DeePMD-kit

DeepModeling

May 24, 2023

GETTING STARTED

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

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

- Han Wang, Linfeng Zhang, Jiequn Han, and Weinan E. "DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics." Computer Physics Communications 228 (2018): 178-184.
- Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang. "DeePMD-kit v2: A software package for Deep Potential models." arXiv:2304.09409.

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

CHAPTER

ONE

GETTING STARTED

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

1.1 Easy install

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

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

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

- 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 splited 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
```

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

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

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

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

One may specify the DeePMD-kit version such as `2.1.1` 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 lammps -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/dpmdkit-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 11 supported:

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

`cu11` is required only when CUDA Toolkit and cuDNN were not installed.

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,cu11,lmp,ipi]
```

MPICH is required for parallel running.

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 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

1.2.3 Get tutorial data via git

```
! 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`

```
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.

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

```
['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 splited randomly as training and validation data.

```
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
index_validation = np.random.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")

# 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 dat.

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
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

```
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.

```
"model":{
    "type_map":    ["H", "C"],
    "descriptor":{
        "type":            "se_e2_a",
        "rcut":            6.00,
        "rcut_smth":       0.50,
```

(continues on next page)

(continued from previous page)

```
        "sel":              "auto",
        "neuron":           [25, 50, 100],
        "resnet_dt":        false,
        "axis_neuron":      16,
        "seed":             1,
        "_comment":         "that's all"
        },
    "fitting_net":{
        "neuron":           [240, 240, 240],
        "resnet_dt":        true,
        "seed":             1,
        "_comment":         "that's all"
    },
    "_comment":    "that's all"'
},
```

The explanation for some of the parameters is as follows:

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

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

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

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

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

The training parameters are given in the following

```
    "training" : {
        "training_data": {
            "systems":            ["../00.data/training_data"],
            "batch_size":         "auto",
            "_comment":           "that's all"
        },
        "validation_data":{
            "systems":            ["../00.data/validation_data/"],
            "batch_size":         "auto",
            "numb_btch":          1,
            "_comment":           "that's all"
        },
        "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␣
↪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
/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)
DEEPMD INFO    Calculate neighbor statistics... (add --skip-neighbor-stat to skip this step)
2023-04-20 23:35:59.335932: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could␣
↪not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file:␣
↪No such file or directory; LD_LIBRARY_PATH: /usr/local/nvidia/lib:/usr/local/nvidia/lib64
2023-04-20 23:35:59.335979: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to␣
↪cuInit: UNKNOWN ERROR (303)
OMP: Info #155: KMP_AFFINITY: Initial OS proc set respected: 0-7
OMP: Info #216: KMP_AFFINITY: decoding x2APIC ids.
OMP: Info #216: KMP_AFFINITY: cpuid leaf 11 not supported.
```

(continues on next page)

(continued from previous page)

```
OMP: Info #216: KMP_AFFINITY: decoding legacy APIC ids.
OMP: Info #157: KMP_AFFINITY: 8 available OS procs
OMP: Info #158: KMP_AFFINITY: Uniform topology
OMP: Info #287: KMP_AFFINITY: topology layer "LL cache" is equivalent to "socket".
OMP: Info #192: KMP_AFFINITY: 1 socket x 4 cores/socket x 2 threads/core (4 total cores)
OMP: Info #218: KMP_AFFINITY: OS proc to physical thread map:
OMP: Info #172: KMP_AFFINITY: OS proc 0 maps to socket 0 core 0 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 1 maps to socket 0 core 0 thread 1
OMP: Info #172: KMP_AFFINITY: OS proc 2 maps to socket 0 core 1 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 3 maps to socket 0 core 1 thread 1
OMP: Info #172: KMP_AFFINITY: OS proc 4 maps to socket 0 core 2 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 5 maps to socket 0 core 2 thread 1
OMP: Info #172: KMP_AFFINITY: OS proc 6 maps to socket 0 core 3 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 7 maps to socket 0 core 3 thread 1
OMP: Info #254: KMP_AFFINITY: pid 118 tid 140 thread 1 bound to OS proc set 2
OMP: Info #254: KMP_AFFINITY: pid 118 tid 142 thread 2 bound to OS proc set 4
OMP: Info #254: KMP_AFFINITY: pid 118 tid 144 thread 4 bound to OS proc set 1
OMP: Info #254: KMP_AFFINITY: pid 118 tid 143 thread 3 bound to OS proc set 6
OMP: Info #254: KMP_AFFINITY: pid 118 tid 145 thread 5 bound to OS proc set 3
OMP: Info #254: KMP_AFFINITY: pid 118 tid 146 thread 6 bound to OS proc set 5
OMP: Info #254: KMP_AFFINITY: pid 118 tid 147 thread 7 bound to OS proc set 7
OMP: Info #254: KMP_AFFINITY: pid 118 tid 148 thread 8 bound to OS proc set 0
OMP: Info #254: KMP_AFFINITY: pid 118 tid 139 thread 9 bound to OS proc set 2
OMP: Info #254: KMP_AFFINITY: pid 118 tid 149 thread 10 bound to OS proc set 4
OMP: Info #254: KMP_AFFINITY: pid 118 tid 150 thread 11 bound to OS proc set 6
OMP: Info #254: KMP_AFFINITY: pid 118 tid 151 thread 12 bound to OS proc set 1
OMP: Info #254: KMP_AFFINITY: pid 118 tid 152 thread 13 bound to OS proc set 3
OMP: Info #254: KMP_AFFINITY: pid 118 tid 153 thread 14 bound to OS proc set 5
OMP: Info #254: KMP_AFFINITY: pid 118 tid 154 thread 15 bound to OS proc set 7
OMP: Info #254: KMP_AFFINITY: pid 118 tid 155 thread 16 bound to OS proc set 0
DEEPMD INFO    training data with min nbor dist: 1.045920568611028
DEEPMD INFO    training data with max nbor size: [4 1]
DEEPMD INFO     _____               _____   __  __  _____           _     _  _
DEEPMD INFO    |  __ \             |  __ \ |  \/  ||  __ \         | |   (_)| |
DEEPMD INFO    | |  | |  ___   ___ | |__) || \  / || |  | | ______ | | __ _ | |_
DEEPMD INFO    | |  | | / _ \ / _ \|  ___/ | |\/| || |  | ||______|| |/ /| || __|
DEEPMD INFO    | |__| ||  __/|  __/| |     | |  | || |__| |        |   < | || |_
DEEPMD INFO    |_____/  \___| \___||_|     |_|  |_||_____/         |_|\_\|_| \__|
DEEPMD INFO    Please read and cite:
DEEPMD INFO    Wang, Zhang, Han and E, Comput.Phys.Comm. 228, 178-184 (2018)
DEEPMD INFO    installed to:         /home/conda/feedstock_root/build_artifacts/deepmd-kit_
↪1678943793317/work/_skbuild/linux-x86_64-3.10/cmake-install
DEEPMD INFO    source :              v2.2.1
DEEPMD INFO    source brach:         HEAD
DEEPMD INFO    source commit:        3ac8c4c7
DEEPMD INFO    source commit at:     2023-03-16 12:33:24 +0800
DEEPMD INFO    build float prec:     double
DEEPMD INFO    build variant:        cuda
DEEPMD INFO    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
DEEPMD INFO    build with tf lib:
DEEPMD INFO    ---Summary of the training---------------------------------------
DEEPMD INFO    running on:           bohrium-14076-1013950
DEEPMD INFO    computing device:     cpu:0
DEEPMD INFO    CUDA_VISIBLE_DEVICES: unset
DEEPMD INFO    Count of visible GPU: 0
```

(continues on next page)

(continued from previous page)

```
DEEPMD INFO    num_intra_threads:    0
DEEPMD INFO    num_inter_threads:    0
DEEPMD INFO    -----------------------------------------------------------------
DEEPMD INFO    ---Summary of DataSystem: training     -----------------------------------------------
↪--
DEEPMD INFO    found 1 system(s):
DEEPMD INFO                                    system  natoms  bch_sz   n_bch   prob  pbc
DEEPMD INFO                      ../00.data/training_data       5       7      23  1.000    T
DEEPMD INFO    -------------------------------------------------------------------------------------
↪--
DEEPMD INFO    ---Summary of DataSystem: validation   -----------------------------------------------
↪--
DEEPMD INFO    found 1 system(s):
DEEPMD INFO                                    system  natoms  bch_sz   n_bch   prob  pbc
DEEPMD INFO                    ../00.data/validation_data       5       7       5  1.000    T
DEEPMD INFO    -------------------------------------------------------------------------------------
↪--
DEEPMD INFO    training without frame parameter
DEEPMD INFO    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
DEEPMD INFO    built lr
DEEPMD INFO    built network
DEEPMD INFO    built training
WARNING:root:To get the best performance, it is recommended to adjust the number of threads by␣
↪setting the environment variables OMP_NUM_THREADS, TF_INTRA_OP_PARALLELISM_THREADS, and TF_INTER_
↪OP_PARALLELISM_THREADS. See https://deepmd.rtfd.io/parallelism/ for more information.
DEEPMD INFO    initialize model from scratch
DEEPMD INFO    start training at lr 1.00e-03 (== 1.00e-03), decay_step 50, decay_rate 0.950006,␣
↪final lr will be 3.51e-08
DEEPMD INFO    batch     200 training time 6.10 s, testing time 0.02 s
DEEPMD INFO    batch     400 training time 4.83 s, testing time 0.02 s
DEEPMD INFO    batch     600 training time 4.84 s, testing time 0.02 s
DEEPMD INFO    batch     800 training time 4.85 s, testing time 0.02 s
DEEPMD INFO    batch    1000 training time 4.85 s, testing time 0.02 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    1200 training time 4.86 s, testing time 0.02 s
DEEPMD INFO    batch    1400 training time 5.39 s, testing time 0.02 s
DEEPMD INFO    batch    1600 training time 4.84 s, testing time 0.02 s
DEEPMD INFO    batch    1800 training time 4.86 s, testing time 0.02 s
DEEPMD INFO    batch    2000 training time 4.84 s, testing time 0.02 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    2200 training time 4.86 s, testing time 0.02 s
DEEPMD INFO    batch    2400 training time 4.90 s, testing time 0.02 s
DEEPMD INFO    batch    2600 training time 4.87 s, testing time 0.02 s
DEEPMD INFO    batch    2800 training time 4.84 s, testing time 0.02 s
DEEPMD INFO    batch    3000 training time 4.86 s, testing time 0.02 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    3200 training time 4.86 s, testing time 0.02 s
DEEPMD INFO    batch    3400 training time 4.99 s, testing time 0.02 s
DEEPMD INFO    batch    3600 training time 4.88 s, testing time 0.02 s
DEEPMD INFO    batch    3800 training time 4.85 s, testing time 0.02 s
DEEPMD INFO    batch    4000 training time 4.88 s, testing time 0.02 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    4200 training time 4.88 s, testing time 0.02 s
DEEPMD INFO    batch    4400 training time 4.86 s, testing time 0.02 s
DEEPMD INFO    batch    4600 training time 4.92 s, testing time 0.02 s
```

(continues on next page)

(continued from previous page)

```
DEEPMD INFO    batch    4800 training time 4.86 s, testing time 0.02 s
DEEPMD INFO    batch    5000 training time 4.86 s, testing time 0.02 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    5200 training time 4.87 s, testing time 0.02 s
DEEPMD INFO    batch    5400 training time 4.88 s, testing time 0.02 s
DEEPMD INFO    batch    5600 training time 4.87 s, testing time 0.02 s
DEEPMD INFO    batch    5800 training time 4.87 s, testing time 0.02 s
DEEPMD INFO    batch    6000 training time 4.90 s, testing time 0.02 s
WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/tensorflow/python/
↪training/saver.py:1066: remove_checkpoint (from tensorflow.python.training.checkpoint_
↪management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to delete files with this prefix.
WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/tensorflow/python/
↪training/saver.py:1066: remove_checkpoint (from tensorflow.python.training.checkpoint_
↪management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to delete files with this prefix.
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    6200 training time 4.86 s, testing time 0.02 s
DEEPMD INFO    batch    6400 training time 4.87 s, testing time 0.02 s
DEEPMD INFO    batch    6600 training time 4.86 s, testing time 0.02 s
DEEPMD INFO    batch    6800 training time 4.84 s, testing time 0.02 s
DEEPMD INFO    batch    7000 training time 4.93 s, testing time 0.02 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    7200 training time 4.89 s, testing time 0.02 s
DEEPMD INFO    batch    7400 training time 4.88 s, testing time 0.02 s
DEEPMD INFO    batch    7600 training time 4.88 s, testing time 0.02 s
DEEPMD INFO    batch    7800 training time 4.87 s, testing time 0.02 s
DEEPMD INFO    batch    8000 training time 4.86 s, testing time 0.02 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    8200 training time 4.87 s, testing time 0.02 s
DEEPMD INFO    batch    8400 training time 4.85 s, testing time 0.02 s
DEEPMD INFO    batch    8600 training time 4.86 s, testing time 0.02 s
DEEPMD INFO    batch    8800 training time 4.87 s, testing time 0.02 s
DEEPMD INFO    batch    9000 training time 4.83 s, testing time 0.02 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    batch    9200 training time 4.87 s, testing time 0.02 s
DEEPMD INFO    batch    9400 training time 4.95 s, testing time 0.02 s
DEEPMD INFO    batch    9600 training time 4.89 s, testing time 0.02 s
DEEPMD INFO    batch    9800 training time 4.87 s, testing time 0.02 s
DEEPMD INFO    batch   10000 training time 4.88 s, testing time 0.02 s
DEEPMD INFO    saved checkpoint model.ckpt
DEEPMD INFO    average training time: 0.0244 s/batch (exclude first 200 batches)
DEEPMD INFO    finished training
DEEPMD INFO    wall time: 256.669 s
```

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

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

(continues on next page)

(continued from previous page)

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

and the starting and final learning rate of this training

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

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

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

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

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

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

and

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

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

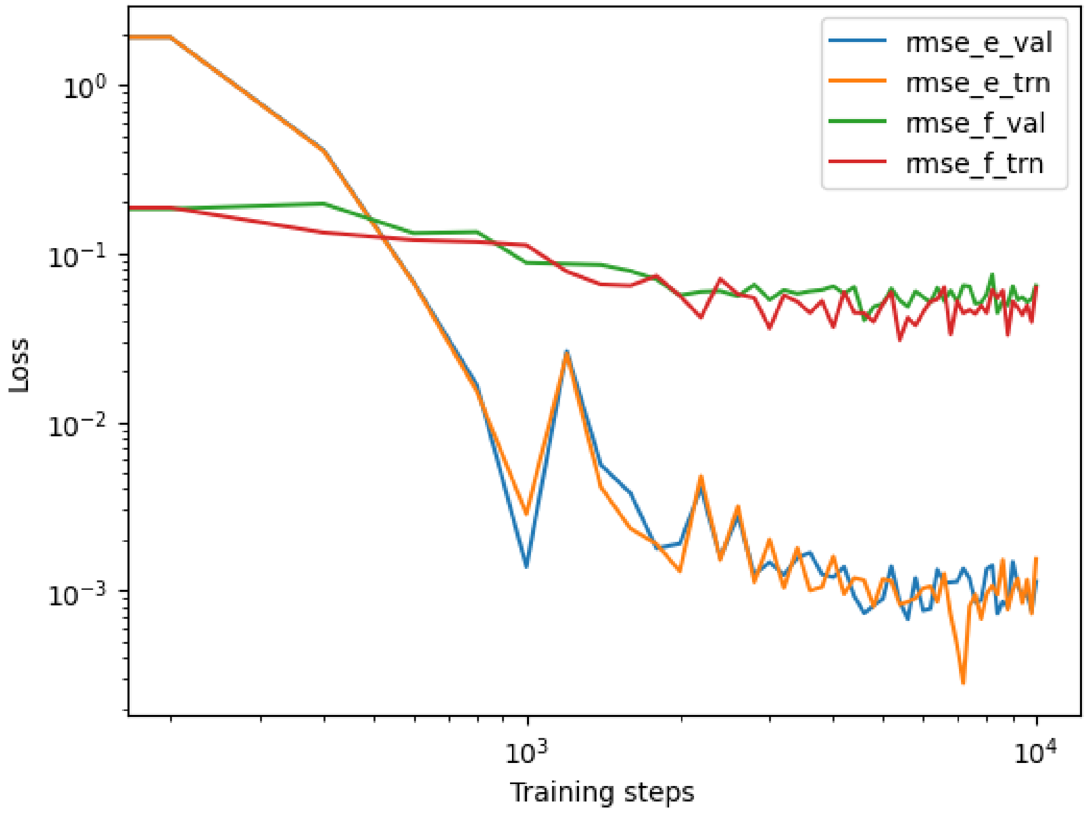
```
! head -n 2 lcurve.out && tail -n 2 lcurve.out
```

```
#  step       rmse_val    rmse_trn    rmse_e_val  rmse_e_trn    rmse_f_val  rmse_f_trn         lr
      0       2.06e+01    1.94e+01      1.34e-01    1.35e-01      6.51e-01    6.14e-01    1.0e-03
   9800       5.49e-02    4.00e-02      7.55e-04    7.28e-04      5.37e-02    3.91e-02    4.3e-08
  10000       6.56e-02    6.37e-02      1.13e-03    1.54e-03      6.44e-02    6.25e-02    3.5e-08
```

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

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

with open("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

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

```
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:
```

(continues on next page)

(continued from previous page)

```
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: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)
2023-04-20 23:40:25.666203: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could␣
↪not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file:␣
↪No such file or directory; LD_LIBRARY_PATH: /usr/local/nvidia/lib:/usr/local/nvidia/lib64
2023-04-20 23:40:25.666257: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to␣
↪cuInit: UNKNOWN ERROR (303)
DEEPMD INFO    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
```

(continues on next page)

(continued from previous page)

```
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: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)
2023-04-20 23:40:30.102300: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could␣
↪not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file:␣
↪No such file or directory; LD_LIBRARY_PATH: /usr/local/nvidia/lib:/usr/local/nvidia/lib64
2023-04-20 23:40:30.102346: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to␣
↪cuInit: UNKNOWN ERROR (303)
WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/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.
WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/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.
DEEPMD WARNING You can use the environment variable DP_INFER_BATCH_SIZE tocontrol the inference␣
↪batch size (nframes * natoms). The default value is 1024.
DEEPMD INFO    # ---------------output of dp test---------------
DEEPMD INFO    # testing system : ../00.data/validation_data
OMP: Info #155: KMP_AFFINITY: Initial OS proc set respected: 0-7
OMP: Info #216: KMP_AFFINITY: decoding x2APIC ids.
OMP: Info #216: KMP_AFFINITY: cpuid leaf 11 not supported.
OMP: Info #216: KMP_AFFINITY: decoding legacy APIC ids.
OMP: Info #157: KMP_AFFINITY: 8 available OS procs
OMP: Info #158: KMP_AFFINITY: Uniform topology
OMP: Info #287: KMP_AFFINITY: topology layer "LL cache" is equivalent to "socket".
OMP: Info #192: KMP_AFFINITY: 1 socket x 4 cores/socket x 2 threads/core (4 total cores)
OMP: Info #218: KMP_AFFINITY: OS proc to physical thread map:
OMP: Info #172: KMP_AFFINITY: OS proc 0 maps to socket 0 core 0 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 1 maps to socket 0 core 0 thread 1
OMP: Info #172: KMP_AFFINITY: OS proc 2 maps to socket 0 core 1 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 3 maps to socket 0 core 1 thread 1
OMP: Info #172: KMP_AFFINITY: OS proc 4 maps to socket 0 core 2 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 5 maps to socket 0 core 2 thread 1
OMP: Info #172: KMP_AFFINITY: OS proc 6 maps to socket 0 core 3 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 7 maps to socket 0 core 3 thread 1
OMP: Info #254: KMP_AFFINITY: pid 254 tid 265 thread 1 bound to OS proc set 2
OMP: Info #254: KMP_AFFINITY: pid 254 tid 267 thread 2 bound to OS proc set 4
OMP: Info #254: KMP_AFFINITY: pid 254 tid 268 thread 3 bound to OS proc set 6
OMP: Info #254: KMP_AFFINITY: pid 254 tid 269 thread 4 bound to OS proc set 1
OMP: Info #254: KMP_AFFINITY: pid 254 tid 270 thread 5 bound to OS proc set 3
OMP: Info #254: KMP_AFFINITY: pid 254 tid 271 thread 6 bound to OS proc set 5
OMP: Info #254: KMP_AFFINITY: pid 254 tid 272 thread 7 bound to OS proc set 7
OMP: Info #254: KMP_AFFINITY: pid 254 tid 273 thread 8 bound to OS proc set 0
DEEPMD INFO    # number of test data : 40
DEEPMD INFO    Energy MAE         : 4.400922e-03 eV
```

(continues on next page)

(continued from previous page)

```
DEEPMD INFO    Energy RMSE        : 5.258026e-03 eV
DEEPMD INFO    Energy MAE/Natoms  : 8.801843e-04 eV
DEEPMD INFO    Energy RMSE/Natoms : 1.051605e-03 eV
DEEPMD INFO    Force  MAE         : 4.277741e-02 eV/A
DEEPMD INFO    Force  RMSE        : 5.514855e-02 eV/A
DEEPMD INFO    Virial MAE         : 6.080471e-02 eV
DEEPMD INFO    Virial RMSE        : 7.882116e-02 eV
DEEPMD INFO    Virial MAE/Natoms  : 1.216094e-02 eV
DEEPMD INFO    Virial RMSE/Natoms : 1.576423e-02 eV
DEEPMD INFO    # -----------------------------------------------
```

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")
```

```python
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
```

(continues on next page)

(continued from previous page)

```
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-04-20 23:40:36.165078: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.
↪cc:64] Could not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared␣
↪object file: No such file or directory; LD_LIBRARY_PATH: /usr/local/nvidia/lib:/usr/local/nvidia/
↪lib64
2023-04-20 23:40:36.165119: W tensorflow/compiler/xla/stream_executor/cuda/cuda_driver.cc:265]␣
↪failed call to cuInit: UNKNOWN ERROR (303)
2023-04-20 23:40:36.165142: I tensorflow/compiler/xla/stream_executor/cuda/cuda_diagnostics.
↪cc:156] kernel driver does not appear to be running on this host (bohrium-14076-1013950): /proc/
↪driver/nvidia/version does not exist
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.
WARNING:deepmd.utils.batch_size:You can use the environment variable DP_INFER_BATCH_SIZE tocontrol␣
↪the inference batch size (nframes * natoms). The default value is 1024.
```

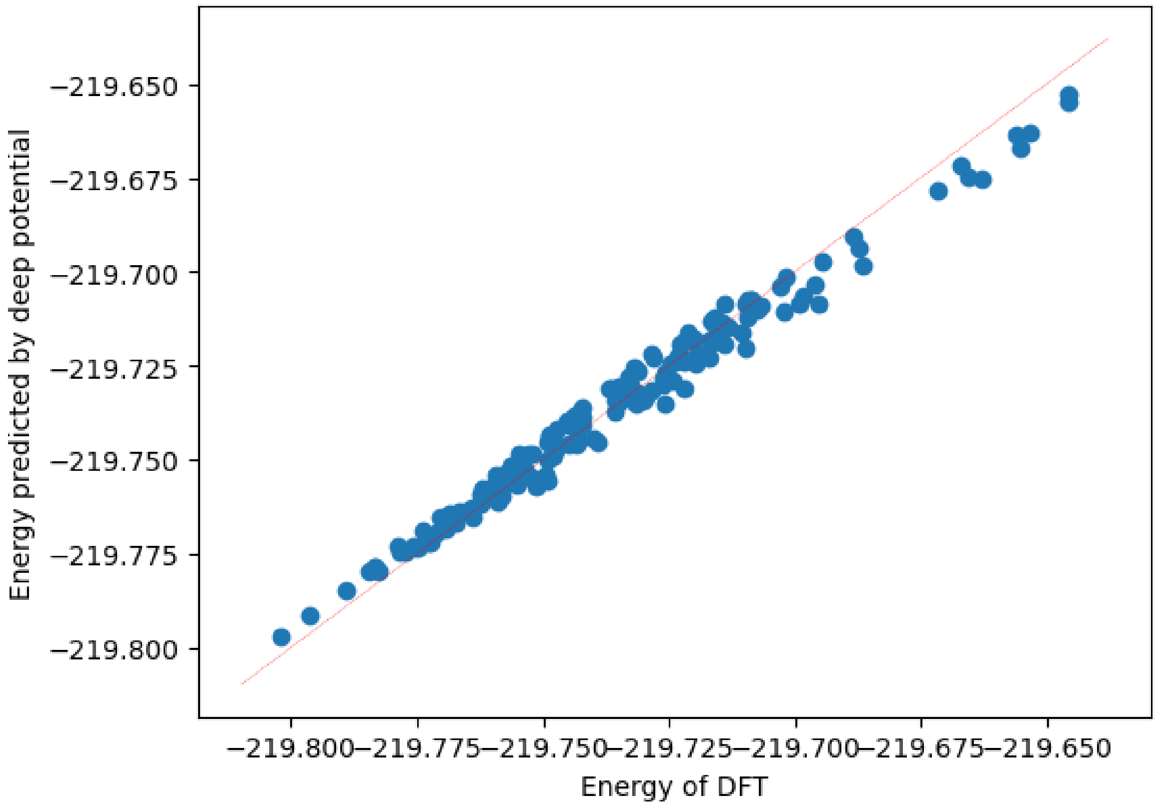
```python
import matplotlib.pyplot as plt
import numpy as np

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.

```
! 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
DeePMD-kit WARNING: Environmental variable OMP_NUM_THREADS is not set. Tune OMP_NUM_THREADS for␣
↪the best performance. See https://deepmd.rtfd.io/parallelism/ for more information.
Summary of lammps deepmd module ...
  >>> Info of deepmd-kit:
  installed to:       /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:
  use deepmd-kit at:  /opt/deepmd-kit-2.2.1DeePMD-kit WARNING: Environmental variable OMP_NUM_
↪THREADS is not set. Tune OMP_NUM_THREADS for the best performance. See https://deepmd.rtfd.io/
↪parallelism/ for more information.
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.
2023-04-20 23:40:39.643206: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could␣
↪not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file:␣
↪No such file or directory; LD_LIBRARY_PATH: /usr/local/nvidia/lib:/usr/local/nvidia/lib64
2023-04-20 23:40:39.643234: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to␣
↪cuInit: UNKNOWN ERROR (303)
2023-04-20 23:40:39.643257: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel␣
↪driver does not appear to be running on this host (bohrium-14076-1013950): /proc/driver/nvidia/
↪version does not exist
2023-04-20 23:40:39.645305: I tensorflow/core/common_runtime/process_util.cc:146] Creating new␣
↪thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads for best␣
↪performance.
2023-04-20 23:40:39.700559: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:354] MLIR␣
↪V1 optimization pass is not enabled
  >>> Info of model(s):
  using   1 model(s): graph.pb
  rcut in model:      6
  ntypes in model:    2
```

(continues on next page)

(continued from previous page)

```
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.

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

Generated 0 of 1 mixed pair_coeff terms from geometric mixing rule
Neighbor list info ...
  update every 10 steps, delay 0 steps, check no
  max neighbors/atom: 2000, page size: 100000
  master list distance cutoff = 7
  ghost atom cutoff = 7
  binsize = 3.5, bins = 3 3 3
  1 neighbor lists, perpetual/occasional/extra = 1 0 0
  (1) pair deepmd, perpetual
      attributes: full, newton on
      pair build: full/bin/atomonly
      stencil: full/bin/3d
      bin: standard
Setting up Verlet run ...
  Unit style    : metal
  Current step  : 0
  Time step     : 0.001
Per MPI rank memory allocation (min/avg/max) = 3.809 | 3.809 | 3.809 Mbytes
   Step         PotEng         KinEng         TotEng          Temp          Press          Volume
         0  -219.77011      0.025852029   -219.74426      50            -810.10259      1060.5429
       100  -219.76784      0.023303362   -219.74454      45.070664     -605.50113      1060.5429
       200  -219.77863      0.032400378   -219.74622      62.665059     -53.929107      1060.5429
       300  -219.77403      0.027115352   -219.74692      52.443373      642.24342      1060.5429
       400  -219.77126      0.023079501   -219.74818      44.637697      861.365        1060.5429
       500  -219.786        0.034433001   -219.75156      66.596322      256.47994      1060.5429
       600  -219.78295      0.029039598   -219.75391      56.165027     -527.21506      1060.5429
       700  -219.777        0.020227709   -219.75677      39.122091     -696.11258      1060.5429
       800  -219.78394      0.022893217   -219.76105      44.277408     -77.227892      1060.5429
       900  -219.77998      0.015506508   -219.76447      29.990893      663.84491      1060.5429
      1000  -219.78328      0.015178419   -219.7681       29.356341      482.8228       1060.5429
      1100  -219.7903       0.018763273   -219.77154      36.28975      -273.19351      1060.5429
      1200  -219.78639      0.012922048   -219.77347      24.992328     -577.90459      1060.5429
      1300  -219.79131      0.015848131   -219.77546      30.65162      -129.85247      1060.5429
      1400  -219.78829      0.011969602   -219.77632      23.150218      545.58517      1060.5429
      1500  -219.78735      0.010610097   -219.77674      20.520821      356.36805      1060.5429
      1600  -219.78834      0.011547453   -219.77679      22.333746     -386.08305      1060.5429
      1700  -219.78549      0.0095297364  -219.77596      18.431312     -522.29867      1060.5429
      1800  -219.78787      0.01302987    -219.77484      25.200865      120.83085      1060.5429
      1900  -219.7845       0.012341623   -219.77216      23.869737      643.66442      1060.5429
      2000  -219.7857       0.017597987   -219.76811      34.035987      255.57892      1060.5429
      2100  -219.7853       0.023253088   -219.76205      44.973429     -465.61243      1060.5429
      2200  -219.77987      0.024650089   -219.75522      47.675348     -708.62743      1060.5429
      2300  -219.78134      0.030690759   -219.75065      59.358512     -221.82549      1060.5429
      2400  -219.77737      0.029446857   -219.74792      56.952699      635.02431      1060.5429
      2500  -219.768        0.022122766   -219.74587      42.787292      826.89652      1060.5429
      2600  -219.77246      0.02691536    -219.74554      52.056572      168.88834      1060.5429
      2700  -219.77746      0.031963987   -219.7455       61.821042     -497.33107      1060.5429
      2800  -219.7733       0.02814671    -219.74515      54.438107     -792.71093      1060.5429
```

(continues on next page)

(continued from previous page)

```
      2900  -219.77498      0.029131114   -219.74585      56.342026     -685.23164      1060.5429
      3000  -219.78212      0.034326288   -219.74779      66.38993      -20.441816      1060.5429
      3100  -219.77222      0.02366469    -219.74856      45.769502      708.42782      1060.5429
      3200  -219.77252      0.022334468   -219.75019      43.196742      753.3138       1060.5429
      3300  -219.78538      0.032458098   -219.75292      62.776693      36.172647      1060.5429
      3400  -219.78047      0.026131264   -219.75434      50.540064     -661.25487      1060.5429
      3500  -219.77926      0.022926821   -219.75633      44.342401     -623.5037       1060.5429
      3600  -219.78369      0.024854728   -219.75884      48.071137      74.821258      1060.5429
      3700  -219.7768       0.016731114   -219.76006      32.359382      709.57785      1060.5429
      3800  -219.77927      0.017595175   -219.76168      34.03055       543.56168      1060.5429
      3900  -219.7864       0.023003584   -219.7634       44.490867     -230.55364      1060.5429
      4000  -219.78098      0.017102387   -219.76388      33.077456     -677.85161      1060.5429
      4100  -219.78581      0.020907229   -219.7649       40.436341     -343.9622       1060.5429
      4200  -219.78717      0.021708329   -219.76546      41.985736      491.95578      1060.5429
      4300  -219.78328      0.018229256   -219.76505      35.256916      680.5279       1060.5429
      4400  -219.79007      0.024931071   -219.76514      48.21879      -26.785455      1060.5429
      4500  -219.78331      0.019795452   -219.76352      38.286071     -624.98799      1060.5429
      4600  -219.78094      0.0196038     -219.76134      37.915399     -584.8297       1060.5429
      4700  -219.78608      0.027516802   -219.75857      53.219812      74.218844      1060.5429
      4800  -219.77656      0.023488867   -219.75307      45.429446      827.4406       1060.5429
      4900  -219.78039      0.032832529   -219.74755      63.500874      634.64896      1060.5429
      5000  -219.78237      0.040761952   -219.7416       78.837046     -224.81626      1060.5429
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
Neigh   | 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   |            |       |  0.05

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

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

CHAPTER

TWO

INSTALLATION

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: 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 splited 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:

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

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

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

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

One may specify the DeePMD-kit version such as `2.1.1` 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 lammps -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:

```
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/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 11 supported:

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

`cu11` is required only when CUDA Toolkit and cuDNN were not installed.

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,cu11,lmp,ipi]
```

MPICH is required for parallel running.

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 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 --recursive https://github.com/deepmodeling/deepmd-kit.git deepmd-kit
```

The `--recursive` option clones all submodules needed by 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. 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

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

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

Environment variables	Allowed value	Default value	Usage
DP_VARIANT	`cpu`, `cuda`, `rocm`	`cpu`	Build CPU variant or GPU variant with CUDA or ROCM support.
CUDA_TOOLKIT_ROOT_DIR	Path	Detected automatically	The path to the CUDA toolkit directory. CUDA 7.0 or later is supported. NVCC is required.
ROCM_ROOT	Path	Detected automatically	The path to the ROCM toolkit directory.
TENSORFLOW_ROOT	Path	Detected automatically	The path to TensorFlow Python library. By default the installer only finds TensorFlow under user site-package directory (`site.getusersitepackages()`) or system site-package directory (`sysconfig.get_path("purelib")`) due to limitation of PEP-517. If not found, the latest TensorFlow (or the environment variable `TENSORFLOW_VERSION` if given) from PyPI will be built against.
DP_ENABLE_NATIVE_OPTIMIZATION	0, 1	0	Enable compilation optimization for the native machine's CPU type. Do not enable it if generated code will run on different CPUs.

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:
```

(continues on next page)

(continued from previous page)

```
  {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
```

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`:

CMake Aurgements	Allowed value	Default value	Usage
-DTENSORFLOW_ROOT=<value>	Path	-	The Path to TensorFlow's C++ interface.
-DCMAKE_INSTALL_PREFIX=<value>	Path	-	The Path where DeePMD-kit will be installed.
-DUSE_CUDA_TOOLKIT=<value>	`TRUE` or `FALSE`	`FALSE`	If `TRUE`, Build GPU support with CUDA toolkit.
-DCUDA_TOOLKIT_ROOT_DIR=<value>	Path	Detected automatically	The path to the CUDA toolkit directory. CUDA 7.0 or later is supported. NVCC is required.
-DUSE_ROCM_TOOLKIT=<value>	`TRUE` or `FALSE`	`FALSE`	If `TRUE`, Build GPU support with ROCM toolkit.
-DCMAKE_HIP_COMPILER_ROCM_ROOT=<value>	Path	Detected automatically	The path to the ROCM toolkit directory.
-DLAMMPS_SOURCE_ROOT=<value>	Path	-	Only neccessary for LAMMPS plugin mode. The path to the LAMMPS source code. LAMMPS 8Apr2021 or later is supported. If not assigned, the plugin mode will not be enabled.
-DUSE_TF_PYTHON_LIBS=<value>	`TRUE` or `FALSE`	`FALSE`	If `TRUE`, Build C++ interface with TensorFlow's Python libraries(TensorFlow's Python Interface is required). And there's no need for building TensorFlow's C++ interface.
-DENABLE_NATIVE_OPTIMIZATION	`TRUE` or `FALSE`	`FALSE`	Enable compilation optimization for the native machine's CPU type. Do not enable it if generated code will run on different CPUs.

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

```
make -j4
make install
```

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

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

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

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.

The library is built in Linux (GLIBC 2.17) with CUDA 11.8. 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 $deepmd_source_dir/source
mkdir build
cd build
cmake -DDEEPMD_C_ROOT=/path/to/libdeepmd_c -DCMAKE_INSTALL_PREFIX=$deepmd_root ..
make -j8
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 be installed.

