVec (C++ function), 804
SimulationRegion::DBOX_XX (C++ member), 804
SimulationRegion::DBOX_YY (C++ member), 804
SimulationRegion::DBOX_ZZ (C++ member), 804
SimulationRegion::diffNearestNeighbor (C++ function), 803
SimulationRegion::getBoxOrigin (C++ function), 802, 803
SimulationRegion::getBoxTensor (C++ function), 802
SimulationRegion::getInterShiftVec (C++ function), 804
SimulationRegion::getNullShiftIndex (C++ function), 803
SimulationRegion::getNumbShiftVec (C++ function), 803
SimulationRegion::getRecBoxTensor (C++ function), 802
SimulationRegion::getShiftIndex (C++ function), 803
SimulationRegion::getShiftVec (C++ function), 803
SimulationRegion::getShiftVecTotalSize (C++ function), 803
SimulationRegion::getVolume (C++ function), 803
SimulationRegion::index3to1 (C++ function), 804
SimulationRegion::inter2Phys (C++ function), 803
SimulationRegion::inter_shift_vec (C++ member), 804
SimulationRegion::isPeriodic (C++ function), 803
SimulationRegion::NBOX_XX (C++ member), 804
SimulationRegion::NBOX_YY (C++ member), 804
SimulationRegion::NBOX_ZZ (C++ member), 804
SimulationRegion::phys2Inter (C++ function), 803
SimulationRegion::recover (C++ function), 802
SimulationRegion::reinitBox (C++ function), 802
SimulationRegion::reinitOrigin (C++ function), 802
SimulationRegion::shift_info_size (C++ member), 804
SimulationRegion::shift_vec (C++ member), 804
SimulationRegion::shift_vec_size (C++ member), 804
SimulationRegion::shiftCoord (C++ function), 803
SimulationRegion::SimulationRegion (C++ function), 802
SimulationRegion::SPACENDIM (C++ member), 804
SimulationRegion::toFaceDistance (C++ function), 803
smin_alpha:
  model/smin_alpha (Argument), 89
smooth_type_embdding:
  model[standard]/descriptor[se_atten]/smooth_type_embdd (Argument), 104
soft_min_force() (in module deepmd.env.op_module), 651
soft_min_force_grad() (in module deepmd.env.op_grads_module), 666
soft_min_switch() (in module deepmd.env.op_module), 651
soft_min_virial() (in module deepmd.env.op_module), 652
soft_min_virial_grad() (in module deepmd.env.op_grads_module), 666
SoftMinForce() (in module deepmd.env.op_module), 625
SoftMinForceGrad() (in module deepmd.env.op_grads_module), 662
SoftMinSwitch() (in module deepmd.env.op_module), 662