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

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

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

```
cd /some/workspace
wget https://github.com/lammps/lammps/archive/stable_23Jun2022_update4.tar.gz
tar xf stable_23Jun2022_update4.tar.gz
```

The source code of LAMMPS is stored in the directory `lammps-stable_23Jun2022_update4`. Now go into the LAMMPS code and copy the DeePMD-kit module like this

```
cd lammps-stable_23Jun2022_update4/src/
cp -r $deepmd_source_dir/source/build/USER-DEEPMD .
make yes-kspace
make yes-extra-fix
make yes-user-deepmd
```

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

```
make mpi -j4
```

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

```
./lmp_mpi -h
```

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

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

2.4.2 Install LAMMPS (plugin mode)

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

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

```
cd /some/workspace
wget https://github.com/lammps/lammps/archive/stable_23Jun2022_update4.tar.gz
tar xf stable_23Jun2022_update4.tar.gz
```

The source code of LAMMPS is stored in the directory `lammps-stable_23Jun2022_update4`. 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_23Jun2022_update4/build/
cd lammps-stable_23Jun2022_update4/build/
```

Now build LAMMPS. Note that `PLUGIN` and `KSPACE` packages 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 PKG_KSPACE=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,

```
patchelf --set-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:

```
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:

```
#!/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
```

(continues on next page)

(continued from previous page)

```
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
          -DCUDA_TOOLKIT_ROOT_DIR=/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

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

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

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

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

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

CHAPTER

THREE

DATA

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

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

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_atten`.

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

ID	Property	Raw file	Required/Optional	Shape	Description
type	Atom type indexes	type.raw	Required	Natoms	Integers that start with 0. If both the training parameter type_map is set and `type_map.raw` is provided, the system atom type should be mapped to `type_map.raw` in `type.raw` and will be mapped to the model atom type when training; otherwise, the system atom type will be always mapped to the model atom type (whether type_map is set or not)
type_map	Atom type names	type_map.raw	Optional	Ntypes	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
nopbc	Non-periodic system	nopbc	Optional	1	If True, this system is non-periodic; otherwise it's periodic

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

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

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

ID	Property	Raw file	Unit	Shape	Description
energy	Frame energies	energy.raw	eV	Nframes	
force	Atomic forces	force.raw	eV/Å	Nframes * Natoms * 3	
virial	Frame virial	virial.raw	eV	Nframes * 9	in the order `XX XY XZ YX YY YZ ZX ZY ZZ`
atom_ener	Atomic energies	atom_ener.raw	eV	Nframes * Natoms	
atom_pref	Weights of atomic forces	atom_pref.raw	1	Nframes * Natoms	
dipole	Frame dipole	dipole.raw	Any	Nframes * 3	
atomic_dipole	Atomic dipole	atomic_dipole.raw	Any	Nframes * Natoms * 3	
polarizability	Frame polarizability	polarizability.raw	Any	Nframes * 9	in the order `XX XY XZ YX YY YZ ZX ZY ZZ`
atomic_polarizability	Atomic polarizability	atomic_polarizability.raw	Any	Nframes * Natoms * 9	in the order `XX XY XZ YX YY YZ ZX ZY ZZ`

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

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

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
set.000/coord.npy
set.000/energy.npy
set.000/force.npy
set.001/coord.npy
set.001/energy.npy
set.001/force.npy
```

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

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

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

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

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

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

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
nline per set is 2000
will make 3 sets
making set 0 ...
making set 1 ...
making set 2 ...
$ ls
box.raw  coord.raw  energy.raw  force.raw  set.000  set.001  set.002  type.raw  virial.raw
```

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

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

```
import dpdata

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

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

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

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

CHAPTER

FOUR

MODEL

4.1 Overall

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

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

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

DeePMD-kit implements the following descriptors:

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

The fitting of the following physical properties is supported

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

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).

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

```
        "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^2_{types} 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.

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

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

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

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

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

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 angles between two neighboring atoms as input (denoted by `e3`).

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

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

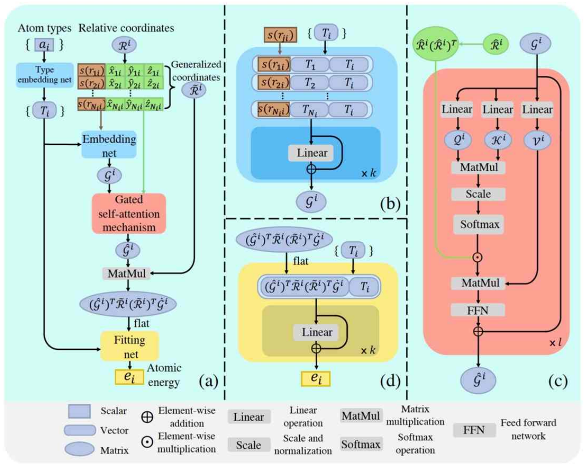
The training input script is very similar to that of *se_e2_a*. The only difference lies in the `descriptor <model/descriptor>` section

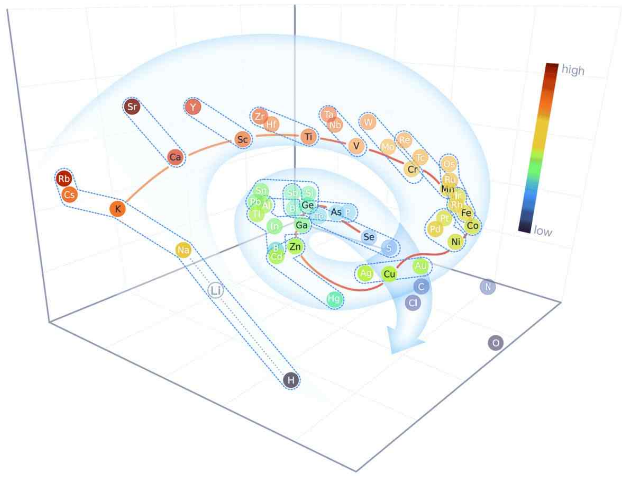
```
        "descriptor": {
            "type":                 "se_e3",
            "sel":                 [40, 80],
            "rcut_smth":         0.50,
            "rcut":                 6.00,
            "neuron":                 [2, 4, 8],
            "resnet_dt":         false,
            "seed":                 1,
            "_comment":                 " that's all"
        },
```

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

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 Installation

Follow the standard installation of Python interface in the DeePMD-kit. After that, you can smoothly use the DPA-1 model with the following instructions.

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:

```
$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

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

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

Fitting "ener"

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

Type embedding

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

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

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

Type map

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

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

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

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:

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

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

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

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

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 $\mathcal{D}_1$, $\mathcal{D}_2$, ..., $\mathcal{D}_N$, the hybrid descriptor this the concatenation of the list, i.e. $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_N)$.

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

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

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

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

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 be not conserved, making the accuracy of the model worse.

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

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

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 The fitting network

The construction of the fitting net is given by section fitting_net

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

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

4.8.2 Loss

The loss function L for training energy is given by

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

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

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

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

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

The loss section in the `input.json` is

```
    "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 give 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, magnatic force and virial, respectively. p_e, p_{fr}, p_{fm} and p_v give the prefactors of the energy, atomic force, magnatic force and virial losses.

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

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

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

```
pref_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, magnatic force and virial, respectively.

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

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

```
$deepmd_source_dir/examples/water_tensor/dipole/dipole_input.json
$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 The fitting Network

The fitting_net section tells DP which fitting net to use.

The JSON of dipole type should be provided like

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

The JSON of polar type should be provided like

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

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

4.10.2 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, YX, YY, YZ, XZ, ZY, ZZ. By contrast, in an atomic system, each atom in sel_type has a tensor label. For example, when fitting a dipole, each frame will provide a #sel_atom x 3 matrices, where #sel_atom is the number of atoms whose type are in sel_type.

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

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

The loss section should be provided like

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

- type should be written as tensor as a distinction from ener mode.

- pref and pref_atomic respectively specify the weight of global loss and atomic loss. It can not be left unset. If set to 0, the corresponding label will NOT be included in the training process.

4.10.3 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.

4.10.4 Train the Model

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

```
dp train input.json
```

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

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

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

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

(continues on next page)

(continued from previous page)

```
                >atomic_system
                >global_system
```

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

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 x numb_dos` vector which gives the total electronic density of states. By contrast, in an atomic system, each atom in has a `atom_dos` label. For example, when fitting the site-projected electronic density of states, each frame will provide a `natom x numb_dos` matrices,

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

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

The loss section should be provided like

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

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

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

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.

```
dp train input.json
```

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

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

(continues on next page)

(continued from previous page)

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

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
0.000000000000000000e+00 1.963193264917645342e-03
0.000000000000000000e+00 1.178440836781313727e-03
0.000000000000000000e+00 1.441258071790407769e-04
0.000000000000000000e+00 1.787297933314058174e-03
0.000000000000000000e+00 1.901603280243024940e-03
0.000000000000000000e+00 2.279848925571981155e-03
0.000000000000000000e+00 2.149355854688561607e-03
0.000000000000000000e+00 1.829848459515726056e-03
0.000000000000000000e+00 1.905156512419792225e-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 Type embedding net

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

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

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

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

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

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

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

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

See here for further explanation of `type embedding`.

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

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

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

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

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

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

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

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

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

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

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

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

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

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 Train a deep Wannier model for Wannier centroids

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

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

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

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

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

```
    "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
```

4.14.2 Train the DPLR model

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

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

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

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

4.14.3 Molecular dynamics simulation with DPLR

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

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

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

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

```


512 atoms
3 atom types
128 bonds
1 bond types

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

Masses

1 16
2 2
3 16

Atoms

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

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

Bonds

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

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

An example input script is provided in

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

Here are some explanations

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

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

Type 1 and 2 (O and H) are `real_atoms`, while type 3 (WCs) are `virtual_atoms`. The model file `ener.pb` stores both the DW and DPLR models, so the position of WCs and the energy can be inferred from it. A virtual bond type is specified by `bond_style zero`. The `special_bonds` command switches off the exclusion of intramolecular interactions.

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

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

```
# "fix dplr" 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. At this time, the training parameter type_map will be mapped to LAMMPS atom types.

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

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

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

The LAMMPS simulation can be started from the example directory by

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

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

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

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

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:

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

See the JCTC paper for details.

4.15.1 Training data

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

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

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

4.15.2 Training the DPRc model

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

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

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

```
"descriptor" :{
    "type":             "hybrid",
    "list" : [
        {
            "type":     "se_e2_a",
            "sel":              [6, 11, 0, 6, 0, 1],
            "rcut_smth":        1.00,
            "rcut":             9.00,
            "neuron":           [12, 25, 50],
            "exclude_types":    [[2, 2], [2, 4], [4, 4], [0, 2], [0, 4], [1, 2], [1, 4], [3, 2],␣
↪[3, 4], [5, 2], [5, 4]],
```

(continues on next page)

(continued from previous page)

```
            "axis_neuron":      12,
            "set_davg_zero":    true,
            "_comment": " QM/QM interaction"
        },
        {
            "type":     "se_e2_a",
            "sel":              [6, 11, 100, 6, 50, 1],
            "rcut_smth":        0.50,
            "rcut":             6.00,
            "neuron":           [12, 25, 50],
            "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:

```
from itertools import combinations_with_replacement, product
qm = (0, 1, 3, 5)
mm = (2, 4)
print("QM/QM:", list(map(list, list(combinations_with_replacement(mm, 2)) + list(product(qm,␣
↪mm)))))
print("QM/MM:", list(map(list, list(combinations_with_replacement(qm, 2)) + list(combinations_with_
↪replacement(mm, 2)))))
```

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

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

Note that atom_ener only works when descriptor/set_davg_zero is `true`.

4.15.3 Run MD simulations

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

CHAPTER

FIVE

TRAINING

5.1 Train a model

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

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

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

```
$ dp train input.json
```

where `input.json` is the name of the input script.

By default, the verbosity level of the DeePMD-kit is `INFO`, one may see a lot of important information on the code and environment showing on the screen. Among them two pieces of information regarding data systems are worth special notice.

```
DEEPMD INFO    ---Summary of DataSystem: training     ---------------------------------------------
↪--
DEEPMD INFO    found 3 system(s):
DEEPMD INFO                                    system  natoms  bch_sz   n_bch   prob  pbc
DEEPMD INFO                      ../data_water/data_0/     192       1      80  0.250    T
DEEPMD INFO                      ../data_water/data_1/     192       1     160  0.500    T
DEEPMD INFO                      ../data_water/data_2/     192       1      80  0.250    T
DEEPMD INFO    --------------------------------------------------------------------------------------
↪--
DEEPMD INFO    ---Summary of DataSystem: validation   ---------------------------------------------
↪--
DEEPMD INFO    found 1 system(s):
DEEPMD INFO                                    system  natoms  bch_sz   n_bch   prob  pbc
DEEPMD INFO                       ../data_water/data_3     192       1      80  1.000    T
DEEPMD INFO    --------------------------------------------------------------------------------------
↪--
```

The DeePMD-kit prints detailed information on the training and validation data sets. The data sets are defined by training_data and validation_data defined in the training section of the input script. The training data set is composed of three data systems, while the validation data set is composed by one data system. The number of atoms, batch size, the number of batches in the system and the probability of using the system are all shown on the screen. The last column presents if the periodic boundary condition is assumed for the system.

During the training, the error of the model is tested every disp_freq training steps with the batch used to train the model and with numb_btch batches from the validating data. The training error and validation error are printed correspondingly in the file disp_file (default is `lcurve.out`). The batch size can be set in the input script by the key batch_size in the corresponding sections for the training and validation data set. An example of the output

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

The file contains 8 columns, from left to right, which are the training step, the validation loss, training loss, root mean square (RMS) validation error of energy, RMS training error of energy, RMS validation error of force, RMS training error of force and the learning rate. The RMS error (RMSE) of the energy is normalized by the number of atoms in the system. One can visualize this file with a simple Python script:

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

data = np.genfromtxt("lcurve.out", names=True)
for name in data.dtype.names[1:-1]:
    plt.plot(data['step'], data[name], label=name)
plt.legend()
plt.xlabel('Step')
plt.ylabel('Loss')
plt.xscale('symlog')
plt.yscale('log')
plt.grid()
plt.show()
```

Checkpoints will be written to files with the prefix save_ckpt every save_freq training steps.

Warning: It is warned that the example water data (in folder `examples/water/data`) is of very limited amount, is provided only for testing purposes, and should not be used to train a production model.

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

The learning_rate section in `input.json` is given as follows

```
    "learning_rate" :{
        "type":                 "exp",
        "start_lr":         0.001,
        "stop_lr":        3.51e-8,
        "decay_steps":        5000,
        "_comment":         "that's all"
    }
```

- start_lr gives the learning rate at the beginning of the training.

- stop_lr gives the learning rate at the end of the training. It should be small enough to ensure that the network parameters satisfactorily converge.
- During the training, the learning rate decays exponentially from start_lr to stop_lr following the formula:

$$\alpha(t) = \alpha_0 \lambda^{t/\tau}$$

where t is the training step, α is the learning rate, α_0 is the starting learning rate (set by start_lr), λ is the decay rate, and τ is the decay steps, i.e.

```
```
lr(t) = start_lr * decay_rate ^ (t / decay_steps)
```
```

5.2.2 Training parameters

Other training parameters are given in the training section.

```
    "training": {
         "training_data": {
            "systems":                ["../data_water/data_0/", "../data_water/data_1/", "../data_
↪water/data_2/"],
            "batch_size":        "auto"
        },
        "validation_data":{
            "systems":                ["../data_water/data_3"],
            "batch_size":        1,
            "numb_btch":        3
        },
        "mixed_precision": {
            "output_prec":      "float32",
            "compute_prec":     "float16"
        },

        "numb_steps":        1000000,
        "seed":                1,
        "disp_file":        "lcurve.out",
        "disp_freq":        100,
        "save_freq":        1000
    }
```

The sections training_data and validation_data give the training dataset and validation dataset, respectively. Taking the training dataset for example, the keys are explained below:

- systems provide paths of the training data systems. DeePMD-kit allows you to provide multiple systems with different numbers of atoms. This key can be a `list` or a `str`.
  - `list`: systems gives the training data systems.
  - `str`: systems should be a valid path. DeePMD-kit will recursively search all data systems in this path.
- At each training step, DeePMD-kit randomly picks batch_size frame(s) from one of the systems. The probability of using a system is by default in proportion to the number of batches in the system. More options are available for automatically determining the probability of using systems. One can set the key auto_prob to
  - `"prob_uniform"` all systems are used with the same probability.

  - "prob_sys_size" the probability of using a system is proportional to its size (number of frames).
  - "prob_sys_size; sidx_0:eidx_0:w_0; sidx_1:eidx_1:w_1;..." the list of systems is divided into blocks. Block i has systems ranging from sidx_i to eidx_i. The probability of using a system from block i is proportional to w_i. Within one block, the probability of using a system is proportional to its size.

- An example of using "auto_prob" is given below. The probability of using systems[2] is 0.4, and the sum of the probabilities of using systems[0] and systems[1] is 0.6. If the number of frames in systems[1] is twice of system[0], then the probability of using system[1] is 0.4 and that of system[0] is 0.2.

```
        "training_data": {
            "systems":                ["../data_water/data_0/", "../data_water/data_1/", "../data_
↪water/data_2/"],
            "auto_prob":        "prob_sys_size; 0:2:0.6; 2:3:0.4",
            "batch_size":        "auto"
        }
```

- The probability of using systems can also be specified explicitly with key sys_probs which is a list having the length of the number of systems. For example

```
        "training_data": {
            "systems":                ["../data_water/data_0/", "../data_water/data_1/", "../data_
↪water/data_2/"],
            "sys_probs":        [0.5, 0.3, 0.2],
            "batch_size":        "auto:32"
        }
```

- The key batch_size specifies the number of frames used to train or validate the model in a training step. It can be set to
  - list: the length of which is the same as the systems. The batch size of each system is given by the elements of the list.
  - int: all systems use the same batch size.
  - "auto": the same as "auto:32", see "auto:N"
  - "auto:N": automatically determines the batch size so that the batch_size times the number of atoms in the system is no less than N.
- The key numb_batch in validate_data gives the number of batches of model validation. Note that the batches may not be from the same system

The section mixed_precision specifies the mixed precision settings, which will enable the mixed precision training workflow for DeePMD-kit. The keys are explained below:

- output_prec precision used in the output tensors, only float32 is supported currently.
- compute_prec precision used in the computing tensors, only float16 is supported currently. Note there are several limitations about mixed precision training:
- Only se_e2_a type descriptor is supported by the mixed precision training workflow.
- The precision of the embedding net and the fitting net are forced to be set to float32.

Other keys in the training section are explained below:

- numb_steps The number of training steps.
- seed The random seed for getting frames from the training data set.

- disp_file The file for printing learning curve.
- disp_freq The frequency of printing learning curve. Set in the unit of training steps
- save_freq The frequency of saving checkpoint.

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

```
positional arguments:
  INPUT                 the input json database

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

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

  --restart RESTART     Restart the training from the provided checkpoint

  --init-frz-model INIT_FRZ_MODEL
                        Initialize the training from the frozen model.
  --skip-neighbor-stat  Skip calculating neighbor statistics. Sel checking, automatic sel, and␣
↪model compression will be disabled. (default: False)
```

`--init-model model.ckpt`, initializes the model training with an existing model that is stored in the checkpoint `model.ckpt`, the network architectures should match.

`--restart model.ckpt`, continues the training from the checkpoint `model.ckpt`.

`--init-frz-model frozen_model.pb`, initializes the training with an existing model that is stored in `frozen_model.pb`.

`--skip-neighbor-stat` will skip calculating neighbor statistics if one is concerned about performance. Some features will be disabled.

To maximize the performance, one should follow FAQ: How to control the parallelism of a job to control the number of threads.

One can set other environmental variables:

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

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. All training parameters below can be set in DP-GUI. By clicking "SAVE JSON", one can download the input file for furthur training.

`model`:

type: `dict`
argument path: `model`

`type_map`:

type: `list`, optional
argument path: `model/type_map`

A list of strings. Give the name to each type of atoms. It is noted that the number of atom type of training system must be less than 128 in a GPU environment. If not given, type.raw in each system should use the same type indexes, and type_map.raw will take no effect.

`data_stat_nbatch`:

type: `int`, optional, default: `10`
argument path: `model/data_stat_nbatch`

The model determines the normalization from the statistics of the data. This key specifies the number of frames in each system used for statistics.

data_stat_protect:

type: `float`, optional, default: `0.01`
argument path: `model/data_stat_protect`

Protect parameter for atomic energy regression.

data_bias_nsample:

type: `int`, optional, default: `10`
argument path: `model/data_bias_nsample`

The number of training samples in a system to compute and change the energy bias.

use_srtab:

type: `str`, optional
argument path: `model/use_srtab`

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

smin_alpha:

type: `float`, optional
argument path: `model/smin_alpha`

The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when use_srtab is provided.

sw_rmin:

type: `float`, optional
argument path: `model/sw_rmin`

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.

type_embedding:

type: `dict`, optional
argument path: `model/type_embedding`

The type embedding.

neuron:

type: `list`, 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

descriptor:

type: `dict`

argument path: `model/descriptor`

The descriptor of atomic environment.

Depending on the value of type, different sub args are accepted.

type:

type: `str` (flag key)

argument path: `model/descriptor/type`

possible choices: *loc_frame*, *se_e2_a*, *se_e3*, *se_a_tpe*, *se_e2_r*, *hybrid*, *se_atten*, *se_a_mask*

The type of the descritpor. See explanation below.

- loc_frame: Defines a local frame at each atom, and the compute the descriptor as local coordinates under this frame.
- se_e2_a: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor.
- se_e2_r: Used by the smooth edition of Deep Potential. Only the distance between atoms is used to construct the descriptor.

- se_e3: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Three-body embedding will be used by this descriptor.
- se_a_tpe: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Type embedding will be used by this descriptor.
- se_atten: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Attention mechanism will be used by this descriptor.
- se_a_mask: Used by the smooth edition of Deep Potential. It can accept a variable number of atoms in a frame (Non-PBC system). aparam are required as an indicator matrix for the real/virtual sign of input atoms.
- hybrid: Concatenate of a list of descriptors as a new descriptor.

When type is set to `loc_frame`:

`sel_a`:

type: `list`

argument path: `model/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: `list`

argument path: `model/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/descriptor[loc_frame]/rcut`

The cut-off radius. The default value is 6.0

`axis_rule`:

type: `list`

argument path: `model/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` | `list`, optional, default: `auto`

argument path: `model/descriptor[se_e2_a]/sel`

This parameter set the number of selected neighbors for each type of atom. It can be:

- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. sel[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- str. Can be "auto:factor" or "auto". "factor" is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the "factor". Finally the number is wraped up to 4 divisible. The option "auto" is equivalent to "auto:1.1".

`rcut:`

type: `float`, optional, default: `6.0`

argument path: `model/descriptor[se_e2_a]/rcut`

The cut-off radius.

`rcut_smth:`

type: `float`, optional, default: `0.5`

argument path: `model/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: `list`, optional, default: `[10, 20, 40]`

argument path: `model/descriptor[se_e2_a]/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

`axis_neuron:`

type: `int`, optional, default: `4`, alias: n_axis_neuron

argument path: `model/descriptor[se_e2_a]/axis_neuron`

Size of the submatrix of G (embedding matrix).

`activation_function:`

type: `str`, optional, default: `tanh`

argument path: `model/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/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/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/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/descriptor[se_e2_a]/trainable`

If the parameters in the embedding net is trainable

`seed:`

type: `NoneType` | `int`, optional
argument path: `model/descriptor[se_e2_a]/seed`

Random seed for parameter initialization

`exclude_types:`

type: `list`, optional, default: `[]`
argument path: `model/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/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` | `list`, optional, default: `auto` argument path: `model/descriptor[se_e3]/sel`

This parameter set the number of selected neighbors for each type of atom. It can be:

- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. sel[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- str. Can be "auto:factor" or "auto". "factor" is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the "factor". Finally the number is wraped up to 4 divisible. The option "auto" is equivalent to "auto:1.1".

`rcut:`

type: `float`, optional, default: `6.0`

argument path: `model/descriptor[se_e3]/rcut`

The cut-off radius.

`rcut_smth:`

type: `float`, optional, default: `0.5`

argument path: `model/descriptor[se_e3]/rcut_smth`

Where to start smoothing. For example the 1/r term is smoothed from rcut to rcut_smth

`neuron:`

type: `list`, optional, default: `[10, 20, 40]`

argument path: `model/descriptor[se_e3]/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

`activation_function:`

type: `str`, optional, default: `tanh`

argument path: `model/descriptor[se_e3]/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/descriptor[se_e3]/resnet_dt`

Whether to use a "Timestep" in the skip connection

`precision:`

type: `str`, optional, default: `default`

argument path: `model/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/descriptor[se_e3]/trainable`

If the parameters in the embedding net are trainable

`seed:`

type: `NoneType` | `int`, optional

argument path: `model/descriptor[se_e3]/seed`

Random seed for parameter initialization

`set_davg_zero:`

type: `bool`, optional, default: `False`

argument path: `model/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` | `list`, optional, default: `auto`

argument path: `model/descriptor[se_a_tpe]/sel`

This parameter set the number of selected neighbors for each type of atom. It can be:

- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. sel[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- str. Can be "auto:factor" or "auto". "factor" is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the "factor". Finally the number is wraped up to 4 divisible. The option "auto" is equivalent to "auto:1.1".

`rcut:`

type: `float`, optional, default: `6.0`

argument path: `model/descriptor[se_a_tpe]/rcut`

The cut-off radius.

`rcut_smth:`

type: `float`, optional, default: `0.5`

argument path: `model/descriptor[se_a_tpe]/rcut_smth`

Where to start smoothing. For example the 1/r term is smoothed from rcut to rcut_smth

`neuron:`

type: `list`, optional, default: `[10, 20, 40]` argument path: `model/descriptor[se_a_tpe]/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

`axis_neuron:`

type: `int`, optional, default: `4`, alias: n_axis_neuron

argument path: `model/descriptor[se_a_tpe]/axis_neuron`

Size of the submatrix of G (embedding matrix).

`activation_function:`

type: `str`, optional, default: `tanh`

argument path: `model/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/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/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/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/descriptor[se_a_tpe]/trainable`

If the parameters in the embedding net is trainable

`seed:`

type: `NoneType` | `int`, optional

argument path: `model/descriptor[se_a_tpe]/seed`

Random seed for parameter initialization

`exclude_types`:

type: `list`, optional, default: `[]`
argument path: `model/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/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/descriptor[se_a_tpe]/type_nchanl`

number of channels for type embedding

`type_nlayer`:

type: `int`, optional, default: `2`
argument path: `model/descriptor[se_a_tpe]/type_nlayer`

number of hidden layers of type embedding net

`numb_aparam`:

type: `int`, optional, default: `0`
argument path: `model/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` | `list`, optional, default: `auto`
argument path: `model/descriptor[se_e2_r]/sel`

This parameter set the number of selected neighbors for each type of atom. It can be:

- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. sel[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- str. Can be "auto:factor" or "auto". "factor" is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the "factor". Finally the number is wraped up to 4 divisible. The option "auto" is equivalent to "auto:1.1".

`rcut`:

type: `float`, optional, default: `6.0`
argument path: `model/descriptor[se_e2_r]/rcut`

The cut-off radius.

`rcut_smth:`

type: `float`, optional, default: `0.5`

argument path: `model/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: `list`, optional, default: `[10, 20, 40]`

argument path: `model/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/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/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/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/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/descriptor[se_e2_r]/trainable`

If the parameters in the embedding net are trainable

`seed:`

type: `NoneType` | `int`, optional

argument path: `model/descriptor[se_e2_r]/seed`

Random seed for parameter initialization

`exclude_types:`

type: `list`, optional, default: `[]`
argument path: `model/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/descriptor[se_e2_r]/set_davg_zero`

Set the normalization average to zero. This option should be set when atom_ener in the energy fitting is used

When type is set to `hybrid`:

`list:`

type: `list`
argument path: `model/descriptor[hybrid]/list`

A list of descriptor definitions

When type is set to `se_atten`:

`sel:`

type: `list` | `str` | `int`, optional, default: `auto`
argument path: `model/descriptor[se_atten]/sel`

This parameter set the number of selected neighbors. Note that this parameter is a little different from that in other descriptors. Instead of separating each type of atoms, only the summation matters. And this number is highly related with the efficiency, thus one should not make it too large. Usually 200 or less is enough, far away from the GPU limitation 4096. It can be:

- int. The maximum number of neighbor atoms to be considered. We recommend it to be less than 200.
- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. Only the summation of sel[i] matters, and it is recommended to be less than 200. - str. Can be "auto:factor" or "auto". "factor" is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the "factor". Finally the number is wraped up to 4 divisible. The option "auto" is equivalent to "auto:1.1".

`rcut:`

type: `float`, optional, default: `6.0`
argument path: `model/descriptor[se_atten]/rcut`

The cut-off radius.

`rcut_smth:`

type: `float`, optional, default: `0.5`

argument path: `model/descriptor[se_atten]/rcut_smth`

Where to start smoothing. For example the 1/r term is smoothed from rcut to rcut_smth

`neuron:`

type: `list`, optional, default: `[10, 20, 40]`

argument path: `model/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/descriptor[se_atten]/axis_neuron`

Size of the submatrix of G (embedding matrix).

`activation_function:`

type: `str`, optional, default: `tanh`

argument path: `model/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/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/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/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/descriptor[se_atten]/trainable`

If the parameters in the embedding net is trainable

seed:

type: `NoneType` | `int`, optional
argument path: `model/descriptor[se_atten]/seed`

Random seed for parameter initialization

exclude_types:

type: `list`, optional, default: `[]`
argument path: `model/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.

set_davg_zero:

type: `bool`, optional, default: `True`
argument path: `model/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

attn:

type: `int`, optional, default: `128`
argument path: `model/descriptor[se_atten]/attn`

The length of hidden vectors in attention layers

attn_layer:

type: `int`, optional, default: `2`
argument path: `model/descriptor[se_atten]/attn_layer`

The number of attention layers

attn_dotr:

type: `bool`, optional, default: `True`
argument path: `model/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/descriptor[se_atten]/attn_mask`

Whether to do mask on the diagonal in the attention matrix

When type is set to `se_a_mask`:

sel:

type: `str` | `list`, optional, default: `auto`
argument path: `model/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: `list`, optional, default: `[10, 20, 40]`
argument path: `model/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/descriptor[se_a_mask]/axis_neuron`

Size of the submatrix of G (embedding matrix).

`activation_function:`

type: `str`, optional, default: `tanh`
argument path:
`model/descriptor[se_a_mask]/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/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/descriptor[se_a_mask]/type_one_side`

If true, the embedding network parameters vary by types of neighbor atoms only, so there will be $N_text{types}$ sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be $N_text{types}^2$ sets of embedding network parameters.

`exclude_types:`

type: `list`, optional, default: `[]`
argument path: `model/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/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/descriptor[se_a_mask]/trainable`

If the parameters in the embedding net is trainable

`seed:`

type: `NoneType` | `int`, optional
argument path: `model/descriptor[se_a_mask]/seed`

Random seed for parameter initialization

`fitting_net:`

type: `dict`, optional
argument path: `model/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/fitting_net/type`
possible choices: *`ener`*, *`dos`*, *`dipole`*, *`polar`*

The type of the fitting. See explanation below.

- ener: Fit an energy model (potential energy surface).
- dos : Fit a density of states model. The total density of states / site-projected density of states labels should be provided by dos.npy or atom_dos.npy in each data system. The file has number of frames lines and number of energy grid columns (times number of atoms in atom_dos.npy). See loss parameter.
- dipole: Fit an atomic dipole model. Global dipole labels or atomic dipole labels for all the selected atoms (see sel_type) should be provided by dipole.npy in each data system. The file either has number of frames lines and 3 times of number of selected atoms columns, or has number of frames lines and 3 columns. See loss parameter.
- polar: Fit an atomic polarizability model. Global polarizazbility labels or atomic polarizability labels for all the selected atoms (see sel_type) should be provided by polarizability.npy in each data system. The file eith has number of frames lines and 9 times of number of selected atoms columns, or has number of frames lines and 9 columns. See loss parameter.

When type is set to `ener`:

`numb_fparam:`

type: `int`, optional, default: `0`
argument path: `model/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/fitting_net[ener]/numb_aparam`

The dimension of the atomic parameter. If set to >0, file aparam.npy should be included to provided the input aparams.

neuron:

type: `list`, optional, default: `[120, 120, 120]`, alias: n_neuron

argument path: `model/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/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/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/fitting_net[ener]/resnet_dt`

Whether to use a "Timestep" in the skip connection

trainable:

type: `bool` | `list`, optional, default: `True`

argument path: `model/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 tihs list should be equal to len(neuron)+1.

rcond:

type: `float`, optional, default: `0.001`

argument path: `model/fitting_net[ener]/rcond`

The condition number used to determine the inital energy shift for each type of atoms.

`seed`:

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

Random seed for parameter initialization of the fitting net

`atom_ener`:

type: `list`, optional, default: `[]`
argument path: `model/fitting_net[ener]/atom_ener`

Specify the atomic energy in vacuum for each type

`layer_name`:

type: `list`, optional
argument path: `model/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/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/fitting_net[dos]/numb_fparam`

The dimension of the frame parameter. If set to >0, file fparam.npy should be included to provided the input fparams.

`numb_aparam`:

type: `int`, optional, default: `0`
argument path: `model/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: `list`, optional, default: `[120, 120, 120]`
argument path: `model/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/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/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/fitting_net[dos]/resnet_dt`

Whether to use a “Timestep” in the skip connection

`trainable:`

type: `bool` | `list`, optional, default: `True`

argument path: `model/fitting_net[dos]/trainable`

Whether the parameters in the fitting net are trainable. This option can be

- bool: True if all parameters of the fitting net are trainable, False otherwise.
- list of bool: Specifies if each layer is trainable. Since the fitting net is composed by hidden layers followed by a output layer, the length of tihs list should be equal to len(neuron)+1.

`rcond:`

type: `float`, optional, default: `0.001`

argument path: `model/fitting_net[dos]/rcond`

The condition number used to determine the inital energy shift for each type of atoms.

`seed:`

type: `NoneType` | `int`, optional

argument path: `model/fitting_net[dos]/seed`

Random seed for parameter initialization of the fitting net

`numb_dos:`

type: `int`, optional, default: `300`

argument path: `model/fitting_net[dos]/numb_dos`

The number of gridpoints on which the DOS is evaluated (NEDOS in VASP)

When type is set to `dipole`:

`neuron:`

type: `list`, optional, default: `[120, 120, 120]`, alias: n_neuron

argument path: `model/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/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/fitting_net[dipole]/resnet_dt`

Whether to use a "Timestep" in the skip connection

`precision:`

type: `str`, optional, default: `default`
argument path: `model/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` | `list`, optional, alias: dipole_type
argument path: `model/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/fitting_net[dipole]/seed`

Random seed for parameter initialization of the fitting net

When type is set to `polar`:

`neuron:`

type: `list`, optional, default: `[120, 120, 120]`, alias: n_neuron
argument path: `model/fitting_net[polar]/neuron`

The number of neurons in each hidden layers of the fitting net. When two hidden layers are of the same size, a skip connection is built.

`activation_function:`

type: `str`, optional, default: `tanh`
argument path: `model/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/fitting_net[polar]/resnet_dt`

Whether to use a “Timestep” in the skip connection

`precision:`

type: `str`, optional, default: `default`
argument path: `model/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/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: `float` | `list`, optional, default: `1.0`
argument path: `model/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/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` | `list`, optional, alias: pol_type
argument path: `model/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/fitting_net[polar]/seed`

Random seed for parameter initialization of the fitting net

`fitting_net_dict:`

type: `dict`, optional
argument path: `model/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.

`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: `list`
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.

`sys_charge_map:`

type: `list`
argument path: `model/modifier[dipole_charge]/sys_charge_map`

The charge of real atoms. The list length should be the same as the type_map

`ewald_beta:`

type: `float`, optional, default: `0.4`
argument path: `model/modifier[dipole_charge]/ewald_beta`

The splitting parameter of Ewald sum. Unit is A^-1

`ewald_h:`

type: `float`, optional, default: `1.0`
argument path: `model/modifier[dipole_charge]/ewald_h`

The grid spacing of the FFT grid. Unit is A

`compress:`

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

Model compression configurations

Depending on the value of type, different sub args are accepted.

`type:`

type: `str` (flag key), default: `se_e2_a`
argument path: `model/compress/type`

possible choices: *se_e2_a*

The type of model compression, which should be consistent with the descriptor type.

When type is set to `se_e2_a` (or its alias `se_a`):

`model_file:`

type: `str`
argument path: `model/compress[se_e2_a]/model_file`

The input model file, which will be compressed by the DeePMD-kit.

`table_config:`

type: `list`
argument path: `model/compress[se_e2_a]/table_config`

The arguments of model compression, including extrapolate(scale of model extrapolation), stride(uniform stride of tabulation's first and second table), and frequency(frequency of tabulation overflow check).

`min_nbor_dist:`

type: `float`
argument path: `model/compress[se_e2_a]/min_nbor_dist`

The nearest distance between neighbor atoms saved in the frozen model.

`spin:`

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

The settings for systems with spin.

`use_spin:`

type: `list`
argument path: `model/spin/use_spin`

Whether to use atomic spin model for each atom type

`spin_norm:`

type: `list`
argument path: `model/spin/spin_norm`

The magnitude of atomic spin for each atom type with spin

`virtual_len:`

type: `list`
argument path: `model/spin/virtual_len`

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

`learning_rate:`

type: `dict`, optional
argument path: `learning_rate`

The definitio of learning rate

`scale_by_worker:`

type: `str`, optional, default: `linear`
argument path: `learning_rate/scale_by_worker`

When parallel training or batch size scaled, how to alter learning rate. Valid values are linear`(default), `sqrt or none.

Depending on the value of type, different sub args are accepted.

`type:`

type: `str` (flag key), default: `exp`
argument path: `learning_rate/type`
possible choices: *exp*

The type of the learning rate.

When type is set to `exp`:

`start_lr:`

type: `float`, optional, default: `0.001`
argument path: `learning_rate[exp]/start_lr`

The learning rate 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*, *dos*, *tensor*, *ener_spin*

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_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]/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_force and limit_pref_force are set to 0, then the force will be ignored.

`limit_pref_f:`

type: `float` | `int`, optional, default: `1.0`

argument path: `loss[ener]/limit_pref_f`

The prefactor of force loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

`start_pref_v:`

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

argument path: `loss[ener]/start_pref_v`

The prefactor of virial loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the virial label should be provided by file virial.npy in each data system. If both start_pref_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]/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]/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]/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]/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]/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]/relative_f`

If provided, relative force error will be used in the loss. The difference of force will be normalized by the magnitude of the force in the label with a shift given by relative_f, i.e. DF_i / (|| F || + relative_f) with DF denoting the difference between prediction and label and || F || denoting the L2 norm of the label.

`enable_atom_ener_coeff`:

type: `bool`, optional, default: `False`

argument path: `loss[ener]/enable_atom_ener_coeff`

If true, the energy will be computed as sum_i c_i E_i. c_i should be provided by file atom_ener_coeff.npy in each data system, otherwise it's 1.

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 intergral of DOS) loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the Cumulative Distribution Function (cumulative intergral of DOS) label should be provided by file Cumulative Distribution Function (cumulative intergral of DOS).npy in each data system. If both start_pref_Cumulative Distribution Function (cumulative intergral of DOS) and limit_pref_Cumulative Distribution Function (cumulative intergral of DOS) are set to 0, then the Cumulative Distribution Function (cumulative intergral of DOS) will be ignored.

`limit_pref_cdf:`

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

argument path: `loss[dos]/limit_pref_cdf`

The prefactor of Cumulative Distribution Function (cumulative intergral of DOS) loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

`start_pref_ados:`

type: `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 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 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.

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 || + relative_f) with DF denoting the difference between prediction and label and || F || denoting the L2 norm of the label.

`enable_atom_ener_coeff:`

type: `bool`, optional, default: `False`

argument path: `loss[ener_spin]/enable_atom_ener_coeff`

If true, the energy will be computed as sum_i c_i E_i. c_i should be provided by file atom_ener_coeff.npy in each data system, otherwise it's 1.

`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` | `list`

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` | `list`, 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.

`auto_prob:`

type: `str`, optional, default: `prob_sys_size`, alias: auto_prob_style

argument path: `training/training_data/auto_prob`

Determine the probability of systems automatically. The method is assigned by this key and can be

- "prob_uniform" : the probability all the systems are equal, namely 1.0/self.get_nsystems()
- "prob_sys_size" : the probability of a system is proportional to the number of batches in the system
- "prob_sys_size;stt_idx:end_idx:weight;stt_idx:end_idx:weight;…" : the list of systems is devided into blocks. A block is specified by stt_idx:end_idx:weight, where stt_idx is the starting index of the system, end_idx is then ending (not including) index of the system, the probabilities of the systems in this block sums up to weight, and the relatively probabilities within this block is proportional to the number of batches in the system.

`sys_probs:`

type: `NoneType` | `list`, 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` | `list`

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` | `list`, optional, default: `auto`

argument path: `training/validation_data/batch_size`

This key can be

- list: the length of which is the same as the systems. The batch size of each system is given by the elements of the list.
- int: all systems use the same batch size.
- string "auto": automatically determines the batch size so that the batch_size times the number of atoms in the system is no less than 32.
- string "auto:N": automatically determines the batch size so that the batch_size times the number of atoms in the system is no less than N.

`auto_prob:`

type: `str`, optional, default: `prob_sys_size`, alias: auto_prob_style

argument path: `training/validation_data/auto_prob`

Determine the probability of systems automatically. The method is assigned by this key and can be

- "prob_uniform" : the probability all the systems are equal, namely 1.0/self.get_nsystems()
- "prob_sys_size" : the probability of a system is proportional to the number of batches in the system
- "prob_sys_size;stt_idx:end_idx:weight;stt_idx:end_idx:weight;..." : the list of systems is devided into blocks. A block is specified by stt_idx:end_idx:weight, where stt_idx is the starting index of the system, end_idx is then ending (not including) index of the system, the probabilities of the systems in this block sums up to weight, and the relatively probabilities within this block is proportional to the number of batches in the system.

`sys_probs:`

type: `NoneType` | `list`, 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 file name of saving check point.

`disp_training:`

type: `bool`, optional, default: `True`

argument path: `training/disp_training`

Displaying verbose information during training.

`time_training:`

type: `bool`, optional, default: `True`
argument path: `training/time_training`

Timing durining training.

`profiling:`

type: `bool`, optional, default: `False`
argument path: `training/profiling`

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.

`nvnmd:`

type: `dict`, optional
argument path: `nvnmd`

The nvnmd options.

`net_size:`

type: `int`
argument path: `nvnmd/net_size`

configuration the number of nodes of fitting_net, just can be set as 128

`map_file:`

type: `str`
argument path: `nvnmd/map_file`

A file containing the mapping tables to replace the calculation of embedding nets

`config_file:`

type: `str`
argument path: `nvnmd/config_file`

A file containing the parameters about how to implement the model in certain hardware

`weight_file:`

type: `str`
argument path: `nvnmd/weight_file`

a *.npy file containing the weights of the model

`enable:`

type: `bool`
argument path: `nvnmd/enable`

enable the nvnmd training

`restore_descriptor:`

type: `bool`
argument path: `nvnmd/restore_descriptor`

enable to restore the parameter of embedding_net from weight.npy

`restore_fitting_net:`

type: `bool`
argument path: `nvnmd/restore_fitting_net`

enable to restore the parameter of fitting_net from weight.npy

`quantize_descriptor`:

type: `bool`
argument path: `nvnmd/quantize_descriptor`

enable the quantizatioin of descriptor

`quantize_fitting_net`:

type: `bool`
argument path: `nvnmd/quantize_fitting_net`

enable the quantizatioin of fitting_net

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.

Num of GPU cards	Seconds every 100 samples	Samples per second	Speed up
1	1.4515	68.89	1.00
2	1.5962	62.65*2	1.82
4	1.7635	56.71*4	3.29
8	1.7267	57.91*8	6.72

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 Perform the multi-task training

Training on multiple data sets (each data set contains several data systems) can be performed in multi-task mode, with one common descriptor and multiple specific fitting nets for each data set. One can simply switch the following parameters in training input script to perform multi-task mode:

- fitting_net –> fitting_net_dict, each key of which can be one individual fitting net.
- training_data, validation_data –> data_dict, each key of which can be one individual data set contains several data systems for corresponding fitting net, the keys must be consistent with those in fitting_net_dict.
- loss –> loss_dict, each key of which can be one individual loss setting for corresponding fitting net, the keys must be consistent with those in fitting_net_dict, if not set, the corresponding fitting net will use the default loss.
- (Optional) fitting_weight, each key of which can be a non-negative integer or float, deciding the chosen probability for corresponding fitting net in training, if not set or invalid, the corresponding fitting net will not be used.

The training procedure will automatically choose single-task or multi-task mode, based on the above parameters. Note that parameters of single-task mode and multi-task mode can not be mixed.

An example input for training energy and dipole in water system can be found here: multi-task input on water.

The supported descriptors for multi-task mode are listed:

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

The supported fitting nets for multi-task mode are listed:

- ener
- dipole
- polar

The output of `dp freeze` command in multi-task mode can be seen in freeze command.

5.5.2 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:

```
    "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:

```
    "model": {
        "type_map": ["O", "H"],
        "descriptor": {},
        "fitting_net_dict": {
            "water_ener": {},
            "water_ener_2": {
                "neuron":       [240, 240, 240],
                "resnet_dt":    true,
            }
        },
    }
```

It will autocomplete the configurations according to the frozen graph.

Note that for newly added fitting net keys, other parts in the input script, including data_dict and loss_dict (optionally fitting_weight), should be set explicitly. While for old fitting net keys, it will inherit the old configurations if not set.

Finally, you can perform the modified multi-task training from the frozen model with command:

```
$ dp train input.json --init_frz_model graph.pb
```

5.5.3 Share layers among energy fitting networks

The multi-task training can be used to train multiple levels of energies (e.g. DFT and CCSD(T)) at the same time. In this situation, one can set model/fitting_net[ener]/layer_name> to share some of layers among fitting networks. The architecture of the layers with the same name should be the same.

For example, if one want to share the first and the third layers for two three-hidden-layer fitting networks, the following parameters should be set.

```
"fitting_net_dict": {
    "ccsd": {
        "neuron": [
            240,
            240,
            240
        ],
        "layer_name": ["l0", null, "l2", null]
    },
    "wb97m": {
        "neuron": [
            240,
            240,
            240
        ],
        "layer_name": ["l0", null, "l2", null]
    }
}
```

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.

```
    "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"
    }
```

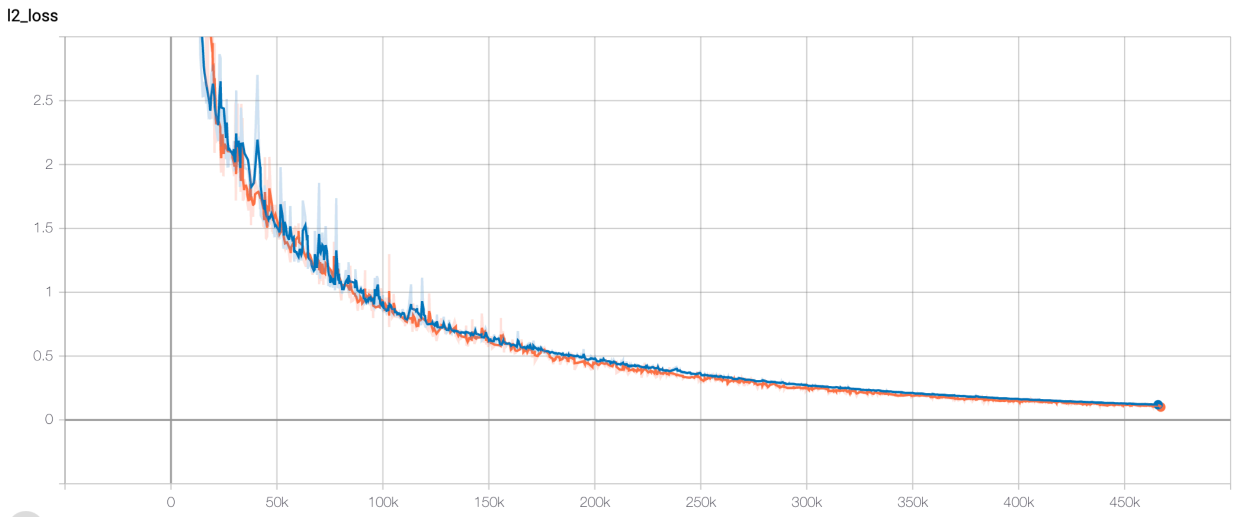
Once you have event files, run TensorBoard and provide the log directory. This should print that TensorBoard has started. Next, connect to http://tensorboard_server_ip:6006.

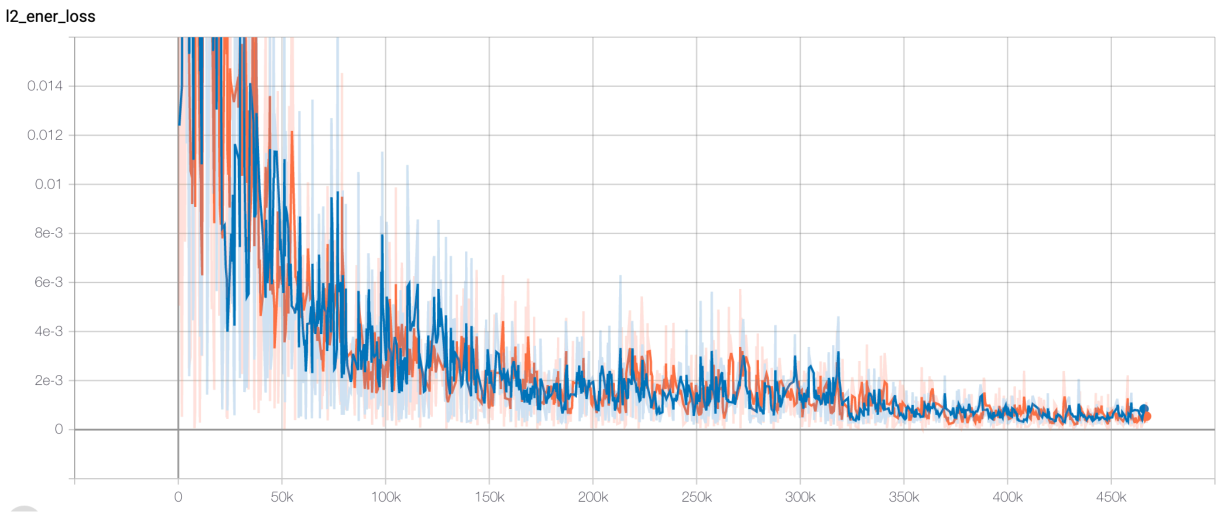
TensorBoard requires a logdir to read logs from. For info on configuring TensorBoard, run TensorBoard –help. One can easily change the log name with "tensorboard_log_dir" and the sampling frequency with "tensorboard_freq".

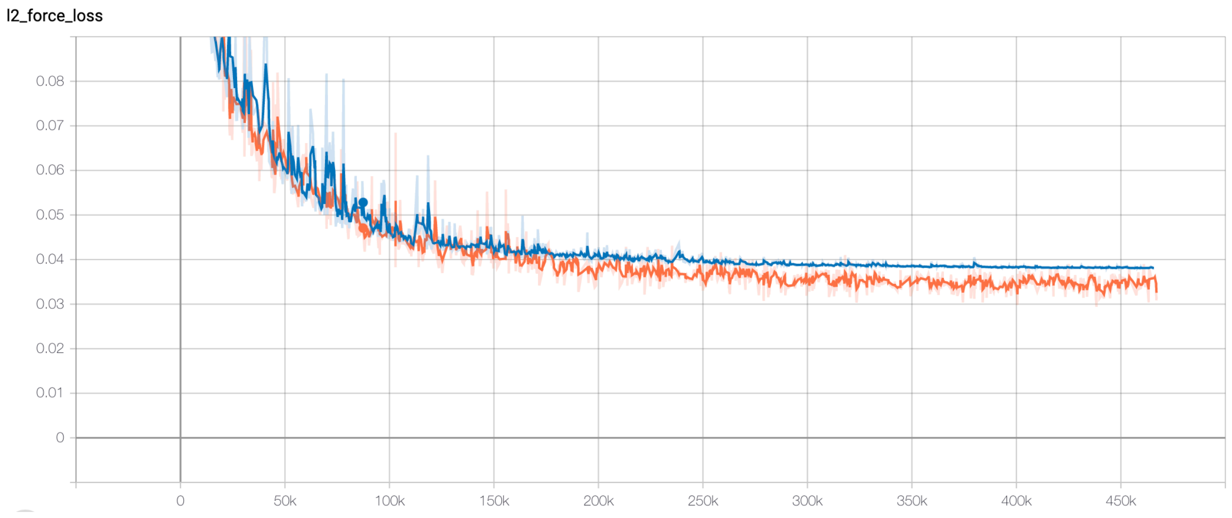
```
tensorboard --logdir path/to/logs
```

5.6.3 Examples

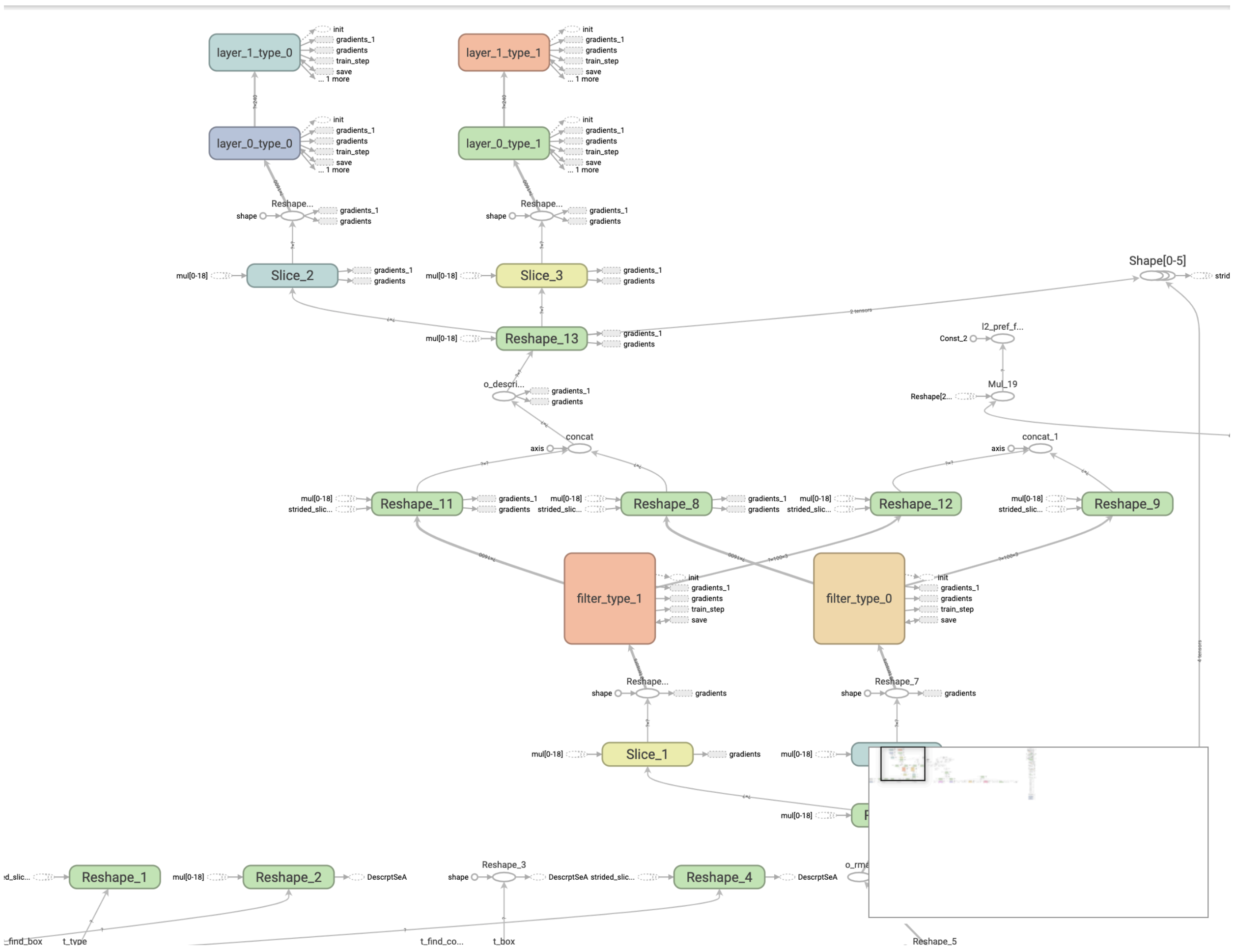
Tracking and visualizing loss metrics(red:train, blue:test)



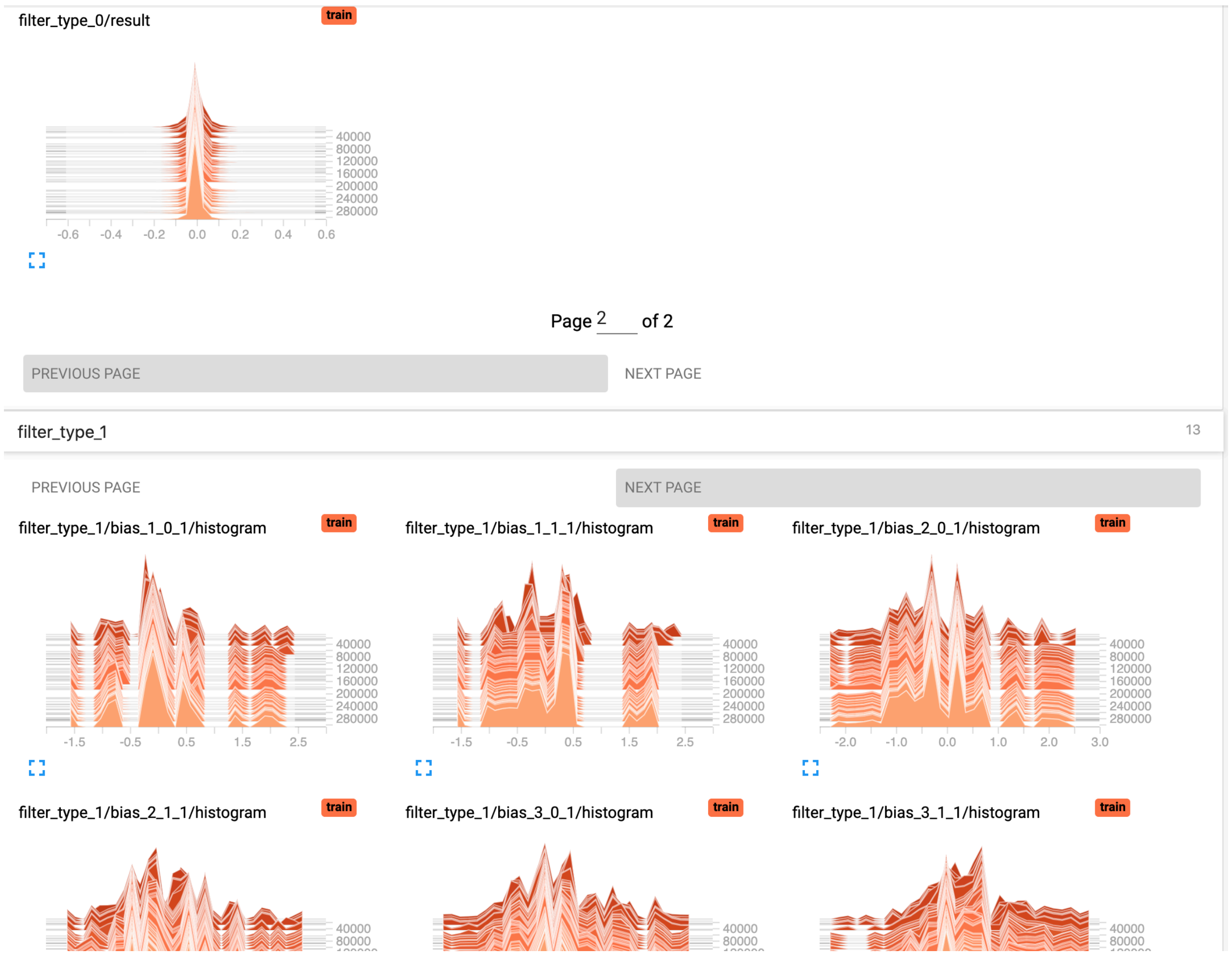


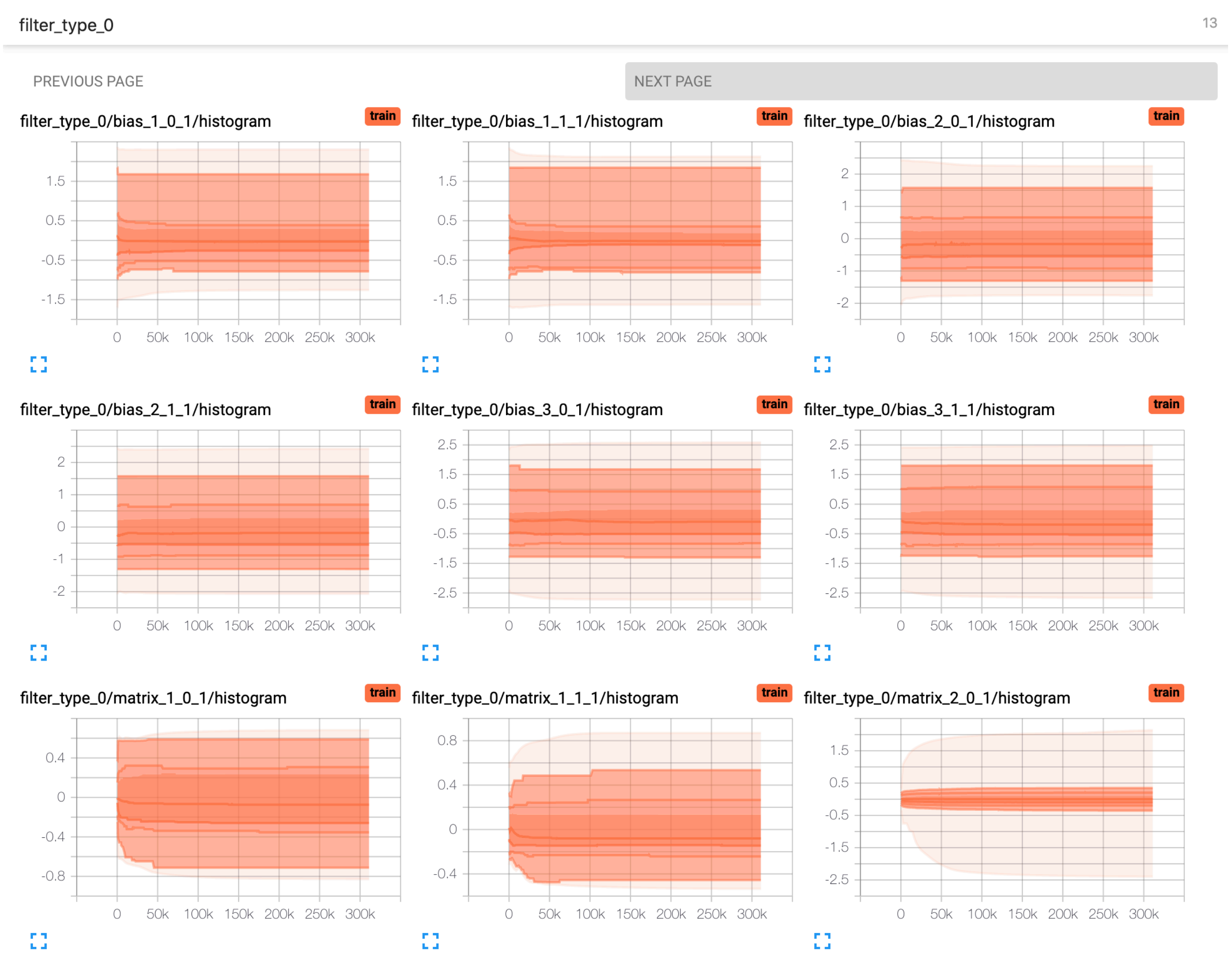


Visualizing DeePMD-kit model graph

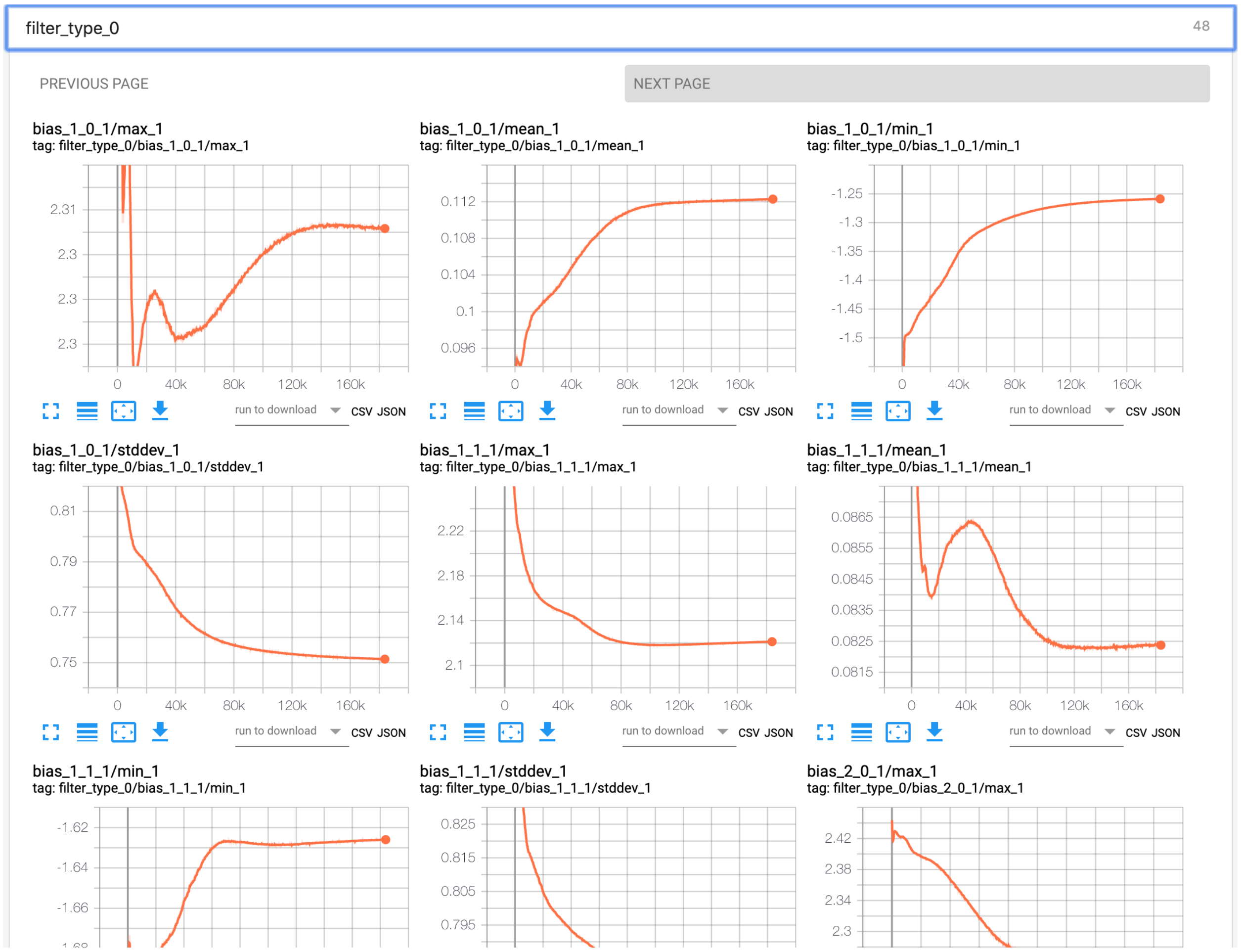


Viewing histograms of weights, biases, or other tensors as they change over time


filter_type_0/result
train
40000
80000
120000
160000
200000
240000
280000
-0.6
-0.4
-0.2
0.0
0.2
0.4
0.6
Page 2 of 2
PREVIOUS PAGE
NEXT PAGE
filter_type_1
13
PREVIOUS PAGE
NEXT PAGE
filter_type_1/bias_1_0_1/histogram
filter_type_1/bias_1_1_1/histogram
filter_type_1/bias_2_0_1/histogram
filter_type_1/bias_2_1_1/histogram
filter_type_1/bias_3_0_1/histogram
filter_type_1/bias_3_1_1/histogram
train


filter_type_0
13
PREVIOUS PAGE
NEXT PAGE
filter_type_0/bias_1_0_1/histogram
train
filter_type_0/bias_1_1_1/histogram
train
filter_type_0/bias_2_0_1/histogram
train
filter_type_0/bias_2_1_1/histogram
train
filter_type_0/bias_3_0_1/histogram
train
filter_type_0/bias_3_1_1/histogram
train
filter_type_0/matrix_1_0_1/histogram
train
filter_type_0/matrix_1_1_1/histogram
train
filter_type_0/matrix_2_0_1/histogram
train
0
50k
100k
150k
200k
250k
300k

Viewing summaries of trainable variables


filter_type_0
48
PREVIOUS PAGE
NEXT PAGE
bias_1_0_1/max_1
tag: filter_type_0/bias_1_0_1/max_1
bias_1_0_1/mean_1
tag: filter_type_0/bias_1_0_1/mean_1
bias_1_0_1/min_1
tag: filter_type_0/bias_1_0_1/min_1
bias_1_0_1/stddev_1
tag: filter_type_0/bias_1_0_1/stddev_1
bias_1_1_1/max_1
tag: filter_type_0/bias_1_1_1/max_1
bias_1_1_1/mean_1
tag: filter_type_0/bias_1_1_1/mean_1
bias_1_1_1/min_1
tag: filter_type_0/bias_1_1_1/min_1
bias_1_1_1/stddev_1
tag: filter_type_0/bias_1_1_1/stddev_1
bias_2_0_1/max_1
tag: filter_type_0/bias_2_0_1/max_1
run to download
CSV JSON
0
40k
80k
120k
160k

5.6.4 Attention

Allowing the tensorboard analysis will takes extra execution time.(eg, 15% increasing @Nvidia GTX 1080Ti double precision with default water sample)

TensorBoard can be used in Google Chrome or Firefox. Other browsers might work, but there may be bugs or performance issues.

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:

```
$ 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:

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

CHAPTER

SIX

FREEZE AND COMPRESS

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

Once the frozen model is obtained from DeePMD-kit, we can get the neural network structure and its parameters (weights, biases, etc.) from the trained model, and compress it in the following way:

```
dp compress -i graph.pb -o graph-compress.pb
```

where `-i` gives the original frozen model, `-o` gives the compressed model. Several other command line options can be passed to `dp compress`, which can be checked with

```
$ dp compress --help
```

An explanation will be provided

```
usage: dp compress [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
                   [-m {master,collect,workers}] [-i INPUT] [-o OUTPUT]
                   [-s STEP] [-e EXTRAPOLATE] [-f FREQUENCY]
                   [-c CHECKPOINT_FOLDER]

optional arguments:
```

(continues on next page)

(continued from previous page)

```
  -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 it's uniform stride, while the second sub-table takes 100 * stride as it's uniform stride. The range of the first table is automatically detected by DeePMD-kit, while the second table ranges from the first table's upper boundary(upper) to the extrapolate(parameter) * upper. Finally, we added a check frequency parameter. It indicates how often the program checks for overflow(if the input environment matrix overflows the first or second table range) during the MD inference.

Justification of model compression

Model compression, with little loss of accuracy, can greatly speed up MD inference time. According to different simulation systems and training parameters, the speedup can reach more than 10 times at both CPU and GPU devices. At the same time, model compression can greatly change memory usage, reducing as much as 20 times under the same hardware conditions.

Acceptable original model version

The model compression interface requires the version of DeePMD-kit used in the original model generation should be `2.0.0-alpha.0` or above. If one has a frozen 1.2 or 1.3 model, one can upgrade it through the `dp convert-from` interface. (eg: `dp convert-from 1.2/1.3 -i old_frozen_model.pb -o new_frozen_model.pb`)

Acceptable descriptor type

Descriptors with `se_e2_a`, `se_e3`, and `se_e2_r` 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

CHAPTER

SEVEN

TEST

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

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:

```
usage: dp model-devi [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}]
                     [-l LOG_PATH] [-m MODELS [MODELS ...]] [-s SYSTEM]
                     [-S SET_PREFIX] [-o OUTPUT] [-f FREQUENCY] [-i ITEMS]

optional arguments:
  -h, --help            show this help message and exit
  -v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}, --log-level {DEBUG,3,INFO,2,WARNING,1,ERROR,0}
                        set verbosity level by string or number, 0=ERROR,
                        1=WARNING, 2=INFO and 3=DEBUG (default: INFO)
  -l LOG_PATH, --log-path LOG_PATH
                        set log file to log messages to disk, if not
                        specified, the logs will only be output to console
                        (default: None)
  -m MODELS [MODELS ...], --models MODELS [MODELS ...]
                        Frozen models file to import (default:
                        ['graph.000.pb', 'graph.001.pb', 'graph.002.pb',
                        'graph.003.pb'])
  -s SYSTEM, --system SYSTEM
                        The system directory, not support recursive detection.
                        (default: .)
  -S SET_PREFIX, --set-prefix SET_PREFIX
                        The set prefix (default: set)
  -o OUTPUT, --output OUTPUT
                        The output file for results of model deviation
                        (default: model_devi.out)
  -f FREQUENCY, --frequency FREQUENCY
                        The trajectory frequency of the system (default: 1)
```

For more details concerning the definition of model deviation and its application, please refer to Yuzhi Zhang, Haidi Wang, Weijie Chen, Jinzhe Zeng, Linfeng Zhang, Han Wang, and Weinan E, DP-GEN: A concurrent learning platform for the generation of reliable deep learning based potential energy models, Computer Physics Communications, 2020, 253, 107206.

CHAPTER

EIGHT

INFERENCE

Note 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.

8.2 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:

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

int main(){
  deepmd::DeepPot dp ("graph.pb");
  std::vector<double > coord = {1., 0., 0., 0., 0., 1.5, 1. ,0. ,3.};
  std::vector<double > cell = {10., 0., 0., 0., 10., 0., 0., 0., 10.};
  std::vector<int > atype = {1, 0, 1};
  double e;
  std::vector<double > f, v;
  dp.compute (e, f, v, coord, atype, cell);
}
```

where `e`, `f` and `v` are predicted energy, force and virial of the system, respectively. See *deepmd::DeepPot* for details.