916 Index
DeePMD-kit

tabulate_fusion() (in module deepmd.env.op_module), 652

tabulate_fusion_grad() (in module deepmd.env.op_module), 652

tabulate_fusion_grad_grad() (in module deepmd.env.op_module), 653

tabulate_fusion_se_a() (in module deepmd.env.op_module), 653

tabulate_fusion_se_a_grad() (in module deepmd.env.op_module), 653

tabulate_fusion_se_a_grad_grad() (in module deepmd.env.op_module), 654

tabulate_fusion_se_atten() (in module deepmd.env.op_module), 654

tabulate_fusion_se_atten_grad() (in module deepmd.env.op_module), 655

tabulate_fusion_se_atten_grad_grad() (in module deepmd.env.op_module), 655

tabulate_fusion_se_r() (in module deepmd.env.op_module), 656

tabulate_fusion_se_r_grad() (in module deepmd.env.op_module), 656

tabulate_fusion_se_r_grad_grad() (in module deepmd.env.op_module), 656

tabulate_fusion_se_t() (in module deepmd.env.op_module), 656

tabulate_fusion_se_t_grad() (in module deepmd.env.op_module), 657

tabulate_fusion_se_t_grad_grad() (in module deepmd.env.op_module), 657

TabulateCheckerGPUExecuteFunctor (C++ struct), 801

TabulateCheckerGPUExecuteFunctor::operator() (C++ function), 801

TabulateFusion() (in module deepmd.env.op_module), 626

TabulateFusionGPUExecuteFunctor (C++ struct), 801

TabulateFusionGPUExecuteFunctor::operator() (C++ function), 801

TabulateFusionGrad() (in module deepmd.env.op_module), 626

TabulateFusionGradGPUExecuteFunctor (C++ struct), 802

TabulateFusionGradGPUExecuteFunctor::operator() (C++ function), 802

TabulateFusionGradGrad() (in module deepmd.env.op_module), 627

TabulateFusionSeA() (in module deepmd.env.op_module), 627

TabulateFusionSeAGrad() (in module deepmd.env.op_module), 627

TabulateFusionSeAGradGrad() (in module deepmd.env.op_module), 628

TabulateFusionSeAtten() (in module deepmd.env.op_module), 628

TabulateFusionSeAttenGrad() (in module deepmd.env.op_module), 628

TabulateFusionSeAttenGradGrad() (in module deepmd.env.op_module), 629

TabulateFusionSeR() (in module deepmd.env.op_module), 629

TabulateFusionSeRGrad() (in module deepmd.env.op_module), 630

TabulateFusionSeRGradGrad() (in module deepmd.env.op_module), 630

TabulateFusionSeT() (in module deepmd.env.op_module), 630

TabulateFusionSeTGrad() (in module deepmd.env.op_module), 631

TabulateFusionSeTGradGrad() (in module deepmd.env.op_module), 631

tanh4_t (in module deepmd.nvnmd.utils.network), 505

tanh4_t_flt_nvnmd() (in module deepmd.env.op_module), 658

TabulateFusionSeTGradGrad() (in module deepmd.env.op_module), 631

tensorboard:

  training/tensorboard (Argument), 129

tensorboard_freq:

  training/tensorboard_freq (Argument), 129

tensorboard_log_dir:

  training/tensorboard_log_dir (Argument), 129

TensorLoss (class in deepmd.loss), 425

TensorLoss (class in deepmd.loss.tensor), 433

TensorModel (class in deepmd.model.tensor), 477

tensors (deepmd.infer.deep_tensor.DeepTensor attribute), 412

test() (in module deepmd.entrypoints), 335

test() (in module deepmd.entrypoints.test), 340

time_training:

  training/time_training (Argument), 128

TPB (C macro), 850

train() (deepmd.train.trainer.DPTrainer method), 509

train() (in module deepmd.entrypoints.train), 341

train_nvnmd() (in module deepmd.nvnmd.entrypoints), 336

trainable:

  model/type_embedding/trainable (Argument), 91

  model[standard]/descriptor[se_a_ebd_v2]/trainable (Argument), 109

  model[standard]/descriptor[se_a_mask]/trainable (Argument), 110
<table>
<thead>
<tr>
<th>Document Path</th>
<th>Argument</th>
<th>Line Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>model[standard]/descriptor[se_e2_a]</td>
<td>trainable</td>
<td>106</td>
</tr>
<tr>
<td>model[standard]/descriptor[se_e2_r]</td>
<td>trainable</td>
<td>101</td>
</tr>
<tr>
<td>model[standard]/descriptor[se_e3]</td>
<td>trainable</td>
<td>97</td>
</tr>
<tr>
<td>model[standard]/fitting_net[dos]</td>
<td>trainable</td>
<td>114</td>
</tr>
<tr>
<td>model[standard]/fitting_net[ener]</td>
<td>trainable</td>
<td>112</td>
</tr>
<tr>
<td>training</td>
<td></td>
<td>125</td>
</tr>
<tr>
<td>training/data_dict</td>
<td></td>
<td>129</td>
</tr>
<tr>
<td>training/disp_file</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>training/disp_freq</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>training/disp_training</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>training/enable_profiler</td>
<td></td>
<td>129</td>
</tr>
<tr>
<td>training/fitting_weight</td>
<td></td>
<td>129</td>
</tr>
<tr>
<td>training/mixed_precision</td>
<td></td>
<td>127</td>
</tr>
<tr>
<td>training/mixed_precision/compute_prec</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>training/mixed_precision/output_prec</td>
<td></td>
<td>127</td>
</tr>
<tr>
<td>training/numb_steps</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>training/profileing</td>
<td></td>
<td>129</td>
</tr>
<tr>
<td>training/profileing_file</td>
<td></td>
<td>129</td>
</tr>
<tr>
<td>training/save_ckpt</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>training/save_freq</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>training/seed</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>training/tensorboard</td>
<td></td>
<td>129</td>
</tr>
<tr>
<td>training/tensorboard_freq</td>
<td></td>
<td>129</td>
</tr>
<tr>
<td>tensorboard_log_dir</td>
<td></td>
<td>129</td>
</tr>
<tr>
<td>time_training</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>training_data</td>
<td></td>
<td>126</td>
</tr>
<tr>
<td>training_data/auto_prob</td>
<td></td>
<td>126</td>
</tr>
<tr>
<td>training_data/batch_size</td>
<td></td>
<td>125</td>
</tr>
<tr>
<td>training_data/set_prefix</td>
<td></td>
<td>125</td>
</tr>
<tr>
<td>training_data/sys_probs</td>
<td></td>
<td>126</td>
</tr>
<tr>
<td>training_data/systems</td>
<td></td>
<td>125</td>
</tr>
<tr>
<td>training/validation_data</td>
<td></td>
<td>126</td>
</tr>
<tr>
<td>training/validation_data/auto_prob</td>
<td></td>
<td>127</td>
</tr>
<tr>
<td>training/validation_data/batch_size</td>
<td></td>
<td>127</td>
</tr>
<tr>
<td>training/validation_data/numb_btch</td>
<td></td>
<td>127</td>
</tr>
<tr>
<td>training/validation_data/set_prefix</td>
<td></td>
<td>126</td>
</tr>
<tr>
<td>training/validation_data/sys_probs</td>
<td></td>
<td>127</td>
</tr>
<tr>
<td>training/validation_data/systems</td>
<td></td>
<td>126</td>
</tr>
<tr>
<td>training</td>
<td></td>
<td>125</td>
</tr>
<tr>
<td>type:</td>
<td></td>
<td>118</td>
</tr>
<tr>
<td>learning_rate/type</td>
<td></td>
<td>119</td>
</tr>
</tbody>
</table>
DeePMD-kit

model/modifier/type (Argument), 91
model/type (Argument), 92
model[standard]/descriptor/type (Argument), 92
model[standard]/fitting_net/type (Argument), 111
type_embedding:
  model/type_embedding (Argument), 90
type_embedding_args() (in module deepmd.utils.argcheck), 519
type_embedding_args() (in module deepmd_utils.utils.argcheck), 583
type_map:
  model/type_map (Argument), 89
type_nchnl:
  model[standard]/descriptor[se_a_tpe]/type_nchnl (Argument), 99
type_nlayer:
  model[standard]/descriptor[se_a_tpe]/type_nlayer (Argument), 99
type_one_side:
  model[standard]/descriptor[se_e2_a_tpe]/type_one_side (Argument), 108
  model[standard]/descriptor[se_a_mask]/type_one_side (Argument), 110
  model[standard]/descriptor[se_a_tpe]/type_one_side (Argument), 98
  model[standard]/descriptor[se_atten_v2]/type_one_side (Argument), 106
  model[standard]/descriptor[se_atten]/type_one_side (Argument), 103
  model[standard]/descriptor[se_e2_a]/type_one_side (Argument), 95
  model[standard]/descriptor[se_e2_r]/type_one_side (Argument), 101
TypeEmbedNet (class in deepmd.utils.type_embedding), 550