You can compile `infer_water.cpp` using `gcc`:

```
gcc infer_water.cpp -L $deepmd_root/lib -L $tensorflow_root/lib -I $deepmd_root/include -Wl,--no-
↪as-needed -ldeepmd_cc -lstdc++ -ltensorflow_cc -Wl,-rpath=$deepmd_root/lib -Wl,-rpath=
↪$tensorflow_root/lib -o infer_water
```

and then run the program:

```
./infer_water
```

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:

```
#include <stdio.h>
#include <stdlib.h>
#include "deepmd/c_api.h"

int main(){
  const char* model = "graph.pb";
  double coord[] = {1., 0., 0., 0., 0., 1.5, 1. ,0. ,3.};
  double cell[] = {10., 0., 0., 0., 10., 0., 0., 0., 10.};
  int atype[] = {1, 0, 1};
  // init C pointers with given memory
  double* e = malloc(sizeof(*e));
  double* f = malloc(sizeof(*f) * 9); // natoms * 3
  double* v = malloc(sizeof(*v) * 9);
  double* ae = malloc(sizeof(*ae) * 9); // natoms
  double* av = malloc(sizeof(*av) * 27); // natoms * 9
  // DP model
  DP_DeepPot* dp = DP_NewDeepPot(model);
```

(continues on next page)

(continued from previous page)

```
  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`:

```
gcc infer_water.c -L $deepmd_root/lib -L $tensorflow_root/lib -I $deepmd_root/include -Wl,--no-as-
↪needed -ldeepmd_c -Wl,-rpath=$deepmd_root/lib -Wl,-rpath=$tensorflow_root/lib -o infer_water
```

and then run the program:

```
./infer_water
```

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`.

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

int main(){
  deepmd::hpp::DeepPot dp ("graph.pb");
  std::vector<double > coord = {1., 0., 0., 0., 0., 1.5, 1. ,0. ,3.};
  std::vector<double > cell = {10., 0., 0., 0., 10., 0., 0., 0., 10.};
  std::vector<int > atype = {1, 0, 1};
  double e;
  std::vector<double > f, v;
  dp.compute (e, f, v, coord, atype, cell);
}
```

Note that the feature of the header-only C++ library is still limited compared to the original C++ library. See *deepmd::hpp::DeepPot* for details.

You can compile `infer_water_hpp.cpp` using `gcc`:

```
gcc infer_water_hpp.hpp -L $deepmd_root/lib -L $tensorflow_root/lib -I $deepmd_root/include -Wl,--
↪no-as-needed -ldeepmd_c -Wl,-rpath=$deepmd_root/lib -Wl,-rpath=$tensorflow_root/lib -o infer_
↪water_hpp
```

and then run the program:

```
./infer_water_hpp
```

In some cases, one may want to pass the custom neighbor list instead of the native neighbor list. The above code can be revised as follows:

```
  // neighbor list
  std::vector<std::vector<int >> nlist_vec = {
    {1, 2},
    {0, 2},
    {0, 1}
    };
  std::vector<int> ilist(3), numneigh(3);
  std::vector<int*> firstneigh(3);
  InputNlist nlist(3, &ilist[0], &numneigh[0], &firstneigh[0]);
  convert_nlist(nlist, nlist_vec);
  dp.compute (e, f, v, coord, atype, cell, 0, nlist, 0);
```

Here, `nlist_vec` means the neighbors of atom 0 are atom 1 and atom 2, the neighbors of atom 1 are atom 0 and atom 2, and the neighbors of atom 2 are atom 0 and atom 1.

CHAPTER

NINE

COMMAND LINE INTERFACE

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-nvnmd}
          ...
```

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-nvnmd

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}] [-l LOG_PATH]
            [-r RAW_MODEL] [-O OLD_MODEL] [-o OUTPUT]
```

Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: “INFO”
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console
-r, --raw-model	the model receiving parameters Default: “raw_frozen_model.pb”
-O, --old-model	the model providing parameters Default: “old_frozen_model.pb”
-o, --output	the model after passing parameters Default: “frozen_model.pb”

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	the input parameter file in json or yaml format

Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: “INFO”
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console
-m, --mpi-log	Possible choices: master, collect, workers Set the manner of logging when running with MPI. ‘master’ logs only on main process, ‘collect’ broadcasts logs from workers to master and ‘workers’ means each process will output its own log Default: “master”
-i, --init-model	Initialize the model by the provided checkpoint.

-r, --restart	Restart the training from the provided checkpoint.
-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 [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
          [-c CHECKPOINT_FOLDER] [-o OUTPUT] [-n NODE_NAMES] [-w NVNMD_WEIGHT]
          [--united-model]
```

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 Default: 100
-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"
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console
-m, --mpi-log	Possible choices: master, collect, workers Set the manner of logging when running with MPI. 'master' logs only on main process, 'collect' broadcasts logs from workers to master and 'workers' means each process will output its own log Default: "master"
-i, --input	The original frozen model, which will be compressed by the code Default: "frozen_model.pb"
-o, --output	The compressed model Default: "frozen_model_compressed.pb"
-s, --step	Model compression uses fifth-order polynomials to interpolate the embedding-net. It introduces two tables with different step size to store the parameters of the polynomials. The first table covers the range of the training data, while the second table is an extrapolation of the training data. The domain of each table is uniformly divided by a given step size. And the step(parameter) denotes the step size of the first table and the second table will use 10 * step as it's step size to save the memory. Usually the value ranges from 0.1 to 0.001. Smaller step means higher accuracy and bigger model size Default: 0.01
-e, --extrapolate	The domain range of the first table is automatically detected by the code: [d_low, d_up]. While the second table ranges from the first table's upper boundary(d_up) to the extrapolate(parameter) * d_up: [d_up, extrapolate * d_up] Default: 5
-f, --frequency	The frequency of tabulation overflow check(Whether the input environment matrix overflow the first or second table range). By default do not check the overflow Default: -1

-c, --checkpoint-folder	path to checkpoint folder Default: "model-compression"
-t, --training-script	The training script of the input frozen model

examples:
dp compress dp compress -i graph.pb -o compressed.pb

9.3.6 doc-train-input

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

```
dp doc-train-input [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
                   [--out-type {rst,json}]
```

Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: "INFO"
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console
--out-type	Possible choices: rst, json The output type Default: "rst"

9.3.7 model-devi

calculate model deviation

```
dp model-devi [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
              [-m MODELS [MODELS ...]] [-s SYSTEM] [-S SET_PREFIX] [-o OUTPUT]
              [-f FREQUENCY]
```

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

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 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

examples:
    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]
               [-r RESTART] [-s {s1,s2}]
               INPUT
```

Positional Arguments

INPUT	the input parameter file in json format

Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: "INFO"
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console
-r, --restart	Restart the training from the provided checkpoint.
-s, --step	Possible choices: s1, s2 steps to train model of NVNMD: s1 (train CNN), s2 (train QNN) Default: "s1"

CHAPTER

TEN

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.

```
from ase import Atoms
from deepmd.calculator import DP

water = Atoms(
    "H2O",
    positions=[(0.7601, 1.9270, 1), (1.9575, 1, 1), (1.0, 1.0, 1.0)],
    cell=[100, 100, 100],
    calculator=DP(model="frozen_model.pb"),
)
print(water.get_potential_energy())
print(water.get_forces())
```

Optimization is also available:

```
from ase.optimize import BFGS

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

10.2 Run MD with LAMMPS

Running an MD simulation with LAMMPS is simpler. In the LAMMPS input file, one needs to specify the pair style as follows

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

where `graph.pb` is the file name of the frozen model. `pair_coeff` maps atom names (`O H`) with LAMMPS atom types (integers from 1 to Ntypes, i.e. `1 2`).

10.3 LAMMPS commands

10.3.1 Enable DeePMD-kit plugin (plugin mode)

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

```
plugin load libdeepmd_lmp.so
```

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

```
LAMMPS_PLUGIN_PATH=$deepmd_root/lib/deepmd_lmp
```

where `$deepmd_root` is the directory to install C++ interface.

The built-in mode doesn't need this step.

10.3.2 pair_style `deepmd`

The DeePMD-kit package provides the pair_style `deepmd`

```
pair_style deepmd models ... keyword value ...
```

- deepmd = style of this pair_style
- models = frozen model(s) to compute the interaction. If multiple models are provided, then only the first model serves to provide energy and force prediction for each timestep of molecular dynamics, and the model deviation will be computed among all models every `out_freq` timesteps.
- keyword = out_file or out_freq or fparam or fparam_from_compute or atomic or relative or relative_v or aparam or ttm

Examples

```
pair_style deepmd graph.pb
pair_style deepmd graph.pb fparam 1.2
pair_style deepmd graph_0.pb graph_1.pb graph_2.pb out_file md.out out_freq 10 atomic relative 1.0
pair_coeff * * O H

pair_style deepmd cp.pb fparam_from_compute TEMP
compute    TEMP all temp
```

Description

Evaluate the interaction of the system by using Deep Potential or Deep Potential Smooth Edition. It is noticed that deep potential is not a "pairwise" interaction, but a multi-body interaction.

This pair style takes the deep potential defined in a model file that usually has the .pb extension. The model can be trained and frozen by package DeePMD-kit, which can have either double or single float precision interface.

The model deviation evalulates the consistency of the force predictions from multiple models. By default, only the maximal, minimal and average model deviations are output. If the key `atomic` is set, then the model deviation of force prediction of each atom will be output.

By default, the model deviation is output in absolute value. If the keyword `relative` is set, then the relative model deviation of the force will be output, including values output by the keyword `atomic`. The relative model deviation of the force on atom i is defined by

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

where D_{f_i} is the absolute model deviation of the force on atom i, f_i is the norm of the force and l is provided as the parameter of the keyword `relative`. If the keyword `relative_v` is set, then the relative model deviation of the virial will be output instead of the absolute value, with the same definition of that of the force:

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

If the keyword `fparam` is set, the given frame parameter(s) will be fed to the model. If the keyword `fparam_from_compute` is set, the global parameter(s) from compute command (e.g., temperature from compute temp command) will be fed to the model as the frame parameter(s). If the keyword `aparam` 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 the training parameter type_map is not set, atom names in the `pair_coeff` command cannot be set. In this case, atom type indexes in *`type.raw`* (integers from 0 to Ntypes-1) will map to LAMMPS atom types.

Spin is specified by keywords `virtual_len` and `spin_norm`. If the keyword `virtual_len` is set, the distance between virtual atom and its corresponding real atom for each type of magnetic atoms will be fed to the model as the spin parameters. If the keyword `spin_norm` is set, the magnitude of the magnetic moment for each type of magnetic atoms will be fed to the model as the spin parameters.

Restrictions

- The `deepmd` pair style is provided in the USER-DEEPMD package, which is compiled from the DeePMD-kit, visit the DeePMD-kit website for more information.

10.3.3 Compute tensorial properties

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

```
compute ID group-ID deeptensor/atom model_file
```

- ID: user-assigned name of the computation
- group-ID: ID of the group of atoms to compute
- deeptensor/atom: the style of this compute
- model_file: the name of the binary model file.

At this time, the training parameter type_map will be mapped to LAMMPS atom types.

Examples

```
compute         dipole all deeptensor/atom dipole.pb
```

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

```
dump            1 all custom 100 water.dump id type c_dipole[1] c_dipole[2] c_dipole[3]
```

Restrictions

- The `deeptensor/atom` compute is provided in the USER-DEEPMD package, which is compiled from the DeePMD-kit, visit the DeePMD-kit website for more information.

10.3.4 Long-range interaction

The reciprocal space part of the long-range interaction can be calculated by LAMMPS command `kspace_style`. To use it with DeePMD-kit, one writes

```
pair_style        deepmd graph.pb
pair_coeff  * *
kspace_style        pppm 1.0e-5
kspace_modify        gewald 0.45
```

Please notice that the DeePMD does nothing to the direct space part of the electrostatic interaction, because this part is assumed to be fitted in the DeePMD model (the direct space cut-off is thus the cut-off of the DeePMD model). The splitting parameter `gewald` is modified by the `kspace_modify` command.

10.3.5 Use of the centroid/stress/atom to get the full 3x3 "atomic-virial"

The DeePMD-kit allows also the computation of per-atom stress tensor defined as:

$$dvatom = \sum_m (\mathrm{r}_n - \mathrm{r}_m) \frac{de_m}{d\mathrm{r}_n}$$

Where r_n is the atomic position of nth atom, v_n velocity of the atom and $\frac{de_m}{d\mathrm{r}_n}$ the derivative of the atomic energy.

In LAMMPS one can get the per-atom stress using the command `centroid/stress/atom`:

```
compute ID group-ID centroid/stress/atom NULL virial
```

see LAMMPS doc page for more details on the meaning of the keywords.

Examples

In order of computing the 9-component per-atom stress

```
compute stress all centroid/stress/atom NULL virial
```

Thus `c_stress` is an array with 9 components in the order `xx,yy,zz,xy,xz,yz,yx,zx,zy`.

If you use this feature please cite D. Tisi, L. Zhang, R. Bertossa, H. Wang, R. Car, S. Baroni - arXiv preprint arXiv:2108.10850, 2021

10.3.6 Computation of heat flux

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

$$\mathrm{J} = \sum_n e_n \mathrm{v}_n + \sum_{n,m} (\mathrm{r}_m - \mathrm{r}_n) \frac{de_m}{d\mathrm{r}_n} \mathrm{v}_n$$

to compute the heat flux with LAMMPS:

```
compute ke_ID all ke/atom
compute pe_ID all pe/atom
compute stress_ID group-ID centroid/stress/atom NULL virial
compute flux_ID all heat/flux ke_ID pe_ID stress_ID
```

Examples

```
compute ke all ke/atom
compute pe all pe/atom
compute stress all centroid/stress/atom NULL virial
compute flux all heat/flux ke pe stress
```

`c_flux` is a global vector of length 6. The first three components are the x, y and z components of the full heat flux vector. The others are the components of the so-called convective portion, see LAMMPS doc page for more detailes.

If you use these features please cite D. Tisi, L. Zhang, R. Bertossa, H. Wang, R. Car, S. Baroni - arXiv preprint arXiv:2108.10850, 2021

10.4 Run path-integral MD with i-PI

The i-PI works in a client-server model. The i-PI provides the server for integrating the replica positions of atoms, while the DeePMD-kit provides a client named `dp_ipi` that computes the interactions (including energy, forces and virials). The server and client communicate via the Unix domain socket or the Internet socket. Installation instructions for i-PI can be found here. The client can be started by

```
i-pi input.xml &
dp_ipi water.json
```

It is noted that multiple instances of the client allow for computing, in parallel, the interactions of multiple replicas of the path-integral MD.

`water.json` is the parameter file for the client `dp_ipi`, and an example is provided:

```
{
    "verbose":                false,
    "use_unix":                true,
    "port":                 31415,
    "host":                 "localhost",
    "graph_file":         "graph.pb",
    "coord_file":         "conf.xyz",
    "atom_type" : {
        "OW":                 0,
        "HW1":                 1,
        "HW2":                 1
    }
}
```

The option **use_unix** is set to **true** to activate the Unix domain socket, otherwise, the Internet socket is used.

The option **port** should be the same as that in input.xml:

```
<port>31415</port>
```

The option **graph_file** provides the file name of the frozen model. The model can have either double or single float precision interface.

The `dp_ipi` gets the atom names from an XYZ file provided by **coord_file** (meanwhile ignores all coordinates in it) and translates the names to atom types by rules provided by **atom_type**.

10.5 Running MD with GROMACS

10.5.1 DP/MM Simulation

This part gives a simple tutorial on how to run a DP/MM simulation for methane in water, which means using DP for methane and TIP3P for water. All relevant files can be found in `examples/methane`.

Topology Preparation

Similar to QM/MM simulation, the internal interactions (including bond, angle, dihedrals, LJ, Columb) of the region described by a neural network potential (NNP) have to be turned off. In GROMACS, bonded interactions can be turned off by modifying `[ bonds ]`, `[ angles ]`, `[ dihedrals ]` and `[ pairs ]` sections. And LJ and Columb interactions must be turned off by `[ exclusions ]` section.

For example, if one wants to simulate ethane in water, using DeepPotential for methane and TIP3P for water, the topology of methane should be like the following (as presented in `examples/methane/methane.itp`):

```
[ atomtypes ]
;name btype  mass  charge ptype    sigma  epsilon
  c3    c3   0.0     0.0     A 0.339771 0.451035
  hc    hc   0.0     0.0     A 0.260018 0.087027

[ moleculetype ]
;name             nrexcl
 methane           3

[ atoms ]
; nr type  resnr residue atom  cgnr  charge   mass
```

(continues on next page)

(continued from previous page)

```
  1   c3      1     MOL   C1     1 -0.1068 12.010
  2   hc      1     MOL   H1     2  0.0267  1.008
  3   hc      1     MOL   H2     3  0.0267  1.008
  4   hc      1     MOL   H3     4  0.0267  1.008
  5   hc      1     MOL   H4     5  0.0267  1.008

[ bonds ]
; i  j  func  b0  kb
 1  2     5
 1  3     5
 1  4     5
 1  5     5

[ exclusions ]
; ai  aj1  aj2  aj3  aj4
  1    2    3    4    5
  2    1    3    4    5
  3    1    2    4    5
  4    1    2    3    5
  5    1    2    3    4
```

For comparison, the original topology file generated by `acpype` will be:

```
; methane_GMX.itp created by acpype (v: 2021-02-05T22:15:50CET) on Wed Sep  8 01:21:53 2021

[ atomtypes ]
;name   bond_type     mass     charge   ptype   sigma         epsilon       Amb
 c3       c3          0.00000  0.00000   A     3.39771e-01   4.51035e-01 ; 1.91  0.1078
 hc       hc          0.00000  0.00000   A     2.60018e-01   8.70272e-02 ; 1.46  0.0208

[ moleculetype ]
;name            nrexcl
 methane          3

[ atoms ]
;   nr  type  resi  res  atom  cgnr     charge      mass       ; qtot   bond_type
     1   c3     1   MOL    C1    1    -0.106800     12.01000 ; qtot -0.107
     2   hc     1   MOL    H1    2     0.026700      1.00800 ; qtot -0.080
     3   hc     1   MOL    H2    3     0.026700      1.00800 ; qtot -0.053
     4   hc     1   MOL    H3    4     0.026700      1.00800 ; qtot -0.027
     5   hc     1   MOL    H4    5     0.026700      1.00800 ; qtot 0.000

[ bonds ]
;   ai     aj funct   r             k
     1      2   1    1.0970e-01    3.1455e+05 ;     C1 - H1
     1      3   1    1.0970e-01    3.1455e+05 ;     C1 - H2
     1      4   1    1.0970e-01    3.1455e+05 ;     C1 - H3
     1      5   1    1.0970e-01    3.1455e+05 ;     C1 - H4

[ angles ]
;   ai     aj     ak    funct   theta         cth
     2      1      3      1    1.0758e+02    3.2635e+02 ;     H1 - C1     - H2
     2      1      4      1    1.0758e+02    3.2635e+02 ;     H1 - C1     - H3
     2      1      5      1    1.0758e+02    3.2635e+02 ;     H1 - C1     - H4
     3      1      4      1    1.0758e+02    3.2635e+02 ;     H2 - C1     - H3
     3      1      5      1    1.0758e+02    3.2635e+02 ;     H2 - C1     - H4
     4      1      5      1    1.0758e+02    3.2635e+02 ;     H3 - C1     - H4
```

DeepMD Settings

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

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

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

```
{
    "graph_file": "/path/to/graph.pb",
    "type_file": "type.raw",
    "index_file": "index.raw",
    "lambda": 1.0,
    "pbc": false
}
```

Here is an explanation for these settings:

- `graph_file` : The graph file (with suffix .pb) generated by `dp freeze` command
- `type_file` : File to specify DP atom types (in space-separated format). Here, `type.raw` looks like

```
1 0 0 0 0
```

- `index_file` : File containing indices of DP atoms (in space-separated format), which should be consistent with the indices' order in .gro file but starting from zero. Here, `index.raw` looks like

```
0 1 2 3 4
```

- `lambda`: Optional, default 1.0. Used in alchemical calculations.
- `pbc`: Optional, default true. If true, the GROMACS periodic condition is passed to DeepMD.

Run Simulation

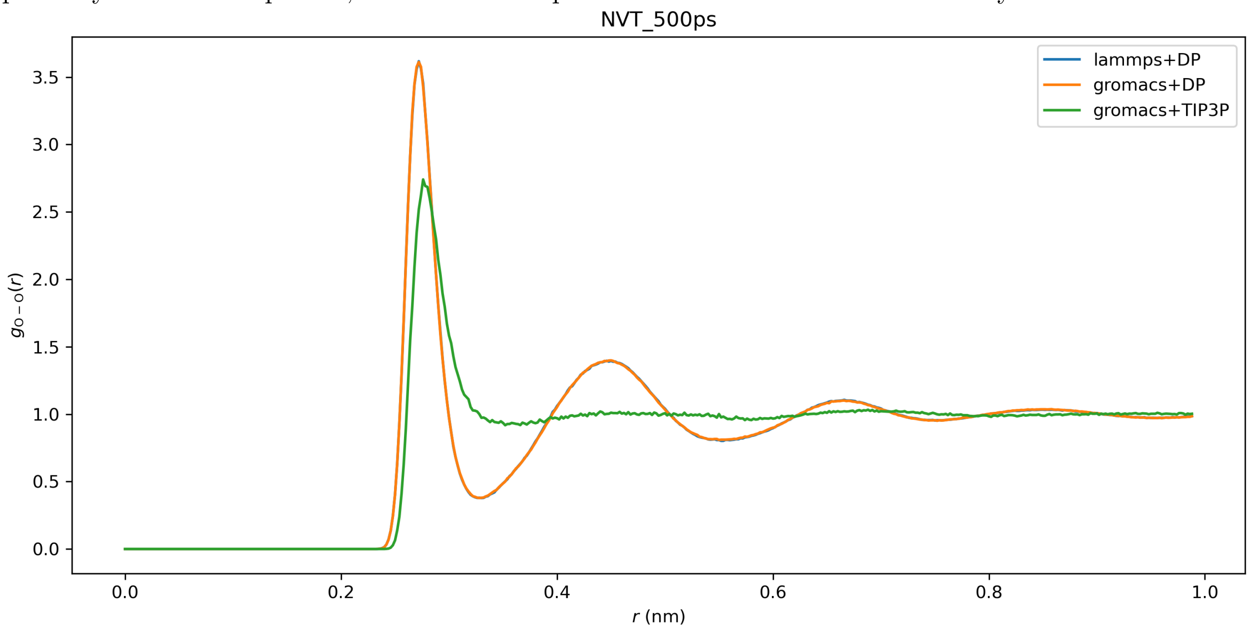
Finally, you can run GROMACS using `gmx mdrun` as usual.

10.5.2 All-atom DP Simulation

This part gives an example of how to simulate all atoms described by a DeepPotential with Gromacs, taking water as an example. Instead of using `[ exclusions ]` to turn off the non-bonded energies, we can simply do this by setting LJ parameters (i.e. epsilon and sigma) and partial charges to 0, as shown in `examples/water/gmx/water.top`:

```
[ atomtypes ]
; name      at.num  mass     charge ptype  sigma      epsilon
HW           1       1.008   0.0000  A    0.00000e+00  0.00000e+00
OW           8      16.00    0.0000  A    0.00000e+00  0.00000e+00
```

As mentioned in the above section, `input.json` and relevant files (`index.raw`, `type.raw`) should also be created. Then, we can start the simulation under the NVT ensemble and plot the radial distribution function (RDF) by `gmx rdf` command. We can see that the RDF given by Gromacs+DP matches perfectly with Lammps+DP, which further provides an evidence on the validity of our simulation.



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

10.6 Interfaces out of DeePMD-kit

The codes of the following interfaces are not a part of the DeePMD-kit package and maintained by other repositories. We list these interfaces here for user convenience.

10.6.1 dpdata

dpdata provides the `predict` method for `System` class:

```
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.

10.6.2 OpenMM plugin for DeePMD-kit

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

10.6.3 AMBER interface to DeePMD-kit

An AMBER interface to DeePMD-kit is written by the [York Lab from Rutgers University. It is open-source at GitLab RutgersLBSR/AmberDPRc. Details can be found in this paper.

10.6.4 DP-GEN

DP-GEN provides a workflow to generate accurate DP models by calling DeePMD-kit's command line interface (CLI) in the local or remote server. Details can be found in this paper.

10.6.5 MLatom

Mlatom provides an interface to the DeePMD-kit within MLatom's workflow by calling DeePMD-kit's CLI. Details can be found in this paper.

10.6.6 ABACUS

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

CHAPTER

ELEVEN

USE NVNMD

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.

11.2 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

```
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.

11.2.1 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

```
{
    "net_size":128,
    "sel":[60, 60],
    "rcut":6.0,
    "rcut_smth":0.5
}
```

where items are defined as:

Item	Mean	Optional Value
net_size	the size of nueral network	128
sel	the number of neighbors	integer list of lengths 1 to 4 are acceptable
rcut	the cutoff radial	(0, 8.0]
rcut_smth	the smooth cutoff parameter	(0, 8.0]

learning_rate

The “learning_rate” section is defined as

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

where items are defined as:

Item	Mean	Optional Value
type	learning rate variant type	exp
start_lr	the learning rate at the beginning of the training	a positive real number
stop_lr	the desired learning rate at the end of the training	a positive real number
decay_stops	the learning rate is decaying every {decay_stops} training steps	a positive integer

loss

The “loss” section is defined as

```
{
    "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:

Item	Mean	Optional Value
start_pref_e	the loss factor of energy at the beginning of the training	zero or positive real number
limit_pref_e	the loss factor of energy at the end of the training	zero or positive real number
start_pref_f	the loss factor of force at the beginning of the training	zero or positive real number
limit_pref_f	the loss factor of force at the end of the training	zero or positive real number
start_pref_v	the loss factor of virial at the beginning of the training	zero or positive real number
limit_pref_v	the loss factor of virial at the end of the training	zero or positive real number

training

The “training” section is defined as

```
{
  "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:

Item	Mean	Optional Value
seed	the randome seed	a integer
stop_batch	the total training steps	a positive integer
numb_test	the accuracy is test by using {numb_test} sample	a positive integer
disp_file	the log file where the training message display	a string
disp_freq	display frequency	a positive integer
save_ckpt	check point file	a string
save_freq	save frequency	a positive integer
systems	a list of data directory which contains the dataset	string list
set_prefix	the prefix of dataset	a string
batch_size	a list of batch size of corresponding dataset	a integer list

11.2.2 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: `nvnmd_cnn` and `nvnmd_qnn`. The `nvnmd_cnn` contains the model after continuous neural network (CNN) training. The `nvnmd_qnn` contains the model after quantized neural network (QNN) training. The binary file `nvnmd_qnn/model.pb` is the model file that is used to perform NVNMD in the server [http://nvnmd.picp.vip].

You can also restart the CNN training from the checkpoint (`nvnmd_cnn/model.ckpt`) by

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

11.3 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
```

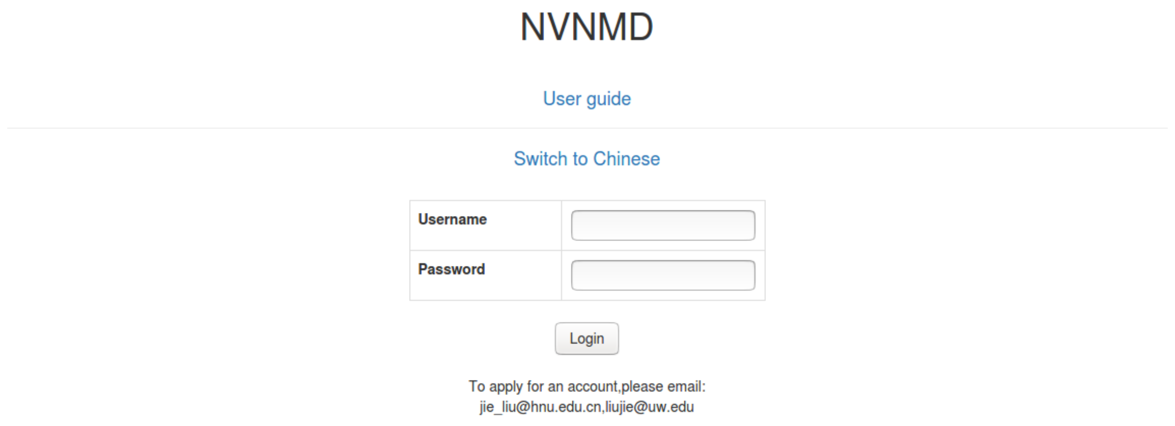
where the frozen model file to import is given via the `-m` command line flag, the path to the testing data set is given via the `-s` command line flag, and the file containing details of energy, forces and virials accuracy is given via the `-d` command line flag, the amount of data for testing is given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

11.4 Running MD

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 (Figure.1).



NVNMD

User guide

Switch to Chinese

Username	
Password	

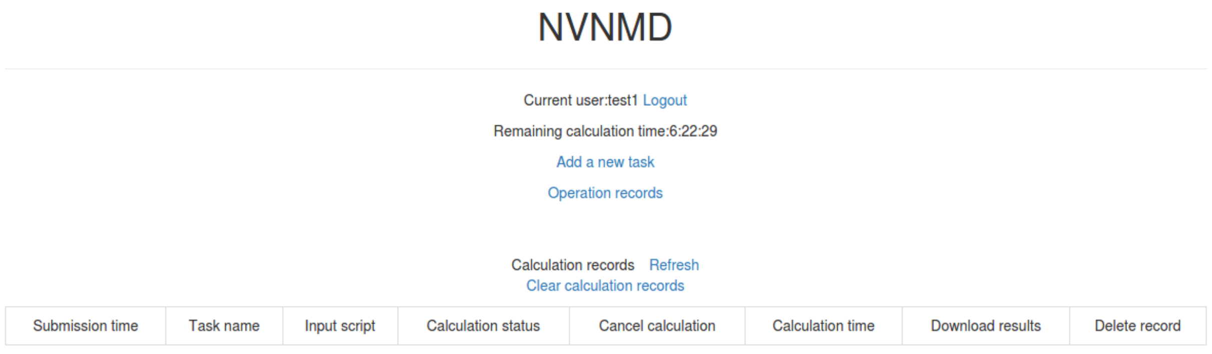
Login

To apply for an account,please email:
jie_liu@hnu.edu.cn,liujie@uw.edu

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 (Figure.2).



NVNMD

Current user:test1 Logout

Remaining calculation time:6:22:29

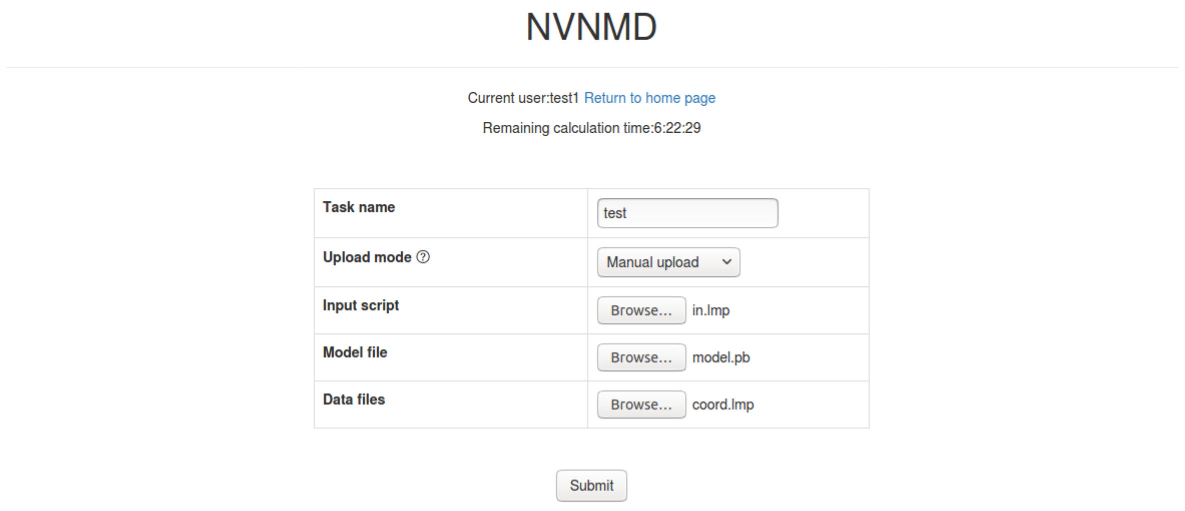
Add a new task

Operation records

Calculation records Refresh
Clear calculation records

Submission time	Task name	Input script	Calculation status	Cancel calculation	Calculation time	Download results	Delete record

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 (Figure.3).



NVNMD

Current user:test1 Return to home page

Remaining calculation time:6:22:29

Task name	test
Upload mode	Manual upload
Input script	Browse... in.lmp
Model file	Browse... model.pb
Data files	Browse... coord.lmp

Submit

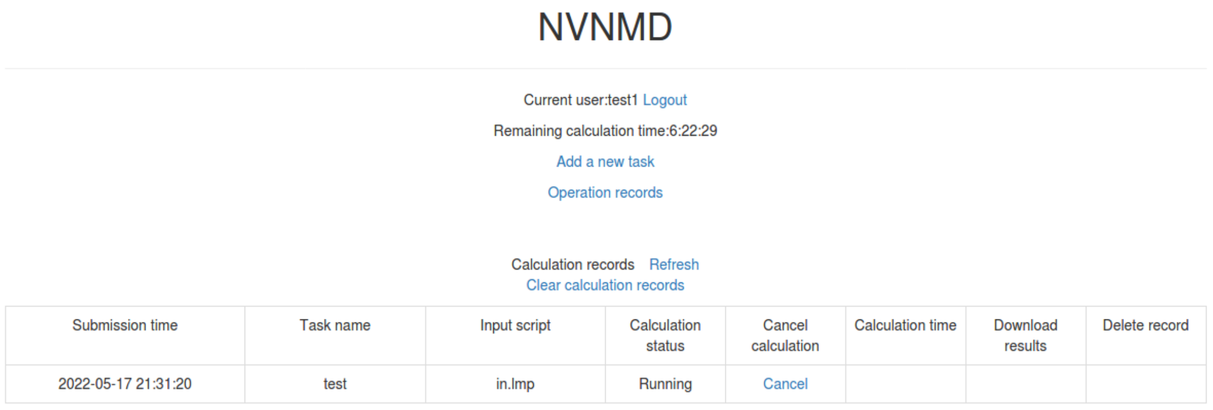
- Task name: name of the task
- Upload mode: two modes of uploading results to online data storage, including `Manual upload` and `Automatic upload`. Results need to be uploaded manually to online data storage with `Manual upload` mode and will be uploaded automatically with `Automatic upload` mode.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

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

```
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 (Figure.4).



NVNMD

Current user:test1 Logout

Remaining calculation time:6:22:29

Add a new task

Operation records

Calculation records Refresh

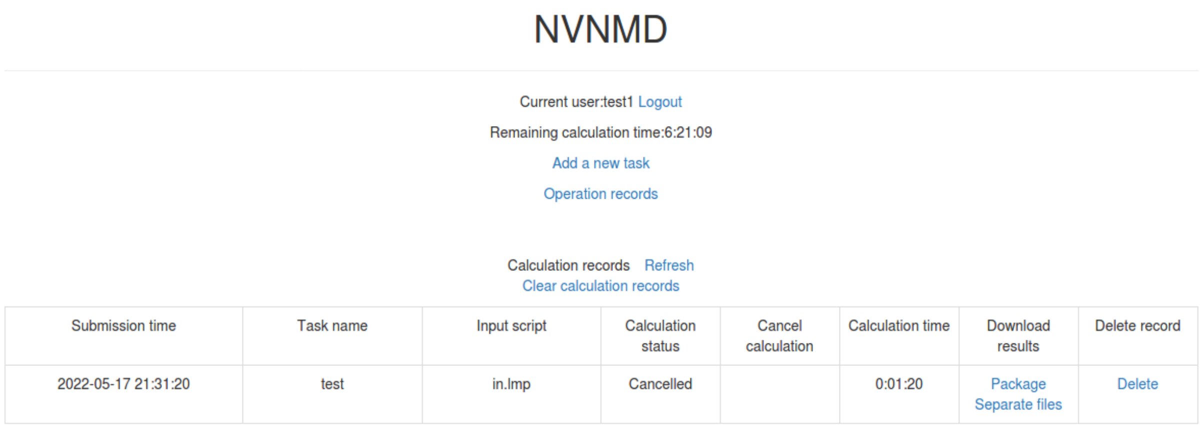
Clear calculation records

Submission time	Task name	Input script	Calculation status	Cancel calculation	Calculation time	Download results	Delete record
2022-05-17 21:31:20	test	in.lmp	Running	Cancel			

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 (Figure.5).



NVNMD

Current user:test1 Logout

Remaining calculation time:6:21:09

Add a new task

Operation records

Calculation records Refresh

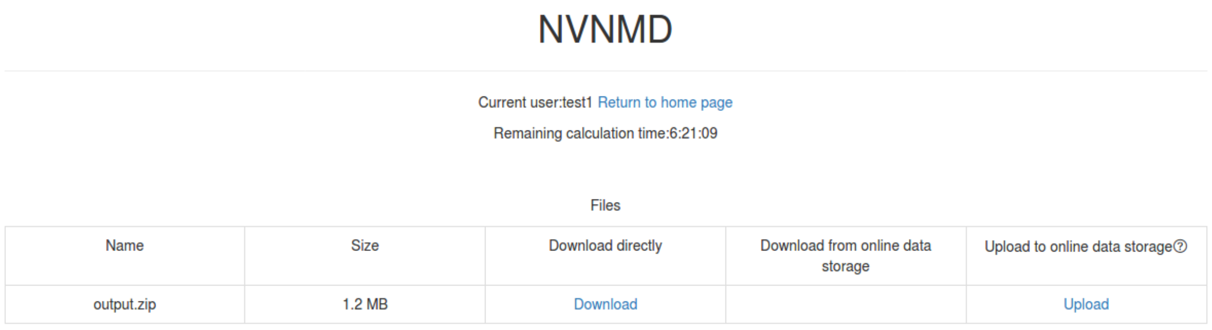
Clear calculation records

Submission time	Task name	Input script	Calculation status	Cancel calculation	Calculation time	Download results	Delete record
2022-05-17 21:31:20	test	in.lmp	Cancelled		0:01:20	Package Separate files	Delete

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 (Figure.6).



NVNMD

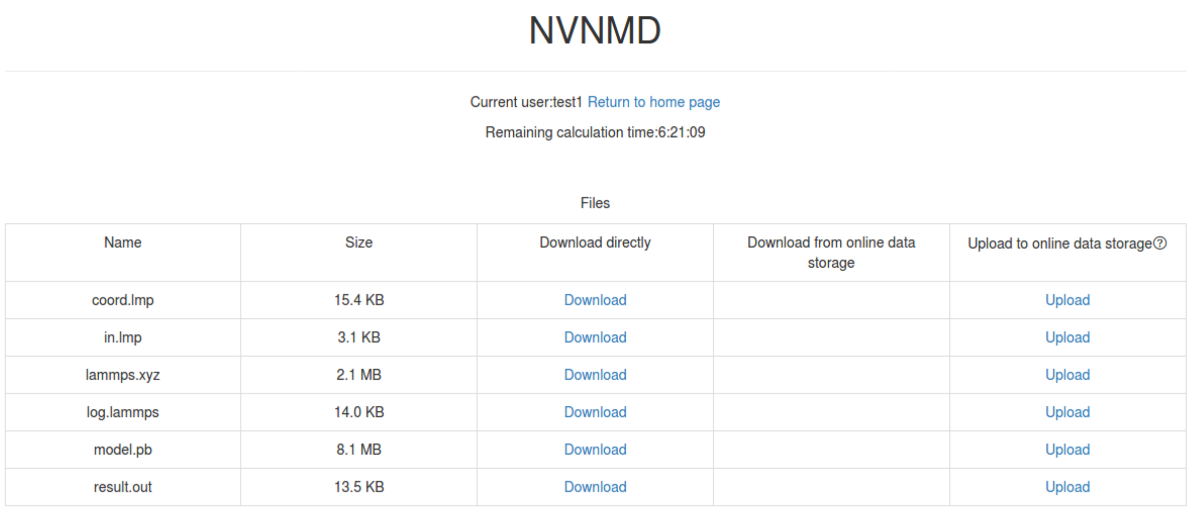
Current user:test1 Return to home page

Remaining calculation time:6:21:09

Files

Name	Size	Download directly	Download from online data storage	Upload to online data storage
output.zip	1.2 MB	Download		Upload

Click `Separate files` to download the required separate files (Figure.7).


NVNMD

Current user:test1 Return to home page

Remaining calculation time:6:21:09

Files

Name	Size	Download directly	Download from online data storage	Upload to online data storage
coord.lmp	15.4 KB	Download		Upload
in.lmp	3.1 KB	Download		Upload
lammps.xyz	2.1 MB	Download		Upload
log.lammps	14.0 KB	Download		Upload
model.pb	8.1 MB	Download		Upload
result.out	13.5 KB	Download		Upload

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.

CHAPTER

TWELVE

FAQS

As a consequence of differences in computers or systems, problems may occur. Some common circumstances are listed as follows. In addition, some frequently asked questions are listed as follows. If other unexpected problems occur, you're welcome to contact us for help.

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])

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

[] means no hidden layer, but there is still a linear output layer. This situation is equal to the linear regression.

Embedding net size tuning form on Al2O3: (Fitting-net size: [240,240,240])

Embedding-net size	Energy L2err(eV)	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[25,50,100]	1.742252e-02	7.259383e-05	4.014115e-02
[10,20,40]	2.909990e-02	1.212496e-04	4.734667e-02
[5,10,20]	3.357767e-02	1.399070e-04	5.706385e-02
[4,8,16]	6.060367e-02	2.525153e-04	7.333304e-02
[3,6,12]	5.656043e-02	2.356685e-04	7.793539e-02
[2,4,8]	5.277023e-02	2.198759e-04	7.459995e-02
[1,2,4]	1.302282e-01	5.426174e-04	9.672238e-02

12.1.2 Cu

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

Fitting-net size	Energy L2err(eV)	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[240,240,240]	4.135548e-02	1.615449e-04	8.940946e-02
[20,20,20]	4.323858e-02	1.689007e-04	8.955762e-02
[10,10,10]	4.399364e-02	1.718502e-04	8.962891e-02
[5,5,5]	4.468404e-02	1.745470e-04	8.970111e-02
[4,4,4]	4.463580e-02	1.743586e-04	8.972011e-02
[3,3,3]	4.493758e-02	1.755374e-04	8.971303e-02
[2,2,2]	4.500736e-02	1.758100e-04	8.973878e-02
[1,1,1]	4.542073e-02	1.774247e-04	8.964761e-02
[]	4.545168e-02	1.775456e-04	8.983201e-02

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

Embedding-net size	Energy L2err(eV)	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[25,50,100]	4.135548e-02	1.615449e-04	8.940946e-02
[20,40,80]	4.203562e-02	1.642016e-04	8.925881e-02
[15,30,60]	4.146672e-02	1.619794e-04	8.936911e-02
[10,20,40]	4.263060e-02	1.665258e-04	8.955818e-02
[5,10,20]	4.994913e-02	1.951138e-04	9.007786e-02
[4,8,16]	1.022157e-01	3.992802e-04	9.532119e-02
[3,6,12]	1.362098e-01	5.320695e-04	1.073860e-01
[2,4,8]	7.061800e-02	2.758515e-04	9.126418e-02
[1,2,4] && seed = 1	9.843161e-02	3.844985e-04	9.348505e-02
[1,2,4] && seed = 2	9.404335e-02	3.673568e-04	9.304089e-02
[1,2,4] && seed = 3	1.508016e-01	5.890688e-04	1.382356e-01
[1,2,4] && seed = 4	9.686949e-02	3.783965e-04	9.294820e-02

12.1.3 Water

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

Fitting-net size	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[240,240,240]	9.1589E-04	5.1540E-02
[200,200,200]	9.3221E-04	5.2366E-02
[160,160,160]	9.4274E-04	5.3403E-02
[120,120,120]	9.5407E-04	5.3093E-02
[80,80,80]	9.4605E-04	5.3402E-02
[40,40,40]	9.8533E-04	5.5790E-02
[20,20,20]	1.0057E-03	5.8232E-02
[10,10,10]	1.0466E-03	6.2279E-02
[5,5,5]	1.1154E-03	6.7994E-02
[4,4,4]	1.1289E-03	6.9613E-02
[3,3,3]	1.2368E-03	7.9786E-02
[2,2,2]	1.3558E-03	9.7042E-02
[1,1,1]	1.4633E-03	1.1265E-01
[]	1.5193E-03	1.2136E-01

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

Embedding-net size	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[25,50,100]	9.1589E-04	5.1540E-02
[20,40,80]	9.5080E-04	5.3593E-02
[15,30,60]	9.7996E-04	5.6338E-02
[10,20,40]	1.0353E-03	6.2776E-02
[5,10,20]	1.1254E-03	7.3195E-02
[4,8,16]	1.2495E-03	8.0371E-02
[3,6,12]	1.3604E-03	9.9883E-02
[2,4,8]	1.4358E-03	9.7389E-02
[1,2,4]	2.1765E-03	1.7276E-01

12.1.4 Mg-Al

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

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

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

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

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 -h` 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.

12.2.4 Tune the performance

There is no one general parallel configuration that works for all situations, so you are encouraged to tune parallel configurations yourself after empirical testing.

Here are some empirical examples. If you wish to use 3 cores of 2 CPUs on one node, you may set the environmental variables and run DeePMD-kit as follows:

```
export OMP_NUM_THREADS=3
export 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:

```
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 rcut < half boxsize?

When seeking the neighbors of atom i under periodic boundary conditions, DeePMD-kit considers all j atoms within cutoff rcut from atom i in all mirror cells.

So, there is no limitation on the setting of rcut.

PS: The reason why some software requires rcut < half box size is that they only consider the nearest mirrors from the center cell. DeePMD-kit is different from them.

12.4 How to set sel?

`sel` is short for "selected number of atoms in `rcut`".

`sel_a[i]` is a list of integers. The length of the list should be the same as the number of atom types in the system.

`sel_a[i]` gives the number of the selected number of type `i` neighbors within `rcut`. To ensure that the results are strictly accurate, `sel_a[i]` should be larger than the largest number of type `i` neighbors in the `rcut`.

However, the computation overhead increases with `sel_a[i]`, therefore, `sel_a[i]` should be as small as possible.

The setting of `sel_a[i]` should balance the above two considerations.

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.

Model version	v0.12	v1.0	v1.1	v1.2	v1.3	v2.0	v2.1	v2.2
Compatibility	😊	😊	😊	😊	😊	😄	😄	😄

Legend:

- 😄: The model is compatible with the DeePMD-kit package.
- 😊: The model is incompatible with the DeePMD-kit package, but one can execute `dp convert-from` to convert an old model to v2.2.
- 😢: The model is incompatible with the DeePMD-kit package, and there is no way to convert models.

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 evaluting the model deviation with multiple models. If the model deviation of these data is high for some data, try to collect more data using DP-GEN.

Values of data

One should be aware that the errors of some data is also affected by the absolute values of this data. Stable structures tend to be more precise than unstable structures because unstable structures may have larger forces. Also, errors will be introduced in the Projector augmented wave (PAW) DFT calculations when the atoms are very close due to the overlap of pseudo-potentials. It is expected to see that data with large forces has larger errors and it is better to compare different models only with the same data.

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.

CHAPTER

THIRTEEN

FIND DEEPMD-KIT C/C++ LIBRARY FROM CMAKE

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

```
find_package(DeePMD REQUIRED)
```

Note that you may need to add ${deepmd_root} to the cached CMake variable `CMAKE_PREFIX_PATH`.

To link against the C interface library, using

```
target_link_libraries(some_library PRIVATE DeePMD::deepmd_c)
```

To link against the C++ interface library, using

```
target_link_libraries(some_library PRIVATE DeePMD::deepmd_cc)
```

CHAPTER

FOURTEEN

CODING CONVENTIONS

14.1 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 GitHub actions CI checks for python code style which will annotate the PR diff for you to see the areas where your code is lacking compared to the set standard.

14.2 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

```
f"something {'this' if x else 'that'}"
```

- Use f-strings `s = f"{x:.2f}"` instead of old style formating with `"%f" % x`. string format method `"{x:.2f}".format()` may be used sparsely where it is more convenient than f-strings.

14.3 Whitespace

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

- Read the Whitespace in Expressions and Statements section of PEP8.
- Avoid trailing whitespaces.
- Do not use excessive whitespace in your expressions and statements.
- You should have blank spaces after commas, colons, and semi-colons if it isn't trailing next to the end of a bracket, brace, or parentheses.
- With any operators you should use space on both sides of the operator.
- Colons for slicing are considered a binary operator, and should not have any spaces between them.
- You should have parentheses with no space, directly next to the function when calling functions `function()`.
- When indexing or slicing the brackets should be directly next to the collection with no space `collection["index"]`.
- Whitespace used to line up variable values is not recommended.
- Make sure you are consistent with the formats you choose when optional choices are available.

14.4 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.

14.5 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

It is a good practice to run pydocstyle check on your code or use a text editor that does it automatically):

```
$ pydocstyle filename.py
```

14.6 Run pycodestyle on your code

It's a good idea to run pycodestyle on your code (or use a text editor that does it automatically):

```
$ pycodestyle filename.py
```

14.7 Run mypy on your code

It's a good idea to run mypy on your code (or use a text editor that does it automatically):

```
$ mypy filename.py
```

14.8 Run pydocstyle on your code

It's a good idea to run pycodestyle on your code (or use a text editor that does it automatically):

```
$ pycodestyle filename.py --max-line-length=88
```

14.9 Run black on your code

Another method of enforcing PEP8 is using a tool such as black. These tools tend to be very effective at cleaning up code but should be used carefully and code should be retested after cleaning it. Try:

```
$ black --help
```

CHAPTER

FIFTEEN

CREATE A MODEL

If you'd like to create a new model that isn't covered by the existing DeePMD-kit library, but reuse DeePMD-kit's other efficient modules such as data processing, trainner, etc, you may want to read this section.

To incorporate your custom model you'll need to:

1. Register and implement new components (e.g. descriptor) in a Python file. You may also want to register new TensorFlow OPs if necessary.
2. Register new arguments for user inputs.
3. Package new codes into a Python package.
4. Test new models.

15.1 Design a new component

When creating a new component, take descriptor as the example, 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 SomeDescript(Descriptor):
    def __init__(self, arg1: bool, arg2: float) -> None:
        pass
```

15.2 Register new arguments

To let someone uses your new component in their input file, you need to create a new method that returns some `Argument` of your new component, and then register new arguments. For example, the code below

```python
from typing import List

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

```

(continues on next page)

(continued from previous page)

```
@descrpt_args_plugin.register("some_descrpt")
def descrpt_some_args() -> List[Argument]:
    return [
        Argument("arg1", bool, optional=False, doc="balabala"),
        Argument("arg2", float, optional=True, default=6.0, doc="haha"),
    ]
```

allows one to use your new descriptor as below:

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

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

15.3 Package new codes

You may use `setuptools` to package new codes into a new Python package. It's crucial to add your new component to `entry_points['deepmd']` in `setup.py`:

```
    entry_points={
        'deepmd': [
            'some_descrpt=deepmd_some_descrtpt:SomeDescript',
        ],
    },
```

where `deepmd_some_descrtpt` is the module of your codes. It is equivalent to `from deepmd_some_descrtpt import SomeDescript`.

If you place `SomeDescript` and `descrpt_some_args` into different modules, you are also expected to add `descrpt_some_args` to `entry_points`.

After you install your new package, you can now use `dp train` to run your new model.

CHAPTER

SIXTEEN

ATOM TYPE EMBEDDING

16.1 Overview

Here is an overview of the DeePMD-kit algorithm. Given a specific centric atom, we can obtain the matrix describing its local environment, named $\mathcal{R}$. It is consist of the distance between the centric atom and its neighbors, as well as a direction vector. We can embed each distance into a vector of M_1 dimension by an `embedding net`, so the environment matrix $\mathcal{R}$ can be embedded into matrix $\mathcal{G}$. We can thus extract a descriptor vector (of $M_1 \times M_2$ dim) of the centric atom from the $\mathcal{G}$ by some matrix multiplication, and put the descriptor into `fitting net` to get 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 decline training complexity largely.

16.2 Preliminary

In the following chart, you can find the meaning of symbols used to clarify the atom-type embedding algorithm.

i: Type of centric atom

j: Type of neighbor atom

s_{ij}: Distance between centric atom and neighbor atom

$\mathcal{G}_{ij}(\cdot)$: Origin embedding net, take s_{ij} as input and output embedding vector of M_1 dim

$\mathcal{G}(\cdot)$: Shared embedding net

Multi$(\cdot)$: Matrix multiplication and flattening, output the descriptor vector of $M_1 \times M_2$ dim

$F_i(\cdot)$: Origin fitting net, take the descriptor vector as input and output energy

$F(\cdot)$: Shared fitting net

$A(\cdot)$: Atom type embedding net, input is atom type, the output is type embedding vector of dim `nchanl`

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

$$E = F_i(\text{Multi}(\mathcal{G}_{ij}(s_{ij})))$$

DeePMD-kit applying atom type embedding:

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

or

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

The difference between the two variants above is whether using the information of centric atom when generating the descriptor. Users can choose by modifying the `type_one_side` hyper-parameter in the input JSON file.

16.3 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
    }
```

16.4 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.

16.4.1 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`.

16.4.2 model (model/ener.py)

When building the operation graph of the `model` in `model.build`. If a `TypeEmbedNet` is detected, it will build the operation graph of `type embed net`, `embedding net` and `fitting net` by order. The building process of `type embed net` can be found in `TypeEmbedNet.build`, which output the type embedding vector of each atom type (of [ntypes $\times$ nchanl] dimensions). We then save the type embedding vector into `input_dict`, so that they can be fetched later in `embedding net` and `fitting net`.

16.4.3 embedding net (descriptor/se*.py)

In `embedding net`, we shall take local environment $\mathcal{R}$ as input and output matrix $\mathcal{G}$. Functions called in this process by the order is

```
build -> _pass_filter -> _filter -> _filter_lower
```

`_pass_filter`: It will first detect whether an atom type embedding exists, if so, it will apply atom type embedding algorithm and doesn't divide the input by type.

`_filter`: It will call `_filter_lower` function to obtain the result of matrix multiplication ($\mathcal{G}^T \cdot \mathcal{R}$), do further multiplication involved in Multi($\cdot$), and finally output the result of descriptor vector of $M_1 \times M_2$ dim.

`_filter_lower`: The main function handling input modification. If type embedding exists, it will call `_concat_type_embedding` function to concat the first column of input $\mathcal{R}$ (the column of s_{ij}) with the atom type embedding information. It will decide whether to use the atom type embedding vector of the centric atom according to the value of `type_one_side` (if set True, then we only use the vector of the neighbor atom). The modified input will be put into the `fitting net` to get $\mathcal{G}$ for further matrix multiplication stage.

16.4.4 fitting net (fit/ener.py)

In `fitting net`, it takes the descriptor vector as input, whose dimension is [natoms, $M_1 \times M_2$]. Because we need to involve information on the centric atom in this step, we need to generate a matrix named `atype_embed` (of dim [natoms, nchanl]), in which each row is the type embedding vector of the specific centric atom. The input is sorted by type of centric atom, we also know the number of a particular atom type (stored in `natoms[2+i]`), thus we get the type vector of the centric atom. In the build phase of the fitting net, it will check whether type embedding exists in `input_dict` and fetch them. After that, call `embed_atom_type` function to look up the embedding vector for the type vector of the centric atom to obtain `atype_embed`, and concat input with it ([input, atype_embed]). The modified input goes through `fitting` net` to get predicted energy.

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

CHAPTER

SEVENTEEN

PYTHON API

17.1 deepmd package

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

class deepmd.DeepEval(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False, auto_batch_size: Union[bool, int, AutoBatchSize] = False)

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.

Attributes

`model_type`
Get type of model.

`model_version`
Get version of model.

`sess`
Get TF session.

Methods

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.

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:

```
>>> 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_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

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

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.

`deepmd.DeepPotential`(model_file: Union[str, Path], load_prefix: str = 'load', default_tf_graph: bool = False) → Union[DeepDipole, DeepGlobalPolar, DeepPolar, DeepPot, DeepDOS, DeepWFC]

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

Returns

`Union`[**`DeepDipole`**, **`DeepGlobalPolar`**, **`DeepPolar`**, **`DeepPot`**, **`DeepWFC`**]
one of the available potentials

Raises

`RuntimeError`
if model file does not correspond to any implementd 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: A

ewald_beta
Splitting parameter of the Ewald sum. Unit: A^{-1}

Attributes

model_type
Get type of model.

model_version
Get version of model.

sess
Get TF session.

Methods

build_fv_graph()	Build the computational graph for the force and virial inference.
eval(coord, box, atype[, eval_fv])	Evaluate the modification.
`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.
modify_data(data, data_sys)	Modify data.
`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_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

load_prefix: `str`

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.

17.1.1 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 sa 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 $\mathcal{D}$ describes the environment of an atom, which should be a function of coordinates and types of its neighbour atoms.

Notes

Only methods and attributes defined in this class are generally public, that can be called by other classes.

Examples

```
>>> descript = Descriptor(type="se_e2_a", rcut=6., rcut_smth=0.5, sel=[50])
>>> type(descript)
<class 'deepmd.descriptor.se_a.DescrptSeA'>
```

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 statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec])	Reveive 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_feed_dict(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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.

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 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[R08579741114c-1]_, i.e.

$$\forall i \in \text{type } 1, j \in \text{type } 2, \frac{\partial E}{\partial r_{ij}} = 0$$

When embedding networks between every two types are built, we can just remove that network. But when type_one_side is enabled, a network may be built for multiple pairs of types. In this case, we need to build a mask to exclude the interaction between two types.

The mask assumes the descriptors are sorted by neighbro type with the fixed number of given sel and each neighbor has the same number of descriptors (for example 4).

References

[1]

abstract 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]]) → None

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

Parameters

data_coord
[`list`[`np.ndarray`]] The coordinates. Can be generated by *`deepmd.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

Notes

This method must be implemented, as it's called by other classes.

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 statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
[`float`] The nearest distance between atoms

graph
[`tf.Graph`] The graph of the model

graph_def
[`tf.GraphDef`] The graph definition of the model

table_extrapolate
[`float`, default: 5.] The scale of model extrapolation

table_stride_1
[float, default: 0.01] The uniform stride of the first table

table_stride_2
[float, default: 0.1] The uniform stride of the second table

check_frequency
[int, default: -1] The overflow check frequency

suffix
[str, optional] The suffix of the scope

Notes

This method is called by others when the descriptor supported compression.

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.

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_feed_dict(coord_: Tensor, atype_: Tensor, natoms: Tensor, box: Tensor, mesh: Tensor) → 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.

Returns

feed_dict
[dict[str, tf.Tensor]] The output feed_dict of current descriptor

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: 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

Notes

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

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()*.

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

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

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.

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 statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec])	Reveive 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_feed_dict(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
get_nlist()	Get the neighbor information of the descriptor, returns the nlist of the descriptor with the largest cut-off radius.
get_nlist_i(ii)	Get the neighbor information of the ii-th descriptor.
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.
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(*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.

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: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict) → None

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

Parameters

data_coord
The coordinates. Can be generated by deepmd.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

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 statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
[`float`] The nearest distance between atoms

graph
[`tf.Graph`] The graph of the model

graph_def
[`tf.GraphDef`] The graph_def of the model

table_extrapolate
[`float`, default: 5.] The scale of model extrapolation

table_stride_1
: [`float`, default: 0.01] The uniform stride of the first table

table_stride_2
: [`float`, default: 0.1] The uniform stride of the second table

check_frequency
: [`int`, default: -1] The overflow check frequency

suffix
: [`str`, `optional`] The suffix of the scope

`enable_mixed_precision`(mixed_prec: Optional[dict] = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
: The mixed precision setting used in the embedding net

`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 statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict

The dict of statisitcs computed from compute_input_stats, including:

sumr
The sum of radial statisitcs.

suma
The sum of relative coord statisitcs.

sumn
The sum of neighbor numbers.

sumr2
The sum of square of radial statisitcs.

suma2
The sum of square of relative coord statisitcs.

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

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

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.

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 statisitcs (avg and std) of the training data.
`enable_compression`(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
`enable_mixed_precision`([mixed_prec])	Reveive 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_feed_dict`(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
get_nlist()	Returns
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.
`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.

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: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict) → None

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

Parameters

data_coord
: The coordinates. Can be generated by deepmd.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

`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

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

$$(\mathcal{R}^i)_j = [\begin{array}{c} s(r_{ji}) \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{array}]$$

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

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

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

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}^i_< \in \mathbb{R}^{N \times M_2}$ takes first M_2 columns of $\mathcal{G}^i$. The equation of embedding network $\mathcal{N}$ can be found at *deepmd.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 maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are "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

precision
: Precision of filter network.

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 statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec])	Reveive 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_feed_dict(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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.

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: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict) → None

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

Parameters

data_coord
: The coordinates. Can be generated by deepmd.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

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 statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
: The nearest distance between atoms

graph
: [`tf.Graph`] The graph of the model

graph_def
: [`tf.GraphDef`] The graph_def of the model

table_extrapolate
: The scale of model extrapolation

table_stride_1
: The uniform stride of the first table

table_stride_2
: The uniform stride of the second table

check_frequency
: The overflow check frequency

suffix
: [`str`, `optional`] The suffix of the scope

`enable_mixed_precision`(mixed_prec: Optional[dict] = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
: The mixed precision setting used in the embedding net

`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

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 statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict

The dict of statisitcs computed from compute_input_stats, including:

sumr
The sum of radial statisitcs.

suma
The sum of relative coord statisitcs.

sumn
The sum of neighbor numbers.

sumr2
The sum of square of radial statisitcs.

suma2
The sum of square of relative coord statisitcs.

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
[list[str]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
[list[int]] Number of neurons in each hidden layers of the embedding net

axis_neuron
Number of the axis neuron (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
If the weights of embedding net are trainable.

seed
Random seed for initializing the network parameters.

type_one_side
Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

type_nchanl
Number of channels for type representation

type_nlayer
Number of hidden layers for the type embedding net (skip connected).

numb_aparam
Number of atomic parameters. If >0 it will be embedded with atom types.

set_davg_zero
Set the shift of embedding net input to zero.

activation_function
The activation function in the embedding net. Supported options are {0}

precision
The precision of the embedding net parameters. Supported options are {1}

exclude_types
[List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

Attributes

precision
Precision of filter network.

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 statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec])	Reveive 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_feed_dict(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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.

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

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

Bases: *Descriptor*

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 maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net

axis_neuron
: Number of the axis neuron (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are “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

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 statisitcs (avg and std) of the training data.
`enable_compression`(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
`enable_mixed_precision`([mixed_prec])	Reveive 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_feed_dict`(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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.
`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.

`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. 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) → None

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

Parameters

data_coord
: The coordinates. Can be generated by deepmd.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

`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[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.**DescrptSeAEfLower**(*args, **kwargs)

Bases: *DescrptSeA*

Helper class for implementing DescrptSeAEf.

Attributes

precision
Precision of filter network.

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 statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec])	Reveive 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_feed_dict(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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.

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

Parameters

data_coord
The coordinates. Can be generated by deepmd.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

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 $\mathcal{D}^i \in \mathcal{R}^{M_1 \times M_2}$ is given by [1]_

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}^i_<$$

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

$$(\mathcal{R}^i)_j = [\begin{array}{c} s(r_{ji}) \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{array}]$$

where $\mathrm{R}_{ji} = \mathrm{R}_j - \mathrm{R}_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \lVert \mathrm{R}_{ji} \rVert$ is its norm. The switching function $s(r)$ is defined as:

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

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

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}^i_< \in \mathbb{R}^{N \times M_2}$ takes first M_2 columns of $\mathcal{G}^i$. The equation of embedding network $\mathcal{N}$ can be found at *deepmd.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 maxmum number of type i atoms in the neighbor list.

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

activation_function
: The activation function in the embedding net. Supported options are {0}

precision
: The precision of the embedding net parameters. Supported options are {1}

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

References

———-

.. [1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018.
: End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 4441–4451.

Attributes

precision
: Precision of filter network.

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 statisitcs (avg and std) of the training data.
`enable_compression`(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
`enable_mixed_precision`([mixed_prec])	Reveive 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_feed_dict`(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
`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 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.

`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[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) → None

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

Parameters

data_coord
The coordinates. Can be generated by deepmd.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

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

class deepmd.descriptor.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 maxmum number of type i atoms in the cut-off radius

neuron
[list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron
Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
If the weights of embedding net are trainable.

seed
Random seed for initializing the network parameters.

type_one_side
Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
[List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero
Set the shift of embedding net input to zero.

activation_function
The activation function in the embedding net. Supported options are "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

precision
Precision of filter network.

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 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, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec])	Reveive 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_feed_dict(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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.

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], ndescrpt: 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.

ndescrpt
: [`int`] The number of descriptors for each atom.

atype
: [`tf.Tensor`] The type of atoms, with the size of shape0.

shape0
: [`tf.Tensor`] The shape of the first dimension of the inputs, which is equal to nsamples * natoms.

nei_type_vec
: [`tf.Tensor`] The type of neighbors, with the size of (shape0, nnei).

Returns

`tf.Tensor`
: The type exclude mask, with the shape of (shape0, ndescrpt), and the precision of

GLOBAL_TF_FLOAT_PRECISION. The mask has the value of 1 if the interaction between two types is not excluded, and 0 otherwise.

See also:

deepmd.descriptor.descriptor.Descriptor.build_type_exclude_mask

Notes

This method has the similiar way to build the type exclude mask as *deepmd.descriptor.descriptor.Descriptor.build_type_exclude_mask()*. The mathmatical expression has been explained in that method. The difference is that the attention descriptor has provided the type of the neighbors (idx_j) that is not in order, so we use it from an extra input.

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) → None

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

Parameters

data_coord
The coordinates. Can be generated by deepmd.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_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.

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

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 maxmum number of type i atoms in the cut-off radius

neuron
[list[int]] Number of neurons in each hidden layers of the embedding net

resnet_dt
Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
If the weights of embedding net are trainable.

seed
Random seed for initializing the network parameters.

type_one_side
Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
[List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

activation_function
The activation function in the embedding net. Supported options are “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

precision
Precision of filter network.

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 statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
`enable_mixed_precision`([mixed_prec])	Reveive 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_feed_dict`(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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.
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.

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)

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

Parameters

data_coord
The coordinates. Can be generated by deepmd.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

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 statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
The nearest distance between atoms

graph
[`tf.Graph`] The graph of the model

graph_def
[`tf.GraphDef`] The graph_def of the model

table_extrapolate
The scale of model extrapolation

table_stride_1
The uniform stride of the first table

table_stride_2
The uniform stride of the second table

check_frequency
The overflow check frequency

suffix
[str, optional] The suffix of the scope

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 statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict

The dict of statisitcs computed from compute_input_stats, including:

sumr
The sum of radial statisitcs.

sumn
The sum of neighbor numbers.

sumr2
The sum of square of radial statisitcs.

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 maxmum number of type i atoms in the cut-off radius

neuron
[list[int]] Number of neurons in each hidden layers of the embedding net

resnet_dt
Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
If the weights of embedding net are trainable.

seed
Random seed for initializing the network parameters.

set_davg_zero
Set the shift of embedding net input to zero.

activation_function
The activation function in the embedding net. Supported options are "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

precision
Precision of filter network.

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 statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
`enable_mixed_precision`([mixed_prec])	Reveive 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_feed_dict`(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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.
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.

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: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict) → None

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

Parameters

data_coord
: The coordinates. Can be generated by deepmd.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

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 statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
: The nearest distance between atoms

graph
: [`tf.Graph`] The graph of the model

graph_def
: [`tf.GraphDef`] The graph_def of the model

table_extrapolate
: The scale of model extrapolation

table_stride_1
: The uniform stride of the first table

table_stride_2
: The uniform stride of the second table

check_frequency
The overflow check frequency

suffix
[`str`, `optional`] The suffix of the scope

`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 statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict

The dict of statisitcs computed from compute_input_stats, including:

sumr
The sum of radial statisitcs.

suma
The sum of relative coord statisitcs.

sumn
The sum of neighbor numbers.

sumr2
The sum of square of radial statisitcs.

suma2
The sum of square of relative coord statisitcs.

`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

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 $\mathcal{D}$ describes the environment of an atom, which should be a function of coordinates and types of its neighbour atoms.

Notes

Only methods and attributes defined in this class are generally public, that can be called by other classes.

Examples

```
>>> descript = Descriptor(type="se_e2_a", rcut=6., rcut_smth=0.5, sel=[50])
>>> type(descript)
<class 'deepmd.descriptor.se_a.DescrptSeA'>
```

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 statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec])	Reveive 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_feed_dict(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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.

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 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[Rafc1ae60e195-1]_, i.e.

$$\forall i \in \text{type } 1, j \in \text{type } 2, \frac{\partial E}{\partial r_{ij}} = 0$$

When embedding networks between every two types are built, we can just remove that network. But when type_one_side is enabled, a network may be built for multiple pairs of types. In this case, we need to build a mask to exclude the interaction between two types.

The mask assumes the descriptors are sorted by neighbro type with the fixed number of given sel and each neighbor has the same number of descriptors (for example 4).

References

[1]

abstract 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]]) → None

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

Parameters

data_coord
[`list`[`np.ndarray`]] The coordinates. Can be generated by *`deepmd.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

Notes

This method must be implemented, as it's called by other classes.

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 statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
[`float`] The nearest distance between atoms

graph
[`tf.Graph`] The graph of the model

graph_def
[`tf.GraphDef`] The graph definition of the model

table_extrapolate
[`float`, default: 5.] The scale of model extrapolation

table_stride_1
[`float`, default: 0.01] The uniform stride of the first table

table_stride_2
[`float`, default: 0.1] The uniform stride of the second table

check_frequency
[`int`, default: -1] The overflow check frequency

suffix
[`str`, `optional`] The suffix of the scope

Notes

This method is called by others when the descriptor supported compression.

`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.

`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_feed_dict`(coord_: Tensor, atype_: Tensor, natoms: Tensor, box: Tensor, mesh: Tensor) → 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.