U
U_Flt64_Int64 (C++ union), 805
U_Flt64_Int64::nflt (C++ member), 805
U_Flt64_Int64::nint (C++ member), 805
uint_64 (C++ type), 851
unaggregated_dy2_dx() (in module deepmd.env.op_module), 658
unaggregated_dy2_dx_s() (in module deepmd.env.op_module), 658
unaggregated_dy_dx() (in module deepmd.env.op_module), 658
unaggregated_dy_dx_s() (in module deepmd.env.op_module), 659
UnaggregatedDy2Dx() (in module deepmd.env.op_module), 632
UnaggregatedDyDx() (in module deepmd.env.op_module), 632
UnaggregatedDyDxS() (in module deepmd.env.op_module), 632
update() (deepmd.nvnmd.utils.fio.FioDic method), 503
update() (deepmd.nvnmd.utils.FioDic method), 495
update_config() (deepmd.nvnmd.utils.config.NvnmdConfig method), 499
update_deepmd_input() (in module deepmd.utils.compat), 520
update_deepmd_input() (in module deepmd.utils.compat), 586
update_sel() (deepmd.descriptor.descriptor.Descriptor class method), 233
update_sel() (deepmd.descriptor.descriptor.Descriptor class method), 285
update_sel() (deepmd.descriptor.Descriptor class method), 238
update_sel() (deepmd.descriptor.Descriptor class method), 242
update_sel() (deepmd.descriptor.Descriptor class method), 290
update_sel() (deepmd.descriptor.hybrid.DescriptorHybrid class method), 294
update_sel() (deepmd.descriptor.se.DescrptSe class method), 296
update_sel() (deepmd.descriptor.se_a.DescrptSeA class method), 316
update_sel() (deepmd.descriptor.se_atten.DescrptSeAtt method), 321
update_sel() (deepmd.descriptor.se_attn.DescrptSeAttn class method), 454
update_sel() (deepmd.descriptor.se_attn.DescrptSeAttn class method), 458
update_sel() (deepmd.descriptor.multi.MultiModel class method), 463
update_sel() (deepmd.descriptor.multi.MultiModel class method), 469
update_sel() (deepmd.descriptor.multi.MultiModel class method), 444
update_sel() (deepmd.descriptor.pairtab.PairTabModel class method), 471
update_sel() (deepmd.descriptor.pairwise_dprc.PairwiseDPRc class method), 474
use_aparam_as_mask:
DeePMD-kit

model[standard]/fitting_net[ener]/use_aparamas_bias (Argument), 113

use_spins:
  model/spin/use_spins (Argument), 92

use_srtab:
  model/use_srtab (Argument), 89

V

valid_on_the_fly() (deepmd.train.trainer.DPTrainer method), 509

validation_data:
  training/validation_data (Argument), 126

validation_data_args() (in module deepmd.utils.utils.argcheck), 583

value() (deepmd.utils.learning_rate.LearningRateExp method), 536

value() (deepmd.utils.LearningRateExp method), 517

variable_summaries() (in module deepmd.utils.network), 539

VariantABCMeta (class in deepmd.utils.plugin), 546

VariantABCMeta (class in deepmd.utils.plugin), 599

VariantMeta (class in deepmd.utils.plugin), 546

VariantMeta (class in deepmd.utils.plugin), 600

version:
  nvnmd/version (Argument), 130

virtual_len:
  model/spin/virtual_len (Argument), 92

W

weight_file:
  nvnmd/weight_file (Argument), 130

weighted_average() (in module deepmd.utils.weight_avg), 552

weighted_average() (in module deepmd.utils.weight_avg), 601

weights:
  model[linear_ener]/weights (Argument), 118

WFCModel (class in deepmd.model), 445

WFCModel (class in deepmd.model.tensor), 479

world_size (deepmd.train.run_options.RunOptions attribute), 508

Wrap (class in deepmd.nvnmd.entrypoints), 485

Wrap (class in deepmd.nvnmd.entrypoints.wrap), 491

wrap() (deepmd.nvnmd.entrypoints.Wrap method), 486

wrap() (deepmd.nvnmd.entrypoints.wrap.Wrap method), 491

wrap() (in module deepmd.nvnmd.entrypoints.wrap), 492

wrap_bias() (deepmd.nvnmd.entrypoints.Wrap method), 486

write_model_devi_out() (in module deepmd.infer.model_devi), 417