Returns

feed_dict
: [`dict`[`str`, `tf.Tensor`]] The output feed_dict of current descriptor

`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: 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

Notes

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

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()*.

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

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

deepmd.descriptor.hybrid module

class deepmd.descriptor.hybrid.**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.

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 statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec])	Reveive 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_feed_dict(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
get_nlist()	Get the neighbor information of the descriptor, returns the nlist of the descriptor with the largest cut-off radius.
get_nlist_i(ii)	Get the neighbor information of the ii-th descriptor.
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.
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(*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.

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: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict) → None

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

Parameters

data_coord
The coordinates. Can be generated by deepmd.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

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 statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
[`float`] The nearest distance between atoms

graph
[`tf.Graph`] The graph of the model

graph_def
[`tf.GraphDef`] The graph_def of the model

table_extrapolate
[`float`, default: 5.] The scale of model extrapolation

table_stride_1
[`float`, default: 0.01] The uniform stride of the first table

table_stride_2
[`float`, default: 0.1] The uniform stride of the second table

check_frequency
[`int`, default: -1] The overflow check frequency

suffix
[`str`, `optional`] The suffix of the scope

`enable_mixed_precision`(mixed_prec: Optional[dict] = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
The mixed precision setting used in the embedding net

`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 statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict

The dict of statisitcs computed from compute_input_stats, including:

sumr
The sum of radial statisitcs.

suma
The sum of relative coord statisitcs.

sumn
The sum of neighbor numbers.

sumr2
The sum of square of radial statisitcs.

suma2
The sum of square of relative coord statisitcs.

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

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.

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 statisitcs (avg and std) of the training data.
`enable_compression`(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
`enable_mixed_precision`([mixed_prec])	Reveive 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_feed_dict`(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
get_nlist()	Returns
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.
`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.

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: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict) → None

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

Parameters

data_coord
: The coordinates. Can be generated by deepmd.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

`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

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 similiar methods without defining them twice.

Attributes

embedding_net_variables
[dict] initial embedding network variables

descrpt_reshape
[tf.Tensor] the reshaped descriptor

descrpt_deriv
[tf.Tensor] the descriptor derivative

rij
[tf.Tensor] distances between two atoms

nlist
[tf.Tensor] the neighbor list

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 statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec])	Reveive 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_feed_dict(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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.

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.

deepmd.descriptor.se_a module

class deepmd.descriptor.se_a.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 $\mathcal{R}^i \in \mathbb{R}^{N \times 4}$ is the coordinate matrix, and each row of $\mathcal{R}^i$ can be constructed as follows

$$(\mathcal{R}^i)_j = [\begin{array}{c} s(r_{ji}) \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{array}]$$

where $\mathrm{R}_{ji} = \mathrm{R}_j - \mathrm{R}_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \lVert \mathrm{R}_{ji} \rVert$ is its norm. The switching function $s(r)$ is defined as:

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

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

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}^i_< \in \mathbb{R}^{N \times M_2}$ takes first M_2 columns of $\mathcal{G}^i$. The equation of embedding network $\mathcal{N}$ can be found at *deepmd.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 maxmum number of type i atoms in the cut-off radius

neuron
: [`list`[`int`]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [`List`[`List`[`int`]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are "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

precision
: Precision of filter network.

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 statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec])	Reveive 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_feed_dict(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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.

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: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict) → None

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

Parameters

data_coord
The coordinates. Can be generated by deepmd.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

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 statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
: The nearest distance between atoms

graph
: [`tf.Graph`] The graph of the model

graph_def
: [`tf.GraphDef`] The graph_def of the model

table_extrapolate
: The scale of model extrapolation

table_stride_1
: The uniform stride of the first table

table_stride_2
: The uniform stride of the second table

check_frequency
: The overflow check frequency

suffix
: [`str`, `optional`] The suffix of the scope

`enable_mixed_precision`(mixed_prec: Optional[dict] = None) → None

Reveive the mixed precision setting.

Parameters

mixed_prec
: The mixed precision setting used in the embedding net

`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

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 statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict

The dict of statisitcs computed from compute_input_stats, including:

sumr
The sum of radial statisitcs.

suma
The sum of relative coord statisitcs.

sumn
The sum of neighbor numbers.

sumr2
The sum of square of radial statisitcs.

suma2
The sum of square of relative coord statisitcs.

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

deepmd.descriptor.se_a_ebd module

`class deepmd.descriptor.se_a_ebd.`**`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
  [`list`[`str`]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius

neuron
  [`list`[`int`]] Number of neurons in each hidden layers of the embedding net

axis_neuron
  Number of the axis neuron (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
  Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
  If the weights of embedding net are trainable.

seed
  Random seed for initializing the network parameters.

type_one_side
  Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

type_nchanl
  Number of channels for type representation

type_nlayer
  Number of hidden layers for the type embedding net (skip connected).

numb_aparam
  Number of atomic parameters. If >0 it will be embedded with atom types.

set_davg_zero
  Set the shift of embedding net input to zero.

activation_function
  The activation function in the embedding net. Supported options are {0}

precision
  The precision of the embedding net parameters. Supported options are {1}

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

Attributes

precision
: Precision of filter network.

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 statisitcs (avg and std) of the training data.
`enable_compression`(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
`enable_mixed_precision`([mixed_prec])	Reveive 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_feed_dict`(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
`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.

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_ef module

`class deepmd.descriptor.se_a_ef.`**`DescrptSeAEf`**(*args, **kwargs)

Bases: *`Descriptor`*

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 maxmum number of type i atoms in the cut-off radius

neuron
: [`list`[`int`]] Number of neurons in each hidden layers of the embedding net

axis_neuron
: Number of the axis neuron (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero
: Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are "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

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 statisitcs (avg and std) of the training data.
`enable_compression`(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
`enable_mixed_precision`([mixed_prec])	Reveive 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_feed_dict`(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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.
`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.

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. 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) → None

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

Parameters

data_coord
: The coordinates. Can be generated by deepmd.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

`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[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.**DescrptSeAEfLower**(*args, **kwargs)

Bases: *DescrptSeA*

Helper class for implementing DescrptSeAEf.

Attributes

precision
Precision of filter network.

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 statisitcs (avg and std) of the training data.
`enable_compression`(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
`enable_mixed_precision`([mixed_prec])	Reveive 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_feed_dict`(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
`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.

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)

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

Parameters

data_coord
: The coordinates. Can be generated by deepmd.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

deepmd.descriptor.se_a_mask module

`class deepmd.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 $\mathcal{D}^i \in \mathcal{R}^{M_1 \times M_2}$ is given by [1]_

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}^i_<$$

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

$$(\mathcal{R}^i)_j = [\begin{array}{c} s(r_{ji}) \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{array}]$$

where $\mathrm{R}_{ji} = \mathrm{R}_j - \mathrm{R}_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \lVert \mathrm{R}_{ji} \rVert$ is its norm. The switching function $s(r)$ is defined as:

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

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

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}^i_< \in \mathbb{R}^{N \times M_2}$ takes first M_2 columns of $\mathcal{G}^i$. The equation of embedding network $\mathcal{N}$ can be found at *deepmd.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 maxmum number of type i atoms in the neighbor list.

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron
: Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

activation_function
: The activation function in the embedding net. Supported options are {0}

precision
: The precision of the embedding net parameters. Supported options are {1}

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

References

———-

.. [1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018.
End-to-end symmetry preserving inter-atomic potential energy model for finite and

extended systems. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 4441–4451.

Attributes

precision
Precision of filter network.

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 statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec])	Reveive 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_feed_dict(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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 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.

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[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) → None

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

Parameters

data_coord
The coordinates. Can be generated by deepmd.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

`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

deepmd.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 maxmum number of type i atoms in the cut-off radius

neuron
[list[int]] Number of neurons in each hidden layers of the embedding net $\mathcal{N}$

axis_neuron
Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

resnet_dt
Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
If the weights of embedding net are trainable.

seed
Random seed for initializing the network parameters.

type_one_side
Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
[List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

set_davg_zero
Set the shift of embedding net input to zero.

activation_function
: The activation function in the embedding net. Supported options are “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

`precision`
: Precision of filter network.

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 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, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
`enable_mixed_precision`([mixed_prec])	Reveive 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_feed_dict`(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
`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.

`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], ndescrpt: 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.

ndescrpt
[int] The number of descriptors for each atom.

atype
[tf.Tensor] The type of atoms, with the size of shape0.

shape0
[tf.Tensor] The shape of the first dimension of the inputs, which is equal to nsamples * natoms.

nei_type_vec
[tf.Tensor] The type of neighbors, with the size of (shape0, nnei).

Returns

tf.Tensor
The type exclude mask, with the shape of (shape0, ndescrpt), and the precision of GLOBAL_TF_FLOAT_PRECISION. The mask has the value of 1 if the interaction between two types is not excluded, and 0 otherwise.

See also:

deepmd.descriptor.descriptor.Descriptor.build_type_exclude_mask

Notes

This method has the similiar way to build the type exclude mask as *deepmd.descriptor.descriptor.Descriptor.build_type_exclude_mask()*. The mathmatical expression has been explained in that method. The difference is that the attention descriptor has provided the type of the neighbors (idx_j) that is not in order, so we use it from an extra input.

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) → None

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

Parameters

data_coord
The coordinates. Can be generated by deepmd.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_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.

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

deepmd.descriptor.se_r module

`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 maxmum number of type i atoms in the cut-off radius

neuron
: [list[int]] Number of neurons in each hidden layers of the embedding net

resnet_dt
: Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
: If the weights of embedding net are trainable.

seed
: Random seed for initializing the network parameters.

type_one_side
: Try to build N_types embedding nets. Otherwise, building N_types^2 embedding nets

exclude_types
: [List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

activation_function
: The activation function in the embedding net. Supported options are "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

precision
: Precision of filter network.

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 statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
enable_mixed_precision([mixed_prec])	Reveive 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_feed_dict(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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.
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.

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)

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

Parameters

data_coord
The coordinates. Can be generated by deepmd.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

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 statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
The nearest distance between atoms

graph
[`tf.Graph`] The graph of the model

graph_def
[`tf.GraphDef`] The graph_def of the model

table_extrapolate
The scale of model extrapolation

table_stride_1
The uniform stride of the first table

table_stride_2
The uniform stride of the second table

check_frequency
The overflow check frequency

suffix
[str, optional] The suffix of the scope

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 statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict

The dict of statisitcs computed from compute_input_stats, including:

sumr
The sum of radial statisitcs.

sumn
The sum of neighbor numbers.

sumr2
The sum of square of radial statisitcs.

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

deepmd.descriptor.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 maxmum number of type i atoms in the cut-off radius

neuron
[list[int]] Number of neurons in each hidden layers of the embedding net

resnet_dt
Time-step dt in the resnet construction: y = x + dt * phi (Wx + b)

trainable
If the weights of embedding net are trainable.

seed
Random seed for initializing the network parameters.

set_davg_zero
Set the shift of embedding net input to zero.

activation_function
The activation function in the embedding net. Supported options are "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

precision
Precision of filter network.

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 statisitcs (avg and std) of the training data.
enable_compression(min_nbor_dist, graph, ...)	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
`enable_mixed_precision`([mixed_prec])	Reveive 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_feed_dict`(coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor.
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.
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.

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: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list, input_dict: dict) → None

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

Parameters

data_coord
: The coordinates. Can be generated by deepmd.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

`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 statisitcs (distance, max_nbor_size and env_mat_range) of the training data.

Parameters

min_nbor_dist
: The nearest distance between atoms

graph
: [`tf.Graph`] The graph of the model

graph_def
: [`tf.GraphDef`] The graph_def of the model

table_extrapolate
: The scale of model extrapolation

table_stride_1
: The uniform stride of the first table

table_stride_2
: The uniform stride of the second table

check_frequency
The overflow check frequency

suffix
[str, `optional`] The suffix of the scope

`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 statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.

Parameters

stat_dict

The dict of statisitcs computed from compute_input_stats, including:

sumr
The sum of radial statisitcs.

suma
The sum of relative coord statisitcs.

sumn
The sum of neighbor numbers.

sumr2
The sum of square of radial statisitcs.

suma2
The sum of square of relative coord statisitcs.

`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

deepmd.entrypoints package

Submodule that contains all the DeePMD-Kit entry point scripts.

`deepmd.entrypoints.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 speccified 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 trining 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

`deepmd.entrypoints.make_model_devi`(*, models: list, system: str, set_prefix: str, output: str, frequency: int, **kwargs)
Make model deviation calculation.

Parameters

models
[`list`] A list of paths of models to use for making model deviation

system
[`str`] The path of system to make model deviation calculation

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 paramter is used to determine the index in the output file.

**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

```
>>> neighbor_stat(system='.', rcut=6., type_map=["C", "H", "O", "N", "P", "S", "Mg", "Na", "HW
↪", "OW", "mNa", "mCl", "mC", "mH", "mMg", "mN", "mO", "mP"])
min_nbor_dist: 0.6599510670195264
max_nbor_size: [23, 26, 19, 16, 2, 2, 1, 1, 72, 37, 5, 0, 31, 29, 1, 21, 20, 5]
```

`deepmd.entrypoints.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`] munber of tests to do

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_dp(*, INPUT: str, init_model: Optional[str], restart: Optional[str], output: str, init_frz_model: str, mpi_log: str, log_level: int, log_path: Optional[str], is_compress: bool = False, skip_neighbor_stat: bool = False, finetune: Optional[str] = None, **kwargs)

Run DeePMD model training.

Parameters

INPUT
    [str] json/yaml control file

init_model
    [Optional[str]] path to checkpoint folder or None

restart
    [Optional[str]] path to checkpoint folder or None

output
    [str] path for dump file with arguments

init_frz_model
    [str] path to frozen model or None

mpi_log
    [str] mpi logging mode

log_level
    [int] logging level defined by int 0-3

log_path
    [Optional[str]] logging file path or None if logs are to be output only to stdout

is_compress
    [bool] indicates whether in the model compress mode

skip_neighbor_stat
    [bool, default=False] skip checking neighbor statistics

finetune
    [Optional[str]] path to pretrained model or None

**kwargs
    additional arguments

Raises

RuntimeError
    if distributed training job name is wrong

deepmd.entrypoints.transfer(*, old_model: str, raw_model: str, output: str, **kwargs)

Transfer operation from old fron graph to new prepared raw graph.

Parameters

old_model
    [str] frozen old graph model

raw_model
[str] new model that will accept ops from old model

output
[str] new model with transfered parameters will be saved to this location

**kwargs
additional arguments

Submodules

deepmd.entrypoints.compress module

Compress a model, which including tabulating the embedding-net.

`deepmd.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 speccified 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

Module that prints train input arguments docstrings.

`deepmd.entrypoints.doc.doc_train_input`(*, out_type: str = 'rst', **kwargs)

Print out trining 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, 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

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[List[str]] = None)`

DeePMD-Kit entry point.

Parameters

args
: [List[str], optional] list of command line arguments, used to avoid calling from the subprocess, as it is quite slow to import tensorflow

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`**`(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

deepmd.entrypoints.neighbor_stat module

`deepmd.entrypoints.neighbor_stat.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", "Mg", "Na", "HW
↪", "OW", "mNa", "mCl", "mC", "mH", "mMg", "mN", "mO", "mP"])
min_nbor_dist: 0.6599510670195264
max_nbor_size: [23, 26, 19, 16, 2, 2, 1, 1, 72, 37, 5, 0, 31, 29, 1, 21, 20, 5]
```

deepmd.entrypoints.test module

Test trained DeePMD model.

`deepmd.entrypoints.test.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`] munber of tests to do

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.

`deepmd.entrypoints.train.train`(*, INPUT: str, init_model: Optional[str], restart: Optional[str], output: str, init_frz_model: str, mpi_log: str, log_level: int, log_path: Optional[str], is_compress: bool = False, skip_neighbor_stat: bool = False, finetune: Optional[str] = None, **kwargs)

Run DeePMD model training.

Parameters

INPUT
: [str] json/yaml control file

init_model
: [Optional[str]] path to checkpoint folder or None

restart
: [Optional[str]] path to checkpoint folder or None

output
: [str] path for dump file with arguments

init_frz_model
: [str] path to frozen model or None

mpi_log
: [str] mpi logging mode

log_level
: [int] logging level defined by int 0-3

log_path
: [Optional[str]] logging file path or None if logs are to be output only to stdout

is_compress
: [bool] indicates whether in the model compress mode

skip_neighbor_stat
: [bool, default=False] skip checking neighbor statistics

finetune
: [Optional[str]] path to pretrained model or None

**kwargs
: additional arguments

Raises

RuntimeError
: if distributed training job name is wrong

deepmd.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 fron graph to new prepared raw graph.

Parameters

old_model
: [str] frozen old graph model

raw_model
: [str] new model that will accept ops from old model

output
: [str] new model with transfered parameters will be saved to this location

**kwargs
: additional arguments

deepmd.fit package

`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
: The descrptor $\mathcal{D}$

neuron
: Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

! numb_dos (added)
: Number of gridpoints on which the DOS is evaluated (NEDOS in VASP)

rcond
: The condition number for the regression of atomic energy.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

seed
: Random seed for initializing the network parameters.

activation_function
: The activation function ϕ 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

`build`(inputs, natoms[, input_dict, reuse, ...])	Build the computational graph for fitting net.
`compute_input_stats`(all_stat[, protection])	Compute the input statistics.
`compute_output_stats`(all_stat[, mixed_type])	Compute the ouput statistics.
`enable_mixed_precision`([mixed_prec])	Reveive the mixed precision setting.
`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.
`init_variables`(graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
`register`(key)	Register a Fitting plugin.

`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

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 ouput statistics.

Parameters

all_stat
: must have the following components: all_stat['dos'] of shape n_sys x n_batch x n_frame x numb_dos can be prepared by model.make_stat_input

mixed_type
: Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

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_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 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 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", "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
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, rot_mat, natoms[, ...])	Build the computational graph for fitting net.
enable_mixed_precision([mixed_prec])	Reveive the mixed precision setting.
get_out_size()	Get the output size.
get_sel_type()	Get selected type.
init_variables(graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
`register`(key)	Register a Fitting plugin.

build(input_d: Tensor, rot_mat: Tensor, natoms: Tensor, input_dict: Optional[dict] = None, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for fitting net.

Parameters

input_d
The input descriptor

rot_mat
The rotation matrix from the descriptor.

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

input_dict
Additional dict for inputs.

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

dipole
The atomic dipole.

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_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 $\mathcal{D}$:

$$E(\mathcal{D}) = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \cdots \circ \mathcal{L}^{(1)} \circ \mathcal{L}^{(0)}$$

The first n hidden layers $\mathcal{L}^{(0)}, \cdots, \mathcal{L}^{(n-1)}$ are given by

$$\mathrm{y} = \mathcal{L}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \boldsymbol{\phi}(\mathrm{x}^T \mathrm{w} + \mathrm{b})$$

where $\mathrm{x} \in \mathbb{R}^{N_1}$ is the input vector and $\mathrm{y} \in \mathbb{R}^{N_2}$ is the output vector. $\mathrm{w} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathrm{b} \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable[i] is True. $\boldsymbol{\phi}$ is the activation function.

The output layer $\mathcal{L}^{(n)}$ is given by

$$\mathrm{y} = \mathcal{L}^{(n)}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \mathrm{x}^T \mathrm{w} + \mathrm{b}$$

where $\mathrm{x} \in \mathbb{R}^{N_{n-1}}$ is the input vector and $\mathrm{y} \in \mathbb{R}$ is the output scalar. $\mathrm{w} \in \mathbb{R}^{N_{n-1}}$ and $\mathrm{b} \in \mathbb{R}$ are weights and bias, respectively, both of which are trainable if trainable[n] is True.

Parameters

descrpt
: The descrptor $\mathcal{D}$

neuron
: Number of neurons N in each hidden layer of the fitting net

resnet_dt
: Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
: Number of frame parameter

numb_aparam
: Number of atomic parameter

rcond
: The condition number for the regression of atomic energy.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

seed
: Random seed for initializing the network parameters.

atom_ener
: Specifying atomic energy contribution in vacuum. The set_davg_zero key in the descrptor should be set.

activation_function
: The activation function ϕ in the embedding net. Supported options are "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

`build`(inputs, natoms[, input_dict, reuse, ...])	Build the computational graph for fitting net.
`change_energy_bias`(data, frozen_model, ...)	Change the energy bias according to the input data and the pretrained model.
`compute_input_stats`(all_stat[, protection])	Compute the input statistics.
`compute_output_stats`(all_stat[, mixed_type])	Compute the ouput statistics.
`enable_mixed_precision`([mixed_prec])	Reveive the mixed precision setting.
`get_numb_aparam`()	Get the number of atomic parameters.
`get_numb_fparam`()	Get the number of frame parameters.
`init_variables`(graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
`register`(key)	Register a Fitting plugin.

`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', 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 sqaure 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 ouput statistics.

Parameters

all_stat
must have the following components: all_stat['energy'] of shape n_sys x n_batch x n_frame can be prepared by model.make_stat_input

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

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_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

init_variables(graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
register(key)	Register a Fitting plugin.

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

```
>>> @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", "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".

Methods

build(input_d, rot_mat, natoms[, ...])	Build the computational graph for fitting net.
enable_mixed_precision([mixed_prec])	Reveive the mixed precision setting.
get_out_size()	Get the output size.
get_sel_type()	Get selected atom types.
init_variables(graph, graph_def[, suffix])	Init the fitting net variables with the given dict.

build(input_d, rot_mat, natoms, input_dict: Optional[dict] = None, reuse=None, suffix='') → Tensor

Build the computational graph for fitting net.

Parameters

input_d
: The input descriptor

rot_mat
: The rotation matrix from the descriptor.

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

input_dict
: Additional dict for inputs.

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

polar
The system polarizability

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_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 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. 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", "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
: 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, rot_mat, natoms[, ...])	Build the computational graph for fitting net.
compute_input_stats(all_stat[, protection])	Compute the input statistics.
enable_mixed_precision([mixed_prec])	Reveive the mixed precision setting.
get_out_size()	Get the output size.
get_sel_type()	Get selected atom types.
init_variables(graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
register(key)	Register a Fitting plugin.

build(input_d: Tensor, rot_mat: Tensor, natoms: Tensor, input_dict: Optional[dict] = None, reuse: Optional[bool] = None, suffix: str = '')

Build the computational graph for fitting net.

Parameters

input_d
: The input descriptor

rot_mat
: The rotation matrix from the descriptor.

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

input_dict
: Additional dict for inputs.

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

atomic_polar
The atomic polarizability

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_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

deepmd.fit.dipole module

class deepmd.fit.dipole.**DipoleFittingSeA**(*args, **kwargs)
Bases: *`Fitting`*

Fit the atomic dipole 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 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", "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
: 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, rot_mat, natoms[, ...])	Build the computational graph for fitting net.
enable_mixed_precision([mixed_prec])	Reveive the mixed precision setting.
get_out_size()	Get the output size.
get_sel_type()	Get selected type.
init_variables(graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
register(key)	Register a Fitting plugin.

build(input_d: Tensor, rot_mat: Tensor, natoms: Tensor, input_dict: Optional[dict] = None, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for fitting net.

Parameters

input_d
: The input descriptor

rot_mat
: The rotation matrix from the descriptor.

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

input_dict
: Additional dict for inputs.

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

`dipole`
The atomic dipole.

`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_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 descrptor $\mathcal{D}$

neuron
Number of neurons N in each hidden layer of the fitting net

resnet_dt
Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
Number of frame parameter

numb_aparam
Number of atomic parameter

! numb_dos (added)
Number of gridpoints on which the DOS is evaluated (NEDOS in VASP)

rcond
The condition number for the regression of atomic energy.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

seed
: Random seed for initializing the network parameters.

activation_function
: The activation function ϕ 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

build(inputs, natoms[, input_dict, reuse, ...])	Build the computational graph for fitting net.
compute_input_stats(all_stat[, protection])	Compute the input statistics.
compute_output_stats(all_stat[, mixed_type])	Compute the ouput statistics.
enable_mixed_precision([mixed_prec])	Reveive the mixed precision setting.
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.
init_variables(graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
`register`(key)	Register a Fitting plugin.

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

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 ouput statistics.

Parameters

all_stat
must have the following components: all_stat['dos'] of shape n_sys x n_batch x n_frame x numb_dos can be prepared by model.make_stat_input

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

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_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 $\mathcal{D}$:

$$E(\mathcal{D}) = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \cdots \circ \mathcal{L}^{(1)} \circ \mathcal{L}^{(0)}$$

The first n hidden layers $\mathcal{L}^{(0)}, \cdots, \mathcal{L}^{(n-1)}$ are given by

$$\mathrm{y} = \mathcal{L}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \boldsymbol{\phi}(\mathrm{x}^T\mathrm{w} + \mathrm{b})$$

where $\mathrm{x} \in \mathbb{R}^{N_1}$ is the input vector and $\mathrm{y} \in \mathbb{R}^{N_2}$ is the output vector. $\mathrm{w} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathrm{b} \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable[i] is True. $\boldsymbol{\phi}$ is the activation function.

The output layer $\mathcal{L}^{(n)}$ is given by

$$\mathrm{y} = \mathcal{L}^{(n)}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \mathrm{x}^T\mathrm{w} + \mathrm{b}$$

where $\mathrm{x} \in \mathbb{R}^{N_{n-1}}$ is the input vector and $\mathrm{y} \in \mathbb{R}$ is the output scalar. $\mathrm{w} \in \mathbb{R}^{N_{n-1}}$ and $\mathrm{b} \in \mathbb{R}$ are weights and bias, respectively, both of which are trainable if trainable[n] is True.

Parameters

descrpt
The descrptor $\mathcal{D}$

neuron
Number of neurons N in each hidden layer of the fitting net

resnet_dt
Time-step dt in the resnet construction: $y = x + dt * \phi(Wx + b)$

numb_fparam
Number of frame parameter

numb_aparam
Number of atomic parameter

rcond
The condition number for the regression of atomic energy.

tot_ener_zero
: Force the total energy to zero. Useful for the charge fitting.

trainable
: If the weights of fitting net are trainable. Suppose that we have N_l hidden layers in the fitting net, this list is of length $N_l + 1$, specifying if the hidden layers and the output layer are trainable.

seed
: Random seed for initializing the network parameters.

atom_ener
: Specifying atomic energy contribution in vacuum. The set_davg_zero key in the descrptor should be set.

activation_function
: The activation function ϕ in the embedding net. Supported options are "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

build(inputs, natoms[, input_dict, reuse, ...])	Build the computational graph for fitting net.
change_energy_bias(data, frozen_model, ...)	Change the energy bias according to the input data and the pretrained model.
compute_input_stats(all_stat[, protection])	Compute the input statistics.
compute_output_stats(all_stat[, mixed_type])	Compute the ouput statistics.
enable_mixed_precision([mixed_prec])	Reveive the mixed precision setting.
get_numb_aparam()	Get the number of atomic parameters.
get_numb_fparam()	Get the number of frame parameters.
init_variables(graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
`register`(key)	Register a Fitting plugin.

`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', 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 sqaure 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 ouput statistics.

Parameters

all_stat
must have the following components: all_stat['energy'] of shape n_sys x n_batch x n_frame can be prepared by model.make_stat_input

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

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_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

deepmd.fit.fitting module

class deepmd.fit.fitting.**Fitting**(*args, **kwargs)

Bases: *PluginVariant*

Attributes

precision
Precision of fitting network.

Methods

init_variables(graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
register(key)	Register a Fitting plugin.

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

```
>>> @Fitting.register("some_fitting")
    class SomeFitting(Fitting):
        pass
```

deepmd.fit.polar 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 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”, “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”.

Methods

build(input_d, rot_mat, natoms[, ...])	Build the computational graph for fitting net.
enable_mixed_precision([mixed_prec])	Reveive the mixed precision setting.
get_out_size()	Get the output size.
get_sel_type()	Get selected atom types.
init_variables(graph, graph_def[, suffix])	Init the fitting net variables with the given dict.

build(input_d, rot_mat, natoms, input_dict: Optional[dict] = None, reuse=None, suffix='') → Tensor

Build the computational graph for fitting net.

Parameters

input_d
: The input descriptor

rot_mat
The rotation matrix from the descriptor.

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

input_dict
Additional dict for inputs.

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

polar
The system polarizability

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_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 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. 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", "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
: 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, rot_mat, natoms[, ...])	Build the computational graph for fitting net.
compute_input_stats(all_stat[, protection])	Compute the input statistics.
enable_mixed_precision([mixed_prec])	Reveive the mixed precision setting.
get_out_size()	Get the output size.
get_sel_type()	Get selected atom types.
init_variables(graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
register(key)	Register a Fitting plugin.

build(input_d: Tensor, rot_mat: Tensor, natoms: Tensor, input_dict: Optional[dict] = None, reuse: Optional[bool] = None, suffix: str = '')

Build the computational graph for fitting net.

Parameters

input_d
: The input descriptor

rot_mat
The rotation matrix from the descriptor.

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

input_dict
Additional dict for inputs.

reuse
The weights in the networks should be reused when get the variable.

suffix
Name suffix to identify this descriptor

Returns

atomic_polar
The atomic polarizability

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_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)

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.

> 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 chanage the order!

Attributes

model_type
: Get type of model.

model_version
: Get version of model.

sess
: Get TF session.

Methods

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_atten.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(coords: ndarray, cells: ndarray, atom_types: List[int], fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, efield: Optional[ndarray] = None, mixed_type: bool = False) → array

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_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

class deepmd.infer.**DeepDipole**(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False)

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

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 chanage the order!

Attributes

model_type
Get type of model.

model_version
Get version of model.

sess
Get TF session.

Methods

`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.

load_prefix: str

class deepmd.infer.**DeepEval**(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False, auto_batch_size: Union[bool, int, AutoBatchSize] = False)

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.

Attributes

`model_type`
Get type of model.

`model_version`
Get version of model.

`sess`
Get TF session.

Methods

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.

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:

```
>>> 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_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

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

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.

class `deepmd.infer.DeepGlobalPolar`(model_file: str, load_prefix: str = 'load', default_tf_graph: bool = False)

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

Attributes

`model_type`
Get type of model.

`model_version`
Get version of model.

`sess`
Get TF session.

Methods

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.

`eval`(coords: ndarray, cells: ndarray, atom_types: List[int], atomic: bool = False, fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, efield: Optional[ndarray] = None) → ndarray

Evaluate the model.

Parameters

coords
: The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
: The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
: The atom types The list should contain natoms ints

atomic
: 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.

`load_prefix``: str`

`class deepmd.infer.`**`DeepPolar`**(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False)

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

> 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 chanage the order!

Attributes

model_type
Get type of model.

model_version
Get version of model.

sess
Get TF session.

Methods

`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.

load_prefix: str

class deepmd.infer.DeepPot(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False, auto_batch_size: Union[bool, int, AutoBatchSize] = True)

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.

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 chanage the order!

Examples

```
>>> from deepmd.infer import DeepPot
>>> import numpy as np
>>> dp = DeepPot('graph.pb')
>>> coord = np.array([[1,0,0], [0,0,1.5], [1,0,3]]).reshape([1, -1])
>>> cell = np.diag(10 * np.ones(3)).reshape([1, -1])
>>> atype = [1,0,1]
>>> e, f, v = dp.eval(coord, cell, atype)
```

where e, f and v are predicted energy, force and virial of the system, respectively.

Attributes

`model_type`
: Get type of model.

`model_version`
: Get version of model.

`sess`
: Get TF session.

Methods

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_ntypes()	Get the number of atom types of this model.
get_ntypes_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.

`eval`(coords: ndarray, cells: ndarray, atom_types: List[int], atomic: bool = False, fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, efield: Optional[ndarray] = None, mixed_type: bool = False) → Tuple[ndarray, ...]

Evaluate the energy, force and virial by using this DP.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

atomic
Calculate the atomic energy and virial

fparam
The frame parameter. The array can be of size : - nframes x dim_fparam. - dim_fparam. Then all frames are assumed to be provided with the same fparam.

aparam
The atomic parameter The array can be of size : - nframes x natoms x dim_aparam. - natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. - dim_aparam. Then all frames and atoms are provided with the same aparam.

efield
The external field on atoms. The array should be of size nframes x natoms x 3

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

eval_descriptor(coords: ndarray, cells: ndarray, atom_types: List[int], fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, efield: Optional[ndarray] = None, mixed_type: bool = False) → array

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

deepmd.infer.**DeepPotential**(model_file: Union[str, Path], load_prefix: str = 'load', default_tf_graph: bool = False) → Union[DeepDipole, DeepGlobalPolar, DeepPolar, DeepPot, DeepDOS, DeepWFC]

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

Returns

Union[*DeepDipole*, *DeepGlobalPolar*, *DeepPolar*, *DeepPot*, *DeepWFC*]
one of the available potentials

Raises

RuntimeError
if model file does not correspond to any implementd potential

class deepmd.infer.**DeepWFC**(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False)

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

> 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 chanage the order!

Attributes

`model_type`
: Get type of model.

`model_version`
: Get version of model.

`sess`
: Get TF session.

Methods

`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.

`load_prefix: str`

`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 wfcc

sys_charge_map
: Gives the amount of charge for the real atoms

ewald_h
: Grid spacing of the reciprocal part of Ewald sum. Unit: A

ewald_beta
: Splitting parameter of the Ewald sum. Unit: A^{-1}

Attributes

model_type
: Get type of model.

model_version
: Get version of model.

sess
: Get TF session.

Methods

build_fv_graph()	Build the computational graph for the force and virial inference.
eval(coord, box, atype[, eval_fv])	Evaluate the modification.
`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.
modify_data(data, data_sys)	Modify data.
`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_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

load_prefix: str

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, mixed_type=False)

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

Returns

model_devi
: [`numpy.ndarray`, n_frames x 8] Model deviation results. The first column is index of steps, the other 7 columns are max_devi_v, min_devi_v, avg_devi_v, max_devi_f, min_devi_f, avg_devi_f, devi_e.

Examples

```
>>> from deepmd.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`**(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: A

ewald_beta
Splitting parameter of the Ewald sum. Unit: A^{-1}

Attributes

`model_type`
Get type of model.

`model_version`
Get version of model.

`sess`
Get TF session.

Methods

build_fv_graph()	Build the computational graph for the force and virial inference.
eval(coord, box, atype[, eval_fv])	Evaluate the modification.
`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.
modify_data(data, data_sys)	Modify data.
`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_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

load_prefix: str

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)

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

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 chanage the order!

Attributes

model_type
: Get type of model.

model_version
: Get version of model.

sess
: Get TF session.

Methods

`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.

`load_prefix:` `str`

deepmd.infer.deep_dos module

`class deepmd.infer.deep_dos.DeepDOS`(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False, auto_batch_size: Union[bool, int, AutoBatchSize] = True)

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.

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 chanage the order!

Attributes

model_type
Get type of model.

model_version
Get version of model.

sess
Get TF session.

Methods

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_atten.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(coords: ndarray, cells: ndarray, atom_types: List[int], fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, efield: Optional[ndarray] = None, mixed_type: bool = False) → array

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_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

class deepmd.infer.deep_eval.**DeepEval**(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False, auto_batch_size: Union[bool, int, AutoBatchSize] = False)

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.

Attributes

model_type
Get type of model.

model_version
Get version of model.

sess
Get TF session.

Methods

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.

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:

```
>>> 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_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

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

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.

deepmd.infer.deep_polar module

`class deepmd.infer.deep_polar.DeepGlobalPolar(model_file: str, load_prefix: str = 'load', default_tf_graph: bool = False)`

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

Attributes

model_type
Get type of model.

model_version
Get version of model.

sess
Get TF session.

Methods

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.

eval(coords: ndarray, cells: ndarray, atom_types: List[int], atomic: bool = False, fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, efield: Optional[ndarray] = None) → ndarray

Evaluate the model.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

atomic
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.

load_prefix: str

class deepmd.infer.deep_polar.DeepPolar(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False)

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

> 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 chanage the order!

Attributes

`model_type`
Get type of model.

`model_version`
Get version of model.

`sess`
Get TF session.

Methods

`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.

`load_prefix: str`

deepmd.infer.deep_pot module

`class deepmd.infer.deep_pot.`**`DeePot`**`(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False, auto_batch_size: Union[bool, int, AutoBatchSize] = True)`

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.

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 chanage the order!

Examples

```
>>> from deepmd.infer import DeepPot
>>> import numpy as np
>>> dp = DeepPot('graph.pb')
>>> coord = np.array([[1,0,0], [0,0,1.5], [1,0,3]]).reshape([1, -1])
>>> cell = np.diag(10 * np.ones(3)).reshape([1, -1])
>>> atype = [1,0,1]
>>> e, f, v = dp.eval(coord, cell, atype)
```

where e, f and v are predicted energy, force and virial of the system, respectively.

Attributes

`model_type`
: Get type of model.

`model_version`
: Get version of model.

`sess`
: Get TF session.

Methods

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_ntypes()	Get the number of atom types of this model.
get_ntypes_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.

`eval`(coords: ndarray, cells: ndarray, atom_types: List[int], atomic: bool = False, fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, efield: Optional[ndarray] = None, mixed_type: bool = False) → Tuple[ndarray, ...]

Evaluate the energy, force and virial by using this DP.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

atomic
Calculate the atomic energy and virial

fparam
The frame parameter. The array can be of size : - nframes x dim_fparam. - dim_fparam. Then all frames are assumed to be provided with the same fparam.

aparam
The atomic parameter The array can be of size : - nframes x natoms x dim_aparam. - natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. - dim_aparam. Then all frames and atoms are provided with the same aparam.

efield
The external field on atoms. The array should be of size nframes x natoms x 3

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

eval_descriptor(coords: ndarray, cells: ndarray, atom_types: List[int], fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, efield: Optional[ndarray] = None, mixed_type: bool = False) → array

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

deepmd.infer.deep_tensor module

class deepmd.infer.deep_tensor.**DeepTensor**(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False)

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

Attributes

model_type
: Get type of model.

model_version
: Get version of model.

sess
: Get TF session.

Methods

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()	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_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.

`eval`(coords: ndarray, cells: ndarray, atom_types: List[int], atomic: bool = True, fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, efield: Optional[ndarray] = None, mixed_type: bool = False) → ndarray

Evaluate the model.

Parameters

coords
The coordinates of atoms. The array should be of size nframes x natoms x 3

cells
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom_types
The atom types The list should contain natoms ints

atomic
If True (default), return the atomic tensor Otherwise return the global tensor

fparam
Not used in this model

aparam
Not used in this model

efield
Not used in this model

mixed_type
Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

Returns

tensor
The returned tensor If atomic == False then of size nframes x output_dim else of size nframes x natoms x output_dim

eval_full(coords: 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 system may have different natoms_vec(s), with the same nloc.

Returns

tensor
The global tensor. shape: [nframes x nout]

force
The component-wise force (negative derivative) on each atom. shape: [nframes x nout x natoms x 3]

virial
The component-wise virial of the tensor. shape: [nframes x nout x 9]

atom_tensor
The atomic tensor. Only returned when atomic == True shape: [nframes x natoms x nout]

atom_virial
The atomic virial. Only returned when atomic == True shape: [nframes x nout x natoms x 9]

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.

load_prefix: str

tensors = {'t_box': 't_box:0', 't_coord': 't_coord:0', 't_mesh': 't_mesh:0', 't_natoms': 't_natoms:0', 't_ntypes': 'descrpt_attr/ntypes:0', 't_ouput_dim': 'model_attr/output_dim:0', 't_rcut': 'descrpt_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)

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

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 chanage the order!

Attributes

model_type
Get type of model.

model_version
Get version of model.

sess
Get TF session.

Methods

`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.

`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.

`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.model_devi module

`deepmd.infer.model_devi.calc_model_devi`(coord, box, atype, models, fname=None, frequency=1, mixed_type=False)

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

Returns

model_devi
: [`numpy.ndarray`, n_frames x 8] Model deviation results. The first column is index of steps, the other 7 columns are max_devi_v, min_devi_v, avg_devi_v, max_devi_f, min_devi_f, avg_devi_f, devi_e.

Examples

```
>>> from deepmd.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)
```

deepmd.infer.model_devi.calc_model_devi_e(es: ndarray) → 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

Returns

max_devi_e
[numpy.ndarray] maximum deviation of energy

deepmd.infer.model_devi.calc_model_devi_f(fs: ndarray) → Tuple[ndarray]

Calculate model deviation of force.

Parameters

fs
[numpy.ndarray] size of n_models x n_frames x n_atoms x 3

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

deepmd.infer.model_devi.calc_model_devi_v(vs: ndarray) → Tuple[ndarray]

Calculate model deviation of virial.

Parameters

vs
[numpy.ndarray] size of n_models x n_frames x 9

Returns

max_devi_v
[numpy.ndarray] maximum deviation of virial in 9 elements

min_devi_v
[numpy.ndarray] minimum deviation of virial in 9 elements

avg_devi_v
[numpy.ndarray] average deviation of virial in 9 elements

deepmd.infer.model_devi.make_model_devi(*, models: list, system: str, set_prefix: str, output: str, frequency: int, **kwargs)

Make model deviation calculation.

Parameters

models
[list] A list of paths of models to use for making model deviation

system
[str] The path of system to make model deviation calculation

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 paramter is used to determine the index in the output file.

**kwargs
Arbitrary keyword arguments.

`deepmd.infer.model_devi.write_model_devi_out`(devi: ndarray, fname: str, header: str = '')

Write output of model deviation.

Parameters

devi
[`numpy.ndarray`] the first column is the steps index

fname
[str] the file name to dump

header
[str, default=""] the header to dump

deepmd.loggers package

Module taking care of logging duties.

`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:

	our notation	python logging	tensorflow cpp	OpenMP
debug	10	10	0	1/on/true/yes
info	20	20	1	0/off/false/no
warning	30	30	2	0/off/false/no
error	40	40	3	0/off/false/no

References

https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U https://stackoverflow.com/questions/35869137/avoid-tensorflow-print-on-standard-error https://stackoverflow.com/questions/56085015/suppress-openmp-debug-messages-when-running-tensorflow-on-cpu

Submodules

deepmd.loggers.loggers module

Logger initialization for package.

`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:

	our notation	python logging	tensorflow cpp	OpenMP
debug	10	10	0	1/on/true/yes
info	20	20	1	0/off/false/no
warning	30	30	2	0/off/false/no
error	40	40	3	0/off/false/no

References

https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U https://stackoverflow.com/questions/35869137/avoid-tensorflow-print-on-standard-error https://stackoverflow.com/questions/56085015/suppress-openmp-debug-messages-when-running-tensorflow-on-cpu

deepmd.loss package

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, protect_value: float = 1e-08, log_fit: bool = False)

Bases: `Loss`

Loss function for DeepDOS models.

Methods

`build`(learning_rate, natoms, model_dict, ...)	Build the loss function graph.
`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.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.
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, 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

build(learning_rate, natoms, model_dict, ...)	Build the loss function graph.
eval(sess, feed_dict, natoms)	Eval the loss function.

print_header	
print_on_training	

build(learning_rate, natoms, model_dict, label_dict, suffix)

Build the loss function graph.

Parameters

learning_rate
[`tf.Tensor`] learning rate

natoms
[`tf.Tensor`] number of atoms

model_dict
[`dict[str, tf.Tensor]`] A dictionary that maps model keys to tensors

label_dict
[`dict[str, tf.Tensor]`] A dictionary that maps label keys to tensors

suffix
[`str`] suffix

Returns

`tf.Tensor`
the total squared loss

`dict[str, tf.Tensor]`
A dictionary that maps loss keys to more loss tensors

`eval`(sess, feed_dict, natoms)
Eval the loss function.

Parameters

sess
[`tf.Session`] TensorFlow session

feed_dict
[`dict[tf.placeholder, tf.Tensor]`] A dictionary that maps graph elements to values

natoms
[`tf.Tensor`] number of atoms

Returns

`dict`
A dictionary that maps keys to values. It should contain key natoms

`print_header`()

`print_on_training`(tb_writer, cur_batch, sess, natoms, feed_dict_test, feed_dict_batch)

`class deepmd.loss.EnerStdLoss`(starter_learning_rate: float, start_pref_e: float = 0.02, limit_pref_e: float = 1.0, start_pref_f: float = 1000, limit_pref_f: float = 1.0, start_pref_v: float = 0.0, limit_pref_v: float = 0.0, start_pref_ae: float = 0.0, limit_pref_ae: float = 0.0, start_pref_pf: float = 0.0, limit_pref_pf: float = 0.0, relative_f: Optional[float] = None, enable_atom_ener_coeff: bool = False)

Bases: *`Loss`*

Standard loss function for DP models.

Parameters

enable_atom_ener_coeff
[bool] if true, the energy will be computed as sum_i c_i E_i

Methods

`build`(learning_rate, natoms, model_dict, ...)	Build the loss function graph.
`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.TensorLoss`(jdata, **kwarg)
Bases: *`Loss`*

Loss function for tensorial properties.

Methods

`build`(learning_rate, natoms, model_dict, ...)	Build the loss function graph.
`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

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)

Bases: *Loss*

Loss function for DeepDOS models.

Methods

build(learning_rate, natoms, model_dict, ...)	Build the loss function graph.
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

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.
`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.ener.EnerSpinLoss(starter_learning_rate: float, start_pref_e: float = 0.02, limit_pref_e: float = 1.0, start_pref_fr: float = 1000, limit_pref_fr: float = 1.0, start_pref_fm: float = 10000, limit_pref_fm: float = 10.0, start_pref_v: float = 0.0, limit_pref_v: float = 0.0, start_pref_ae: float = 0.0, limit_pref_ae: float = 0.0, start_pref_pf: float = 0.0, limit_pref_pf: float = 0.0, relative_f: Optional[float] = None, enable_atom_ener_coeff: bool = False, use_spin: Optional[list] = None)

Bases: *Loss*

Methods

build(learning_rate, natoms, model_dict, ...)	Build the loss function graph.
eval(sess, feed_dict, natoms)	Eval the loss function.

print_header	
print_on_training	

build(learning_rate, natoms, model_dict, label_dict, suffix)

Build the loss function graph.

Parameters

learning_rate
[tf.Tensor] learning rate

natoms
[tf.Tensor] number of atoms

model_dict
[dict[str, tf.Tensor]] A dictionary that maps model keys to tensors

label_dict
[dict[str, tf.Tensor]] A dictionary that maps label keys to tensors

suffix
[str] suffix

Returns

tf.Tensor
the total squared loss

dict[str, tf.Tensor]
A dictionary that maps loss keys to more loss tensors

eval(sess, feed_dict, natoms)

Eval the loss function.

Parameters

sess
[tf.Session] TensorFlow session

feed_dict
[dict[`tf.placeholder`, `tf.Tensor`]] A dictionary that maps graph elements to values

natoms
[`tf.Tensor`] number of atoms

Returns

`dict`
A dictionary that maps keys to values. It should contain key natoms

`print_header``()`

`print_on_training``(tb_writer, cur_batch, sess, natoms, feed_dict_test, feed_dict_batch)`

`class deepmd.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)`

Bases: *`Loss`*

Standard loss function for DP models.

Parameters

enable_atom_ener_coeff
[bool] if true, the energy will be computed as sum_i c_i E_i

Methods

`build`(learning_rate, natoms, model_dict, ...)	Build the loss function graph.
`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

deepmd.loss.loss module

class deepmd.loss.loss.Loss

Bases: `object`

The abstract class for the loss function.

Methods

build(learning_rate, natoms, model_dict, ...)	Build the loss function graph.
eval(sess, feed_dict, natoms)	Eval the loss function.

abstract build(learning_rate: Tensor, natoms: Tensor, model_dict: Dict[str, Tensor], label_dict: Dict[str, Tensor], suffix: str) → Tuple[Tensor, Dict[str, Tensor]]

Build the loss function graph.

Parameters

learning_rate
[`tf.Tensor`] learning rate

natoms
[`tf.Tensor`] number of atoms

model_dict
[`dict[str, tf.Tensor]`] A dictionary that maps model keys to tensors

label_dict
[`dict[str, tf.Tensor]`] A dictionary that maps label keys to tensors

suffix
[`str`] suffix

Returns

`tf.Tensor`
the total squared loss

`dict[str, tf.Tensor]`
A dictionary that maps loss keys to more loss tensors

abstract eval(sess: 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

build(learning_rate, natoms, model_dict, ...)	Build the loss function graph.
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

deepmd.model package

`class deepmd.model.`**`DOSModel`**(descrpt, fitting, typeebd=None, type_map: Optional[List[str]] = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01)

Bases: *`Model`*

DOS model.

Parameters

descrpt
Descriptor

fitting
Fitting net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
Number of frames used for data statistic

data_stat_protect
Protect parameter for atomic energy regression

Methods

build(coord_, atype_, natoms, box, mesh, ...)	Build the model.
`build_descrpt`(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
init_variables(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_numb_dos	
get_rcut	
get_type_map	

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 to the checkpoint and meta file

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)

get_ntypes()

get_numb_dos()

get_rcut()

get_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 = 'dos'

class deepmd.model.**DipoleModel**(*args, **kwargs)

Bases: *TensorModel*

Methods

build(coord_, atype_, natoms, box, mesh, ...)	Build the model.
build_descrpt(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
init_variables(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_out_size	
get_rcut	
get_sel_type	
get_type_map	

class deepmd.model.**EnerModel**(descrpt, fitting, typeebd=None, type_map: Optional[List[str]] = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01, use_srtab: Optional[str] = None, smin_alpha: Optional[float] = None, sw_rmin: Optional[float] = None, sw_rmax: Optional[float] = None, spin: Optional[Spin] = None)

Bases: *Model*

Energy model.

Parameters

descrpt
Descriptor

fitting
: Fitting net

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
: Number of frames used for data statistic

data_stat_protect
: Protect parameter for atomic energy regression

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

smin_alpha
: The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when use_srtab is provided.

sw_rmin
: The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

sw_rmin
: The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

Methods

build(coord_, atype_, natoms, box, mesh, ...)	Build the model.
`build_descrpt`(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
init_variables(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_rcut	
get_type_map	
natoms_match	
natoms_not_match	

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 to the checkpoint and meta file

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)

get_ntypes()

get_rcut()

get_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 = 'ener'

natoms_match(force, natoms)

natoms_not_match(force, natoms, atype) class deepmd.model.GlobalPolarModel(*args, **kwargs)

Bases: *TensorModel*

Methods

build(coord_, atype_, natoms, box, mesh, ...)	Build the model.
build_descrpt(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
init_variables(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_out_size	
get_rcut	
get_sel_type	
get_type_map	

class deepmd.model.MultiModel(descrpt, fitting_dict, fitting_type_dict, typeebd=None, type_map: Optional[List[str]] = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01, use_srtab: Optional[str] = None, smin_alpha: Optional[float] = None, sw_rmin: Optional[float] = None, sw_rmax: Optional[float] = None)

Bases: *Model*

Multi-task model.

Parameters

descrpt
Descriptor

fitting_dict
Dictionary of fitting nets

fitting_type_dict
Dictionary of types of fitting nets

typeebd
Type embedding net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
Number of frames used for data statistic

data_stat_protect
Protect parameter for atomic energy regression

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

smin_alpha
: The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when use_srtab is provided.

sw_rmin
: The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

sw_rmin
: The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

Methods

build(coord_, atype_, natoms, box, mesh, ...)	Build the model.
`build_descrpt`(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
init_variables(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_rcut	
get_type_map	

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 to the checkpoint and meta file

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)

`get_ntypes`()

`get_rcut`()

`get_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'`

`class deepmd.model.`**`PolarModel`**(*args, **kwargs)

Bases: *`TensorModel`*

Methods

`build`(coord_, atype_, natoms, box, mesh, ...)	Build the model.
`build_descrpt`(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
`init_variables`(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_out_size	
get_rcut	
get_sel_type	
get_type_map	

`class deepmd.model.`**`WFCModel`**(*args, **kwargs)

Bases: *`TensorModel`*

Methods

`build`(coord_, atype_, natoms, box, mesh, ...)	Build the model.
`build_descrpt`(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
`init_variables`(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_out_size	
get_rcut	
get_sel_type	
get_type_map	

Submodules

deepmd.model.dos module

`class deepmd.model.dos.DOSModel`(descrpt, fitting, typeebd=None, type_map: Optional[List[str]] = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01)

Bases: *Model*

DOS model.

Parameters

descrpt
: Descriptor

fitting
: Fitting net

type_map
: Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
: Number of frames used for data statistic

data_stat_protect
: Protect parameter for atomic energy regression

Methods

build(coord_, atype_, natoms, box, mesh, ...)	Build the model.
`build_descrpt`(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
init_variables(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_numb_dos	
get_rcut	
get_type_map	

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 to the checkpoint and meta file

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)

get_ntypes()

get_numb_dos()

get_rcut()

get_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 = 'dos'

deepmd.model.ener module

class deepmd.model.ener.EnerModel(descrpt, fitting, typeebd=None, type_map: Optional[List[str]] = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01, use_srtab: Optional[str] = None, smin_alpha: Optional[float] = None, sw_rmin: Optional[float] = None, sw_rmax: Optional[float] = None, spin: Optional[Spin] = None)

Bases: *Model*

Energy model.

Parameters

descrpt
Descriptor

fitting
Fitting net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
Number of frames used for data statistic

data_stat_protect
Protect parameter for atomic energy regression

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

smin_alpha
The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when use_srtab is provided.

sw_rmin
The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

sw_rmin
: The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

Methods

build(coord_, atype_, natoms, box, mesh, ...)	Build the model.
`build_descrpt`(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
init_variables(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_rcut	
get_type_map	
natoms_match	
natoms_not_match	

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 to the checkpoint and meta file

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)

get_ntypes()

get_rcut()

get_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 = 'ener'

natoms_match(force, natoms)

natoms_not_match(force, natoms, atype)

deepmd.model.model module

class deepmd.model.model.Model

Bases: `ABC`

Methods

build(coord_, atype_, natoms, box, mesh, ...)	Build the model.
build_descrpt(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
init_variables(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

abstract build(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 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 to the checkpoint and meta file

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

`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

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 to the checkpoint and meta file

suffix
[str, optional] The suffix of the scope

reuse
[bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns

tf.Tensor
The descriptor tensor

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

deepmd.model.model_stat module

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

class deepmd.model.multi.MultiModel(descrpt, fitting_dict, fitting_type_dict, typeebd=None, type_map: Optional[List[str]] = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01, use_srtab: Optional[str] = None, smin_alpha: Optional[float] = None, sw_rmin: Optional[float] = None, sw_rmax: Optional[float] = None)

Bases: *Model*

Multi-task model.

Parameters

descrpt
Descriptor

fitting_dict
Dictionary of fitting nets

fitting_type_dict
Dictionary of types of fitting nets

typeebd
Type embedding net

type_map
Mapping atom type to the name (str) of the type. For example type_map[1] gives the name of the type 1.

data_stat_nbatch
Number of frames used for data statistic

data_stat_protect
Protect parameter for atomic energy regression

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

smin_alpha
The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when use_srtab is provided.

sw_rmin
The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

sw_rmin
The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use_srtab is provided.

Methods

build(coord_, atype_, natoms, box, mesh, ...)	Build the model.
build_descrpt(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
init_variables(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_rcut	
get_type_map	

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 to the checkpoint and meta file

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)

get_ntypes()

get_rcut()

get_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'

deepmd.model.tensor module

class deepmd.model.tensor.**DipoleModel**(*args, **kwargs)

Bases: *TensorModel*

Methods

build(coord_, atype_, natoms, box, mesh, ...)	Build the model.
build_descrpt(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
init_variables(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_out_size	
get_rcut	
get_sel_type	
get_type_map	

class deepmd.model.tensor.**GlobalPolarModel**(*args, **kwargs)

Bases: *TensorModel*

Methods

`build`(coord_, atype_, natoms, box, mesh, ...)	Build the model.
`build_descrpt`(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
`init_variables`(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_out_size	
get_rcut	
get_sel_type	
get_type_map	

`class deepmd.model.tensor.PolarModel`(*args, **kwargs)

Bases: *TensorModel*

Methods

`build`(coord_, atype_, natoms, box, mesh, ...)	Build the model.
`build_descrpt`(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
`init_variables`(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_out_size	
get_rcut	
get_sel_type	
get_type_map	

`class deepmd.model.tensor.TensorModel`(tensor_name: str, descrpt, fitting, typeebd=None, type_map: Optional[List[str]] = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01)

Bases: *Model*

Tensor model.

Parameters

tensor_name
: Name of the tensor.

descrpt
: Descriptor

fitting
: Fitting net

typeebd
: 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

build(coord_, atype_, natoms, box, mesh, ...)	Build the model.
`build_descrpt`(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
init_variables(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_out_size	
get_rcut	
get_sel_type	
get_type_map	

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 to the checkpoint and meta file

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)

`get_ntypes`()

`get_out_size`()

`get_rcut`()

`get_sel_type`()

`get_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

`build`(coord_, atype_, natoms, box, mesh, ...)	Build the model.
`build_descrpt`(coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
`init_variables`(graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.

data_stat	
get_ntypes	
get_out_size	
get_rcut	
get_sel_type	
get_type_map	

deepmd.nvnmd package

Subpackages

deepmd.nvnmd.data package

nvnmd.data ==========.

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

deepmd.nvnmd.data.data module

deepmd.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.descrpt2r4(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(type_i, type_input, inputs_i, is_exclude, activation_fn, bavg, stddev, trainable, suffix, seed, seed_shift, uniform_seed, filter_neuron, filter_precision, filter_resnet_dt, embedding_net_variables)`

Replace se_a.py/DescrptSeA/_filter_lower.

deepmd.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 relastionship of atomic distance, cutoff function, and embedding matrix.

three mapping table will be built:

$$
\begin{aligned}
&r^2_{ji} \rightarrow s_{ji}\\
&r^2_{ji} \rightarrow h_{ji}\\
&r^2_{ji} \rightarrow \mathcal{G}_{ji}
\end{aligned}
$$

where s_{ji} is cut-off function, $h_{ji} = \frac{s(r_{ji})}{r_{ji}}$, and $\mathcal{G}_{ji}$ is embedding matrix.

The mapping funciton can be define as:

$$
\begin{aligned}
&y = f(x) = y_k + (x - x_k) * dy_k\\
&y_k = f(x_k)\\
&dy_k = \frac{f(x_{k+1}) - f(x_k)}{dx}\\
&x_k \leq x < x_{k+1}\\
&x_k = k * dx
\end{aligned}
$$

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

build_grad(x, y, Nr, Nc)	: Build gradient of tensor y of x.
build_map_coef(cfgs, x, ys, grads, ...[, rank])	Build mapping table coefficient cfgs: cfg list cfg = x0, x1, dx.
build_s2g(s)	Build s->G s is switch function G is embedding net output.
build_s2g_grad()	Build gradient of G with respect to s.
build_u2s(r2)	Build tensor s, s=s(r2).
build_u2s_grad()	Build gradient of s with respect to u (r^2).
mapping(x, dic_map, cfgs[, rank])	Evaluate value by mapping table operation of tensorflow.
mapping2(x, dic_map, cfgs[, rank])	Evaluate value by mapping table of numpy.
plot_lines(x, dic1[, dic2])	Plot lines to see accuracy.
run_s2g()	Build s-> 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.

build_map	

build_grad(x, y, Nr, Nc)

: Build gradient of tensor y of x.

build_map()

build_map_coef(cfgs, x, ys, grads, grad_grads, Nr, Nc, rank=4)

Build mapping table coefficient cfgs: cfg list cfg = x0, x1, dx.

coef2: a x + b = y / b = y0 a = (y1 - y0) / L

coef4: a x^3 + b x^2 + c x + d = y: / d = y0 | c = y0' | b = (3 y1 - dx dy' - 2dx y0' - 3y0) / dx^2 a = (dx y1' - 2 y1 + dx y0' + 2 y0) / dx^3

build_s2g(s)

Build s->G s is switch function G is embedding net output.

build_s2g_grad()

Build gradient of G with respect to s.

build_u2s(r2)

Build tensor s, s=s(r2).

build_u2s_grad()

Build gradient of s with respect to u (r^2).

mapping(x, dic_map, cfgs, rank=4)

Evaluate value by mapping table operation of tensorflow.

mapping2(x, dic_map, cfgs, rank=4)

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_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:

```
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_dscp()	Wrap the configuration of descriptor.
wrap_fitn()	Wrap the weights of fitting net.
wrap_map()	Wrap the mapping table of embedding network.
wrap_weight(weight, NBIT_DISP, NBIT_WEIGHT)	weight: weights of fittingNet NBIT_DISP: nbits of exponent of weight max value NBIT_WEIGHT: nbits of mantissa of weights.

wrap	
wrap_bias	
wrap_head	

wrap()

wrap_bias(bias, NBIT_DATA, NBIT_DATA_FL)

wrap_dscp()

Wrap the configuration of descriptor.

[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

wrap_fitn()

Wrap the weights of fitting net.

wrap_head(nhs, nws)

wrap_map()

Wrap the mapping table of embedding network.

wrap_weight(weight, NBIT_DISP, NBIT_WEIGHT)

weight: weights of fittingNet NBIT_DISP: nbits of exponent of weight max value NBIT_WEIGHT: nbits of mantissa of weights.

deepmd.nvnmd.entrypoints.**save_weight**(sess, file_name: str = 'nvnmd/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.

```
train_attr/min_nbor_dist
descrpt_attr/t_avg:0
descrpt_attr/t_std:0
filter_type_{atom i}/matrix_{layer l}_{atomj}:0
filter_type_{atom i}/bias_{layer l}_{atomj}:0
layer_{layer l}_type_{atom i}/matrix:0
layer_{layer l}_type_{atom i}/bias:0
final_layer_type_{atom i}/matrix:0
final_layer_type_{atom i}/bias:0
```

deepmd.nvnmd.entrypoints.freeze.**save_weight**(sess, file_name: str = 'nvnmd/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 relastionship of atomic distance, cutoff function, and embedding matrix.

three mapping table will be built:

$$r^2_{ji} \rightarrow s_{ji}$$
$$r^2_{ji} \rightarrow h_{ji}$$
$$r^2_{ji} \rightarrow \mathcal{G}_{ji}$$

where s_{ji} is cut-off function, $h_{ji} = \frac{s(r_{ji})}{r_{ji}}$, and $\mathcal{G}_{ji}$ is embedding matrix.

The mapping funciton can be define as:

$$y = f(x) = y_k + (x - x_k) * dy_k$$
$$y_k = f(x_k)$$
$$dy_k = \frac{f(x_{k+1}) - f(x_k)}{dx}$$
$$x_k \leq x < x_{k+1}$$
$$x_k = k * dx$$

where dx is interpolation interval.

Parameters

config_file
: input file name an .npy file containing the configuration information of NVNMD model

weight_file
: input file name an .npy file containing the weights of NVNMD model

map_file
: output file name an .npy file containing the mapping tables of NVNMD model

References

DOI: 10.1038/s41524-022-00773-z

Methods

build_grad(x, y, Nr, Nc)	: Build gradient of tensor y of x.
build_map_coef(cfgs, x, ys, grads, ...[, rank])	Build mapping table coefficient cfgs: cfg list cfg = x0, x1, dx.
build_s2g(s)	Build s->G s is switch function G is embedding net output.
build_s2g_grad()	Build gradient of G with respect to s.
build_u2s(r2)	Build tensor s, s=s(r2).
build_u2s_grad()	Build gradient of s with respect to u (r^2).
mapping(x, dic_map, cfgs[, rank])	Evaluate value by mapping table operation of tensorflow.
mapping2(x, dic_map, cfgs[, rank])	Evaluate value by mapping table of numpy.
plot_lines(x, dic1[, dic2])	Plot lines to see accuracy.
run_s2g()	Build s-> 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.

build_map	

build_grad(x, y, Nr, Nc)

: Build gradient of tensor y of x.

build_map()

build_map_coef(cfgs, x, ys, grads, grad_grads, Nr, Nc, rank=4)

Build mapping table coefficient cfgs: cfg list cfg = x0, x1, dx.

coef2: a x + b = y / b = y0 a = (y1 - y0) / L

coef4: a x^3 + b x^2 + c x + d = y: / d = y0 | c = y0' | b = (3 y1 - dx dy' - 2dx y0' - 3y0) / dx^2 a = (dx y1' - 2 y1 + dx y0' + 2 y0) / dx^3

build_s2g(s)

Build s->G s is switch function G is embedding net output.

build_s2g_grad()

Build gradient of G with respect to s.

build_u2s(r2)

Build tensor s, s=s(r2).

build_u2s_grad()

Build gradient of s with respect to u (r^2).

mapping(x, dic_map, cfgs, rank=4)

Evaluate value by mapping table operation of tensorflow.

mapping2(x, dic_map, cfgs, rank=4)

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_u2s()

Build u->s graph and run it to get value of mapping table.

deepmd.nvnmd.entrypoints.mapt.**mapt**(*, 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

deepmd.nvnmd.entrypoints.train.**normalized_input**(fn, PATH_CNN, CONFIG_CNN)

Normalize a input script file for continuous neural network.

deepmd.nvnmd.entrypoints.train.**normalized_input_qnn**(jdata, PATH_QNN, CONFIG_CNN, WEIGHT_CNN, MAP_CNN)

Normalize a input script file for quantize neural network.

deepmd.nvnmd.entrypoints.train.**train_nvnmd**(*, INPUT: str, restart: Optional[str], step: str, **kwargs)

deepmd.nvnmd.entrypoints.wrap module

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:

```
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_dscp()	Wrap the configuration of descriptor.
wrap_fitn()	Wrap the weights of fitting net.
wrap_map()	Wrap the mapping table of embedding network.
wrap_weight(weight, NBIT_DISP, NBIT_WEIGHT)	weight: weights of fittingNet NBIT_DISP: nbits of exponent of weight max value NBIT_WEIGHT: nbits of mantissa of weights.

wrap	
wrap_bias	
wrap_head	

wrap()

wrap_bias(bias, NBIT_DATA, NBIT_DATA_FL)

wrap_dscp()

Wrap the configuration of descriptor.

[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

wrap_fitn()

Wrap the weights of fitting net.

wrap_head(nhs, nws)

wrap_map()

Wrap the mapping table of embedding network.

wrap_weight(weight, NBIT_DISP, NBIT_WEIGHT)

weight: weights of fittingNet NBIT_DISP: nbits of exponent of weight max value NBIT_WEIGHT: nbits of mantissa of weights.

deepmd.nvnmd.entrypoints.wrap.**wrap**(*, nvnmd_config: Optional[str] = 'nvnmd/config.npy', nvnmd_weight: Optional[str] = 'nvnmd/weight.npy', nvnmd_map: Optional[str] = 'nvnmd/map.npy', nvnmd_model: Optional[str] = 'nvnmd/model.pb', **kwargs)

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 initialed 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(slbin, nfull)	Extend the element of list (slbin) to the length (nfull).
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 attched element of list is 0.
flt2bin(data, nbit_expo, 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(slbin, 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(slbin, 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_expo_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(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

extent 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 attched element of list is 0.

such as, when

slbin = [‘10010’,’10100’],
nfull = 4

extent 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)

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

load([file_name, default_value])	Load .txt file into string list.
save([file_name, data])	Save string list into .txt file.

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.`**get_filter_weight**(weights: dict, spe_i: int, spe_j: int, layer_l: int)

Get weight and bias of embedding network.

Parameters

weights
: [dict] weights

spe_i
: [int] special order of central atom i 0~ntype-1

spe_j
: [int] special order of neighbor atom j 0~ntype-1

layer_l
: layer order in embedding network 1~nlayer

`deepmd.nvnmd.utils.`**get_fitnet_weight**(weights: dict, spe_i: int, layer_l: int, nlayer: int = 10)

Get weight and bias of fitting network.

Parameters

weights
: [dict] weights

spe_i
: [int] special order of central atom i 0~ntype-1

layer_l
: [int] layer order in embedding network 0~nlayer-1

nlayer
: [int] number of layers

`deepmd.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

`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

disp_message()	Display the log of NVNMD.
get_deepmd_jdata()	Generate input script with member element one by one.
get_dscp_jdata()	Generate model/descriptor in input script.
get_fitn_jdata()	Generate model/fitting_net 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_training_jdata()	Generate training in input script.
init_ctrl(jdata[, jdata_parent])	Initial members about control signal.
init_dpin(jdata[, jdata_parent])	Initial members about other deepmd input.
init_dscp(jdata[, jdata_parent])	Initial members about descriptor.
init_fitn(jdata[, jdata_parent])	Initial members about fitting network.
init_from_config(jdata)	Initial member element one by one.
init_from_deepmd_input(jdata)	Initial members with input script of deepmd.
init_from_jdata([jdata])	Initial this class with jdata loaded from input script.
init_nbit(jdata[, jdata_parent])	Initial members about quantification precision.
init_net_size()	Initial net_size.
init_size(jdata[, jdata_parent])	Initial members about ram capacity.
init_train_mode([mod])	Configure for taining cnn or qnn.
init_value()	Initial member with dict.
save([file_name])	Save all configuration to file.
update_config()	Update config from dict.

get_s_range	

disp_message()
: Display the log of NVNMD.

get_deepmd_jdata()
: Generate input script with member element one by one.

get_dscp_jdata()
: Generate model/descriptor in input script.

get_fitn_jdata()
: Generate model/fitting_net 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_training_jdata()
: Generate training in input script.

init_ctrl(jdata: dict, jdata_parent: dict = {}) → dict
: Initial members about control signal.

init_dpin(jdata: dict, jdata_parent: dict = {}) → dict
: Initial members about other deepmd input.

init_dscp(jdata: dict, jdata_parent: dict = {}) → dict
: Initial members about descriptor.

init_fitn(jdata: dict, jdata_parent: dict = {}) → dict
: Initial members about fitting network.

init_from_config(jdata)
: Initial member element one by one.

init_from_deepmd_input(jdata)
: Initial members with input script of deepmd.

init_from_jdata(jdata: dict = {})
: Initial this class with jdata loaded from input script.

init_nbit(jdata: dict, jdata_parent: dict = {}) → dict
: Initial members about quantification precision.

init_net_size()
: Initial net_size.

init_size(jdata: dict, jdata_parent: dict = {}) → dict
: Initial members about ram capacity.

init_train_mode(mod='cnn')
: Configure for taining cnn or qnn.

init_value()
: Initial member with dict.

save(file_name=None)
: Save all configuration to file.

update_config()
: Update config from dict.

deepmd.nvnmd.utils.encode module

class deepmd.nvnmd.utils.encode.**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(slbin, nfull)	Extend the element of list (slbin) to the length (nfull).
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 attched element of list is 0.
flt2bin(data, nbit_expo, 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(slbin, 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(slbin, 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_expo_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 $[0, 2^n bit - 1]$ 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

extent 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 attched element of list is 0.

such as, when

slbin = ['10010','10100'],
nfull = 4

extent 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)

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)

deepmd.nvnmd.utils.fio module

class deepmd.nvnmd.utils.fio.**Fio**
Bases: object

Basic class for FIO.

Methods

create_file_path	
exits	
get_file_list	
is_file	
is_path	
mkdir	

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

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.fio.**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.fio.**FioJsonDic**

Bases: `object`

Input and output for .json file containing dictionary.

Methods

load([file_name, default_value])	Load .json file into dict.
save([file_name, dic])	Save dict into .json file.

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

load	
save	

load(file_name='', default_value={})

save(file_name='', dic={})

class deepmd.nvnmd.utils.fio.**FioTxt**
Bases: `object`

Input and output for .txt file with string.

Methods

load([file_name, default_value])	Load .txt file into string list.
save([file_name, data])	Save string list into .txt file.

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_wb`**(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_weight`**(weights: dict, spe_i: int, spe_j: int, layer_l: int)

Get weight and bias of embedding network.

Parameters

weights
[`dict`] weights

spe_i
[`int`] special order of central atom i 0~ntype-1

spe_j
[`int`] special order of neighbor atom j 0~ntype-1

layer_l
layer order in embedding network 1~nlayer

`deepmd.nvnmd.utils.weight.`**`get_fitnet_weight`**(weights: dict, spe_i: int, layer_l: int, nlayer: int = 10)

Get weight and bias of fitting network.

Parameters

weights
: [dict] weights

spe_i
: [int] special order of central atom i 0~ntype-1

layer_l
: [int] layer order in embedding network 0~nlayer-1

nlayer
: [int] number of layers

`deepmd.nvnmd.utils.weight.get_normalize`(weights: dict)

Get normalize parameter (avg and std) of s_{ji}.

`deepmd.nvnmd.utils.weight.get_weight`(weights, key)

Get weight value according to key.

deepmd.op package

This module will house cust Tf OPs after CMake installation.

`deepmd.op.import_ops`()

Import all custom TF ops that are present in this submodule.

Notes

Initialy this subdir is unpopulated. CMake will install all the op module python files and shared libs.

deepmd.train package

Submodules

deepmd.train.run_options module

Module taking care of important package constants.

`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 distribured training it is true for tha main MPI process in serail it is always true

world_size: int
: total worker count

my_rank: int
: index of the MPI task

nodename: str
: name of the node

node_list_
: [List[str]] the list of nodes of the current mpirun

my_device: str
: deviice type - gpu or cpu

Methods

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

deepmd.train.trainer module

class deepmd.train.trainer.**DPTrainer**(jdata, run_opt, is_compress=False)
: Bases: object

Methods

save_compressed()	Save the compressed graph.

build	
eval_single_list	
get_evaluation_results	
get_feed_dict	
get_global_step	
print_header	
print_on_training	
save_checkpoint	
train	
valid_on_the_fly	

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

`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)

Bases: `object`

Class for a data system.

It loads data from hard disk, and mantains the data as a data_dict

Parameters

sys_path
Path to the data system

set_prefix
Prefix for the directories of different sets

shuffle_test
If the test data are shuffled

type_map
: Gives the name of different atom types

optional_type_map
: If the type_map.raw in each system is optional

modifier
: Data modifier that has the method modify_data

trn_all_set
: Use all sets as training dataset. Otherwise, if the number of sets is more than 1, the last set is left for test.

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.
reduce(key_out, key_in)	Generate a new item from the reduction of another atom.

reset_get_batch	

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.

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: float, 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')

Bases: `object`

Class for manipulating many data systems.

It is implemented with the help of DeepmdData

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.

compute_energy_shift	
get_data_dict	
print_summary	
set_sys_probs	

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.
Default value of data

`add_dict`(adict: dict) → None

Add items to the data system by a dict. adict should have items like .. code-block:: python.

adict[key] = {
“ndof”: ndof, “atomic”: atomic, “must”: must, “high_prec”: high_prec, “type_sel”: type_sel, “repeat”: repeat,
}

For the explaination of the keys see add

`compute_energy_shift`(rcond=0.001, key='energy')

`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_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 dat of system with index sys_idx will be returned. If is None, the currently selected system will be returned.

n_test
Number of test data. If set to -1 all test data will be get.

get_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')

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 α is the learning rate, α_0 is the starting learning rate, λ is the decay rate, and τ is the decay steps.

Parameters

start_lr
Starting learning rate α_0

stop_lr
Stop learning rate α_1

decay_steps
Learning rate decay every this number of steps τ

decay_rate
The decay rate λ. If stop_step is provided in build, then it will be determined automatically and overwritten.

Methods

build(global_step[, stop_step])	Build the learning rate.
start_lr()	Get the start lr.
value(step)	Get the lr at a certain step.

build(global_step: Tensor, stop_step: Optional[int] = None) → Tensor
Build the learning rate.

Parameters

global_step
The tf Tensor prividing the global training step

stop_step
The stop step. If provided, the decay_rate will be determined automatically and overwritten.

Returns

learning_rate
The learning rate

start_lr() → float
Get the start lr.

value(step: int) → float
Get the lr at a certain step.

class deepmd.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) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

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) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

class deepmd.utils.**Plugin**

Bases: object

A class to register and restore plugins.

Examples

```
>>> plugin = Plugin()
>>> @plugin.register("xx")
    def xxx():
        pass
>>> print(plugin.plugins['xx'])
```

Attributes

plugins
[Dict[str, object]] plugins

Methods

get_plugin(key)	Visit a plugin by key.
register(key)	Register a plugin.

get_plugin(key) → object

Visit a plugin by key.

Parameters

key
[str] key of the plugin

Returns

`object`
the plugin

register(key: str) → Callable[[object], object]
Register a plugin.

Parameters

key
[`str`] key of the plugin

Returns

`Callable`[[`object`], `object`]
decorator

class deepmd.utils.**PluginVariant**(*args, **kwargs)
Bases: `object`

A class to remove type from input arguments.

Submodules

deepmd.utils.argcheck module

class deepmd.utils.argcheck.**ArgsPlugin**
Bases: `object`

Methods

get_all_argument([exclude_hybrid])	Get all arguments.
register(name[, alias])	Register a descriptor argument plugin.

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

```
>>> some_plugin = ArgsPlugin()
>>> @some_plugin.register("some_descrpt")
    def descrpt_some_descrpt_args():
        return []
```

deepmd.utils.argcheck.**descrpt_hybrid_args**()

deepmd.utils.argcheck.**descrpt_local_frame_args**()

deepmd.utils.argcheck.**descrpt_se_a_args**()

deepmd.utils.argcheck.**descrpt_se_a_mask_args**()

deepmd.utils.argcheck.**descrpt_se_a_tpe_args**()

deepmd.utils.argcheck.**descrpt_se_atten_args**()

deepmd.utils.argcheck.**descrpt_se_r_args**()

deepmd.utils.argcheck.**descrpt_se_t_args**()

deepmd.utils.argcheck.**descrpt_variant_type_args**(exclude_hybrid: bool = False) → Variant

deepmd.utils.argcheck.**fitting_dipole**()

deepmd.utils.argcheck.**fitting_dos**()

deepmd.utils.argcheck.**fitting_ener**()

deepmd.utils.argcheck.**fitting_polar**()

deepmd.utils.argcheck.**fitting_variant_type_args**()

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.**learning_rate_args**()

deepmd.utils.argcheck.**learning_rate_dict_args**()

deepmd.utils.argcheck.**learning_rate_exp**()

deepmd.utils.argcheck.**learning_rate_variant_type_args**()

deepmd.utils.argcheck.**limit_pref**(item)

deepmd.utils.argcheck.**list_to_doc**(xx)

deepmd.utils.argcheck.**loss_args**()

deepmd.utils.argcheck.**loss_dict_args**()

deepmd.utils.argcheck.**loss_dos**()

deepmd.utils.argcheck.**loss_ener**()

deepmd.utils.argcheck.**loss_ener_spin**()

deepmd.utils.argcheck.**loss_tensor**()

deepmd.utils.argcheck.**loss_variant_type_args**()

deepmd.utils.argcheck.**make_index**(keys)

deepmd.utils.argcheck.**make_link**(content, ref_key)

deepmd.utils.argcheck.**mixed_precision_args**()

deepmd.utils.argcheck.**model_args**()

deepmd.utils.argcheck.**model_compression**()

deepmd.utils.argcheck.**model_compression_type_args**()

deepmd.utils.argcheck.**modifier_dipole_charge**()

deepmd.utils.argcheck.**modifier_variant_type_args**()

deepmd.utils.argcheck.**normalize**(data)

deepmd.utils.argcheck.**normalize_data_dict**(data_dict)

deepmd.utils.argcheck.**normalize_fitting_net_dict**(fitting_net_dict)

deepmd.utils.argcheck.**normalize_fitting_weight**(fitting_keys, data_keys, fitting_weight=None)

deepmd.utils.argcheck.**normalize_learning_rate_dict**(fitting_keys, learning_rate_dict)

deepmd.utils.argcheck.**normalize_learning_rate_dict_with_single_learning_rate**(fitting_keys, learning_rate)

deepmd.utils.argcheck.**normalize_loss_dict**(fitting_keys, loss_dict)

deepmd.utils.argcheck.**normalize_multi_task**(data)

deepmd.utils.argcheck.**spin_args**()

deepmd.utils.argcheck.**start_pref**(item)

deepmd.utils.argcheck.**training_args**()

deepmd.utils.argcheck.**training_data_args**()

deepmd.utils.argcheck.**type_embedding_args**()

deepmd.utils.argcheck.**validation_data_args**()

deepmd.utils.batch_size module

class deepmd.utils.batch_size.AutoBatchSize(initial_batch_size: int = 1024, factor: float = 2.0)

Bases: `object`

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)	Excuate a method with given batch size.
execute_all(callable, total_size, natoms, ...)	Excuate a method with all given data.

execute(callable: Callable, start_index: int, natoms: int) → Tuple[int, tuple]

Excuate a method with given batch size.

Parameters

callable
[`Callable`] The method should accept the batch size and start_index as parameters, and returns executed batch size and data.

start_index
[`int`] start index

natoms
[`int`] natoms

Returns

`int`
executed batch size * number of atoms

`tuple`
result from callable, None if failing to execute

Raises

OutOfMemoryError
OOM when batch size is 1

execute_all(callable: Callable, total_size: int, natoms: int, *args, **kwargs) → Tuple[ndarray]

Excuate a method with all given data.

Parameters

callable
[`Callable`] The method should accept *args and **kwargs as input and return the similiar array.

total_size
[`int`] Total size

natoms
[`int`] The number of atoms

*args
Variable length argument list.

**kwargs
If 2D np.ndarray, assume the first axis is batch; otherwise do nothing.

deepmd.utils.compat module

Module providing compatibility between 0.x.x and 1.x.x input versions.

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, `optional`] whether to show deprecation warning, by default True

dump
[`Optional`[`Union`[`str`, `Path`]], `optional`] whether to dump converted file, by default None

Returns

`Dict`[`str`, `Any`]
converted output

deepmd.utils.compat.**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, `optional`] whether to show deprecation warning, by default True

dump
[`Optional`[`Union`[`str`, `Path`]], `optional`] whether to dump converted file, by default None

Returns

`Dict`[`str`, `Any`]
converted output

`deepmd.utils.compat.remove_decay_rate`(jdata: Dict[str, Any])

Convert decay_rate to stop_lr.

Parameters

jdata
[`Dict`[`str`, `Any`]] input data

`deepmd.utils.compat.update_deepmd_input`(jdata: Dict[str, Any], warning: bool = True, dump: Optional[Union[str, Path]] = None) → Dict[str, Any]

deepmd.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

`deepmd.utils.convert.convert_10_to_21`(input_model: str, output_model: str)

Convert DP 1.0 graph to 2.1 graph.

Parameters

input_model
[`str`] filename of the input graph

output_model
[`str`] filename of the output graph

`deepmd.utils.convert.`**`convert_12_to_21`**(input_model: str, output_model: str)

Convert DP 1.2 graph to 2.1 graph.

Parameters

input_model
[`str`] filename of the input graph

output_model
[`str`] filename of the output graph

`deepmd.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)

Convert DP 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.detect_model_version`(input_model: str)

Detect DP graph version.

Parameters

input_model
[`str`] filename of the input graph

deepmd.utils.data module

`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)

Bases: `object`

Class for a data system.

It loads data from hard disk, and mantains the data as a data_dict

Parameters

sys_path
Path to the data system

set_prefix
Prefix for the directories of different sets

shuffle_test
: If the test data are shuffled

type_map
: Gives the name of different atom types

optional_type_map
: If the type_map.raw in each system is optional

modifier
: Data modifier that has the method modify_data

trn_all_set
: Use all sets as training dataset. Otherwise, if the number of sets is more than 1, the last set is left for test.

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.
reduce(key_out, key_in)	Generate a new item from the reduction of another atom.

reset_get_batch	

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.

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.data_system module

class deepmd.utils.data_system.**DeepmdDataSystem**(systems: List[str], batch_size: int, test_size: int, rcut: float, 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')

Bases: `object`

Class for manipulating many data systems.

It is implemented with the help of DeepmdData

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.

compute_energy_shift	
get_data_dict	
print_summary	
set_sys_probs	

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.
Default value of data

add_dict(adict: dict) → None

Add items to the data system by a dict. adict should have items like .. code-block:: python.

adict[key] = {
“ndof”: ndof, “atomic”: atomic, “must”: must, “high_prec”: high_prec, “type_sel”: type_sel, “repeat”: repeat,

}

For the explaination of the keys see add

compute_energy_shift(rcond=0.001, key='energy')

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_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 dat of system with index sys_idx will be returned. If is None, the currently selected system will be returned.

n_test
Number of test data. If set to -1 all test data will be get.

get_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.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.finetune module

`deepmd.utils.finetune.`**`replace_model_params_with_pretrained_model`**(jdata: Dict[str, Any], pretrained_model: str)

Replace the model params in input script according to pretrained model.

Parameters

jdata
: [`Dict`[`str`, `Any`]] input script

pretrained_model
: [`str`] filename of the pretrained model

deepmd.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_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

`Dict`
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 tf.GraphDef object

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

deepmd.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

deepmd.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

deepmd.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

deepmd.utils.graph.**get_type_embedding_net_variables_from_graph_def**(graph_def: GraphDef, suffix: str = '') → Dict

Get the type embedding net variables with the given tf.GraphDef object.

Parameters

graph_def
: [tf.GraphDef] The input tf.GraphDef object

suffix
: [str, optional] The suffix of the scope

Returns

Dict
: The embedding net variables within the given tf.GraphDef object

deepmd.utils.graph.**load_graph_def**(model_file: str) → Tuple[Graph, GraphDef]

Load graph as well as the graph_def from the frozen model(model_file).

Parameters

model_file
: [str] The input frozen model path

Returns

tf.Graph
: The graph loaded from the frozen model

tf.GraphDef
: The graph_def loaded from the frozen model

deepmd.utils.learning_rate module

class deepmd.utils.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 α is the learning rate, α_0 is the starting learning rate, λ is the decay rate, and τ is the decay steps.

Parameters

start_lr
: Starting learning rate α_0

stop_lr
: Stop learning rate α_1

decay_steps
: Learning rate decay every this number of steps τ

decay_rate
: The decay rate λ. If stop_step is provided in build, then it will be determined automatically and overwritten.

Methods

build(global_step[, stop_step])	Build the learning rate.
start_lr()	Get the start lr.
value(step)	Get the lr at a certain step.

build(global_step: Tensor, stop_step: Optional[int] = None) → Tensor

Build the learning rate.

Parameters

global_step
: The tf Tensor prividing the global training step

stop_step
: The stop step. If provided, the decay_rate will be determined automatically and overwritten.

Returns

learning_rate
: The learning rate

start_lr() → float

Get the start lr.

value(step: int) → float

Get the lr at a certain step.

deepmd.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

pretrained_model
: [`str`] filename of the pretrained frozen multi-task united model

deepmd.utils.neighbor_stat module

class `deepmd.utils.neighbor_stat.`**`NeighborStat`**(ntypes: 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 neareest nbor distance between atoms, max nbor size of atoms and the output data range of the environment matrix.

Parameters

ntypes
: The num of atom types

rcut
: The cut-off radius

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) → 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

deepmd.utils.network module

`deepmd.utils.network.embedding_net`(xx, network_size, precision, activation_fn=<function tanh>, resnet_dt=False, name_suffix='', stddev=1.0, bavg=0.0, seed=None, trainable=True, uniform_seed=False, initial_variables=None, mixed_prec=None)

The embedding network.

The embedding network function $\mathcal{N}$ is constructed by is the composition of multiple layers $\mathcal{L}^{(i)}$:

$$\mathcal{N} = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \cdots \circ \mathcal{L}^{(1)}$$

A layer $\mathcal{L}$ is given by one of the following forms, depending on the number of nodes: [1]

$$\mathrm{y} = \mathcal{L}(\mathrm{x}; \mathrm{w}, \mathrm{b}) = \begin{cases} \boldsymbol{\phi}(\mathrm{x}^T\mathrm{w} + \mathrm{b}) + \mathrm{x}, & N_2 = N_1 \\ \boldsymbol{\phi}(\mathrm{x}^T\mathrm{w} + \mathrm{b}) + (\mathrm{x}, \mathrm{x}), & N_2 = 2N_1 \\ \boldsymbol{\phi}(\mathrm{x}^T\mathrm{w} + \mathrm{b}), & \text{otherwise} \end{cases}$$

where $\mathrm{x} \in \mathbb{R}^{N_1}$ is the input vector and $\mathrm{y} \in \mathbb{R}^{N_2}$ is the output vector. $\mathrm{w} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathrm{b} \in \mathbb{R}^{N_2}$ are weights and biases, respectively, both of which are trainable if trainable is True. $\boldsymbol{\phi}$ is the activation function.

Parameters

xx
: [`Tensor`] Input tensor x of shape [-1,1]

network_size
: [`list of int`] Size of the embedding network. For example [16,32,64]

precision:
: Precision of network weights. For example, tf.float64

activation_fn:
: Activation function $\boldsymbol{\phi}$

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

`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) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

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) * N_t / 2 + 1 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

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[ConfigProto] = None)

Bases: `object`

Run an op with data parallelism.

Parameters

builder
[`Callable`[..., `Tuple`[`Dict`[`str`, `tf.Tensor`], `Tuple`[`tf.Tensor`]]]] returns two objects: a dict which stores placeholders by key, and a tuple with the final op(s)

nthreads
[`int`, `optional`] the number of threads

config
[`tf.ConfigProto`, `optional`] tf.ConfigProto

Examples

```
>>> from deepmd.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()))
[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
```

Methods

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

class deepmd.utils.path.DPH5Path(path: str)

Bases: *DPPath*

The path class to data system (DeepmdData) for HDF5 files.

Parameters

path
: [`str`] path

Notes

OS - HDF5 relationship:
    directory - Group file - Dataset

Methods

glob(pattern)	Search path using the glob pattern.
is_dir()	Check if self is directory.
is_file()	Check if self is file.
load_numpy()	Load NumPy array.
load_txt([dtype])	Load NumPy array from text.
rglob(pattern)	This is like calling *DPPath.glob()* with **/ added in front of the given relative pattern.

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 *`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.`**`DPOSPath`**(path: str)
Bases: *`DPPath`*

The OS path class to data system (DeepmdData) for real directories.

Parameters

path
[`str`] path

Methods

glob(pattern)	Search path using the glob pattern.
is_dir()	Check if self is directory.
is_file()	Check if self is file.
load_numpy()	Load NumPy array.
load_txt(**kwargs)	Load NumPy array from text.
rglob(pattern)	This is like calling *DPPath.glob()* with **/ added in front of the given relative pattern.

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

glob(pattern)	Search path using the glob pattern.
is_dir()	Check if self is directory.
is_file()	Check if self is file.
load_numpy()	Load NumPy array.
load_txt(**kwargs)	Load NumPy array from text.
rglob(pattern)	This is like calling *DPPath.glob()* with **/ added in front of the given relative pattern.

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
[str] glob pattern

Returns

List[*DPPath*]
list of paths

deepmd.utils.plugin module

Base of plugin systems.

class deepmd.utils.plugin.Plugin

Bases: object

A class to register and restore plugins.

Examples

```
>>> plugin = Plugin()
>>> @plugin.register("xx")
    def xxx():
        pass
>>> print(plugin.plugins['xx'])
```

Attributes

plugins
[Dict[str, object]] plugins

Methods

get_plugin(key)	Visit a plugin by key.
register(key)	Register a plugin.

get_plugin(key) → object

Visit a plugin by key.

Parameters

key
[str] key of the plugin

Returns

object
the plugin

register(key: str) → Callable[[object], object]

Register a plugin.

Parameters

key
[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

deepmd.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.random.random(size=None)

Return random floats in the half-open interval [0.0, 1.0).

Parameters

size
Output shape.

Returns

np.ndarray
Arrays with results and their shapes.

deepmd.utils.random.seed(val: Optional[int] = None)

Seed the generator.

Parameters

val
[int] Seed.

deepmd.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

deepmd.utils.sess.run_sess(sess: Session, *args, **kwargs)

Run session with erorrs caught.

Parameters

sess
[tf.Session] TensorFlow Session

*args
Variable length argument list.

**kwargs
Arbitrary keyword arguments.

Returns

Any
the result of sess.run()

deepmd.utils.spin module

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

build([reuse, suffix])	Build the computational graph for the spin.
get_ntypes_spin()	Returns the number of atom types which contain spin.
get_spin_norm()	Returns the list of magnitude of atomic spin for each atom type.
get_use_spin()	Returns the list of whether to use spin for each atom type.
get_virtual_len()	Returns the list of distance between real atom and virtual atom for each atom type.

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

class deepmd.utils.tabulate.**DPTabulate**(descrpt: ~deepmd.descriptor.descriptor.Descriptor, neuron: ~typing.List[int], graph: ~tensorflow.python.framework.ops.Graph, graph_def: ~tensorflow.core.framework.graph_pb2.GraphDef, type_one_side: bool = False, exclude_types: ~typing.List[~typing.List[int]] = [], activation_fn: ~typing.Callable[[~tensorflow.python.framework.ops.Tensor], ~tensorflow.python.framework.ops.Tensor] = <function tanh>, suffix: str = '')

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 it's uniform stride The range of the first table is automatically detected by deepmd-kit, while the second table ranges from the first table's upper boundary(upper) to the extrapolate(parameter) * upper.

Parameters

descrpt
: Descriptor of the original model

neuron
: Number of neurons in each hidden layers of the embedding net
: *mathcalN*

graph
: [`tf.Graph`] The graph of the original model

graph_def
: [`tf.GraphDef`] The graph_def of the original model

type_one_side
: Try to build N_types tables. Otherwise, building N_types^2 tables

exclude_types
: [`List`[`List`[`int`]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

activation_function
: The activation function in the embedding net. Supported options are {"tanh","gelu"} in common.ACTIVATION_FN_DICT.

suffix
: [`str`, `optional`] The suffix of the scope

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

stride1
: The uniform stride of the second table

Returns

lower
[dict[str, int]] The lower boundary of environment matrix by net

upper
[dict[str, int]] The upper boundary of environment matrix by net

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)

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.
init_variables(graph, graph_def[, suffix])	Init the type embedding net variables with the given dict.

build(ntypes: int, reuse=None, suffix='')

Build the computational graph for the descriptor.

Parameters

ntypes
: Number of atom types.

reuse
: The weights in the networks should be reused when get the variable.

suffix
: Name suffix to identify this descriptor

Returns

`embedded_types`
: The computational graph for embedded types

`init_variables`(graph: Graph, graph_def: GraphDef, suffix='') → None

Init the type embedding net variables with the given dict.

Parameters

graph
: [`tf.Graph`] The input frozen model graph

graph_def
: [`tf.GraphDef`] The input frozen model graph_def

suffix
: Name suffix to identify this descriptor

`deepmd.utils.type_embed.`**`embed_atom_type`**(ntypes: int, natoms: 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:
: Number of types.

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

type_embedding:
: The type embedding. It has the shape of [ntypes, embedding_dim]

Returns

`atom_embedding`
: The embedded type of each atom. It has the shape of [numb_atoms, embedding_dim]

deepmd.utils.weight_avg module

deepmd.utils.weight_avg.**weighted_average**(errors: List[Dict[str, Tuple[float, float]]]) → Dict

Compute wighted average of prediction errors (MAE or RMSE) for model.

Parameters

errors
[List[Dict[str, Tuple[float, float]]]] List: the error of systems Dict: the error of quantities, name given by the key str: the name of the quantity, must starts with 'mae' or 'rmse' Tuple: (error, weight)

Returns

Dict
weighted averages

17.1.2 Submodules

17.1.3 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, **kwargs)

Bases: Calculator

Implementation of ASE deepmd calculator.

Implemented propertie are energy, forces and stress

Parameters

model
[Union[str, Path]] path to the model

label
[str, optional] calculator label, by default "DP"

type_dict
[Dict[str, int], optional] mapping of element types and their numbers, best left None and the calculator will infer this information from model, by default None

Examples

Compute potential energy

```
>>> 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

```
>>> 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`([atoms, properties, system_changes])	Run calculation with deepmd model.
`calculate_numerical_forces`(atoms[, d])	Calculate numerical forces using finite difference.
`calculate_numerical_stress`(atoms[, d, voigt])	Calculate numerical stress using finite difference.
`calculate_properties`(atoms, properties)	This method is experimental; currently for internal use.
`check_state`(atoms[, tol])	Check for any system changes since last calculation.
`get_magnetic_moments`([atoms])	Calculate magnetic moments projected onto atoms.
`get_property`(name[, atoms, allow_calculation])	Get the named property.
`get_stresses`([atoms])	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.

calculation_required	
export_properties	
get_atoms	
get_charges	
get_default_parameters	
get_dipole_moment	
get_forces	
get_magnetic_moment	
get_potential_energies	
get_potential_energy	
get_stress	
read_atoms	
todict	

calculate(atoms: Optional[Atoms] = None, properties: List[str] = ['energy', 'forces', 'virial'], system_changes: List[str] = ['positions', 'numbers', 'cell', 'pbc', 'initial_charges', 'initial_magmoms'])

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: List[str] = ['energy', 'free_energy', 'forces', 'virial', 'stress']

Properties calculator can handle (energy, forces, …)

name = 'DP'

17.1.4 deepmd.common module

Collection of functions and classes used throughout the whole package.

deepmd.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 keyworrd applies to per atom quantity or not, by default False

must
: [bool, `optional`] specifi if the *.npy data file must exist, by default False

high_prec
: [bool, `optional`] if true load data to np.float64 else np.float32, by default False

type_sel
: [bool, `optional`] select only certain type of atoms, by default None

repeat
: [`int`, `optional`] if specify repaeat data repeat times, by default 1

default
: [`float`, `optional`, default=0.] default value of data

dtype
: [`np.dtype`, `optional`] the dtype of data, overwrites high_prec if provided

`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

Original paper https://arxiv.org/abs/1606.08415

`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

Original paper https://arxiv.org/abs/1606.08415

`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]
: correspondingg 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 presicion 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 acorresponding 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`(jdata: Dict[str, _DICT_VAL], key: str, deprecated_key: List[str] = []) → _DICT_VAL

Assert that supplied dictionary conaines specified key.

Returns

`_DICT_VAL`
value that was store unde supplied key

Raises

`RuntimeError`
    if the key is not present

`deepmd.common.make_default_mesh`(test_box: ndarray, cell_size: float = 3.0) → ndarray

Get number of cells of size=`cell_size` fit into average box.

Parameters

test_box
    [`np.ndarray`] numpy array with cells of shape Nx9

cell_size
    [`float`, `optional`] length of one cell, by default 3.0

Returns

`np.ndarray`
    mesh for supplied boxes, how many cells fit in each direction

`deepmd.common.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

`deepmd.common.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 specifing type for each atoms as integer

select_types
    [`np.ndarray`] types of atoms you want to find indices for

Returns

`np.ndarray`
    indices of types of atoms defined by select_types in atom_types array

Warning: select_types array will be sorted before finding indices in atom_types

17.1.5 deepmd.env module

Module that sets tensorflow working environment and exports inportant 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.

17.1.6 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) → List[str]

Get library path from a module.

Parameters

module
[`str`] The module name.

Returns

`list[str]`
The library path.

`deepmd.lmp.get_op_dir()` → str

Get the directory of the deepmd-kit OP library.

CHAPTER

EIGHTEEN

OP API

18.1 op_module

Python wrappers around TensorFlow ops.

This file is MACHINE GENERATED! Do not edit.

`deepmd.env.op_module.`**`AddFltNvnmd`**(x, w, name=None)

TODO: add doc.

Parameters

- **`x`** – A Tensor. Must be one of the following types: float32, float64.
- **`w`** – A Tensor. Must have the same type as x.
- **`name`** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`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.
- **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.`**`DescrptSeAEf`**(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.`**`DescrptSeAEfPara`**(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **`coord`** – A Tensor. Must be one of the following types: float32, float64.
- **`type`** – A Tensor of type int32.
- **`natoms`** – A Tensor of type int32.
- **`box`** – A Tensor. Must have the same type as coord.
- **`mesh`** – A Tensor of type int32.
- **`ef`** – A Tensor. Must have the same type as coord.
- **`davg`** – A Tensor. Must have the same type as coord.
- **`dstd`** – A Tensor. Must have the same type as coord.
- **`rcut_a`** – A float.
- **`rcut_r`** – A float.
- **`rcut_r_smth`** – A float.
- **`sel_a`** – A list of ints.
- **`sel_r`** – A list of ints.
- **`name`** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.`**`DescrptSeAEfVert`**(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **`coord`** – A Tensor. Must be one of the following types: float32, float64.
- **`type`** – A Tensor of type int32.
- **`natoms`** – A Tensor of type int32.
- **`box`** – A Tensor. Must have the same type as coord.
- **`mesh`** – A Tensor of type int32.
- **`ef`** – A Tensor. Must have the same type as coord.
- **`davg`** – A Tensor. Must have the same type as coord.
- **`dstd`** – A Tensor. Must have the same type as coord.

- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.`**`DescrptSeAMask`**(coord, type, mask, box, natoms, mesh, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **mask** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **natoms** – A Tensor of type int32.
- **mesh** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.`**`DescrptSeR`**(coord, type, natoms, box, mesh, davg, dstd, rcut, rcut_smth, sel, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut** – A float.
- **rcut_smth** – A float.

- **sel** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.`**`DotmulFltNvnmd`**`(x, w, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.`**`EwaldRecp`**`(coord, charge, natoms, box, ewald_beta, ewald_h, name=None)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **charge** – A Tensor. Must have the same type as coord.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **ewald_beta** – A float.
- **ewald_h** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (energy, force, virial).

energy: A Tensor. Has the same type as coord. force: A Tensor. Has the same type as coord. virial: A Tensor. Has the same type as coord.

`deepmd.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.
- **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.`**`GeluGradGradCustom`**(dy, dy_, x, name=None)

TODO: add doc.

Parameters

- **`dy`** – A Tensor. Must be one of the following types: float32, float64.
- **`dy`** – A Tensor. Must have the same type as dy.
- **`x`** – A Tensor. Must have the same type as dy.
- **`name`** – A name for the operation (optional).

Returns

A Tensor. Has the same type as dy.

`deepmd.env.op_module.`**`MapAparam`**(aparam, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **`aparam`** – A Tensor. Must be one of the following types: float32, float64.
- **`nlist`** – A Tensor of type int32.
- **`natoms`** – A Tensor of type int32.
- **`n_a_sel`** – An int.
- **`n_r_sel`** – An int.
- **`name`** – A name for the operation (optional).

Returns

A Tensor. Has the same type as aparam.

`deepmd.env.op_module.`**`MapFltNvnmd`**(x, table, table_grad, table_info, name=None)

TODO: add doc.

Parameters

- **`x`** – A Tensor. Must be one of the following types: float32, float64.
- **`table`** – A Tensor. Must have the same type as x.
- **`table_grad`** – A Tensor. Must have the same type as x.
- **`table_info`** – A Tensor. Must have the same type as x.
- **`name`** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.`**`MatmulFitnetNvnmd`**(x, w, nbitx, nbitw, normw, name=None)

TODO: add doc.

Parameters

- **`x`** – A Tensor. Must be one of the following types: float32, float64.
- **`w`** – A Tensor. Must have the same type as x.
- **`nbitx`** – An int.
- **`nbitw`** – An int.
- **`normw`** – An int.

- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

`deepmd.env.op_module.`**`MatmulFlt2fixNvnmd`**(x, w, nbit, name=None)
TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **nbit** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

`deepmd.env.op_module.`**`MatmulFltNvnmd`**(x, w, normx, normw, name=None)
TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **normx** – An int.
- **normw** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

`deepmd.env.op_module.`**`MulFltNvnmd`**(x, w, name=None)
TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

`deepmd.env.op_module.`**`NeighborStat`**(coord, type, natoms, box, mesh, rcut, name=None)
TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.

- **rcut** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (max_nbor_size, min_nbor_dist).

max_nbor_size: A Tensor of type int32. min_nbor_dist: A Tensor. Has the same type as coord.

`deepmd.env.op_module.PairTab`(table_info, table_data, type, rij, nlist, natoms, scale, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **table_info** – A Tensor of type float64.
- **table_data** – A Tensor of type float64.
- **type** – A Tensor of type int32.
- **rij** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **scale** – A Tensor. Must have the same type as rij.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (atom_energy, force, atom_virial).

atom_energy: A Tensor. Has the same type as rij. force: A Tensor. Has the same type as rij. atom_virial: A Tensor. Has the same type as rij.

`deepmd.env.op_module.ParallelProdForceSeA`(net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel, parallel=False, start_frac=0, end_frac=1, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **parallel** – An optional bool. Defaults to False.
- **start_frac** – An optional float. Defaults to 0.
- **end_frac** – An optional float. Defaults to 1.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as net_deriv.

`deepmd.env.op_module.ProdEnvMatA`(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

Compute the environment matrix for descriptor se_e2_a.

Each row of the environment matrix $\mathcal{R}^i$ can be constructed as follows

$$(\mathcal{R}^i)_j = [\begin{array}{c} s(r_{ji}) \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{array}]$$

In the above equation, $\mathrm{R}_{ji} = \mathrm{R}_j - \mathrm{R}_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \|\mathrm{R}_{ji}\|$ is its norm. The switching function $s(r)$ is defined as:

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

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]: 2 <= i < Ntypes+2, number of type i atoms.
- **box** – A Tensor. Must have the same type as coord. The box of frames.
- **mesh** – A Tensor of type int32. Gor historical reasons, only the length of the Tensor matters. If size of mesh == 6, pbc is assumed. If size of mesh == 0, no-pbc is assumed.
- **davg** – A Tensor. Must have the same type as coord. Average value of the environment matrix for normalization.
- **dstd** – A Tensor. Must have the same type as coord. Standard deviation of the environment matrix for normalization.
- **rcut_a** – A float. This argument is not used.
- **rcut_r** – A float. The cutoff radius for the environment matrix.
- **rcut_r_smth** – A float. From where the environment matrix should be smoothed.
- **sel_a** – A list of ints. sel_a[i] specifies the maxmum number of type i atoms in the cut-off radius.
- **sel_r** – A list of ints. This argument is not used.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. The environment matrix. descrpt_deriv: A Tensor. Has the same type as coord. The derivative of the environment matrix. rij: A Tensor. Has the same type as coord. The distance between the atoms. nlist: A Tensor of type int32. The neighbor list of each atom.

`deepmd.env.op_module.`**`ProdEnvMatAMix`**(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

Compute the environment matrix mixing the atom types.

The sorting of neighbor atoms depends not on atom types, but on the distance and index. The atoms in nlist matrix will gather forward and thus save space for gaps of types in ProdEnvMatA, resulting in optimized and relative small sel_a.

The additional outputs are listed as following:

Parameters

- **`coord`** – A Tensor. Must be one of the following types: float32, float64.
- **`type`** – A Tensor of type int32.
- **`natoms`** – A Tensor of type int32.
- **`box`** – A Tensor. Must have the same type as coord.
- **`mesh`** – A Tensor of type int32.
- **`davg`** – A Tensor. Must have the same type as coord.
- **`dstd`** – A Tensor. Must have the same type as coord.
- **`rcut_a`** – A float.
- **`rcut_r`** – A float.
- **`rcut_r_smth`** – A float.
- **`sel_a`** – A list of ints.
- **`sel_r`** – A list of ints.
- **`name`** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist, ntype, nmask).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. ntype: A Tensor of type int32. The corresponding atom types in nlist. nmask: A Tensor of type bool. The atom mask in nlist.

`deepmd.env.op_module.`**`ProdEnvMatANvnmdQuantize`**(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **`coord`** – A Tensor. Must be one of the following types: float32, float64.
- **`type`** – A Tensor of type int32.
- **`natoms`** – A Tensor of type int32.

- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.`**`ProdEnvMatR`**`(`coord, type, natoms, box, mesh, davg, dstd, rcut, rcut_smth, sel, name=None`)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut** – A float.
- **rcut_smth** – A float.
- **sel** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.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.
- **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.`**`ProdVirialSeA`**(net_deriv, in_deriv, rij, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **rij** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

`deepmd.env.op_module.`**`ProdVirialSeR`**(net_deriv, in_deriv, rij, nlist, natoms, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **rij** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.

- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

`deepmd.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.

`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.

`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.

`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, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

`deepmd.env.op_module.`**`TabulateFusionSeR`**(table, table_info, em, last_layer_size, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

`deepmd.env.op_module.`**`TabulateFusionSeRGrad`**(table, table_info, em, dy, descriptor, name=None)

TODO: add doc.

Parameters

- **`table`** – A Tensor. Must be one of the following types: float32, float64.
- **`table_info`** – A Tensor. Must have the same type as table.
- **`em`** – A Tensor. Must have the same type as table.
- **`dy`** – A Tensor. Must have the same type as table.
- **`descriptor`** – A Tensor. Must have the same type as table.
- **`name`** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

`deepmd.env.op_module.`**`TabulateFusionSeRGradGrad`**(table, table_info, em, dz_dy_dem, descriptor, name=None)

TODO: add doc.

Parameters

- **`table`** – A Tensor. Must be one of the following types: float32, float64.
- **`table_info`** – A Tensor. Must have the same type as table.
- **`em`** – A Tensor. Must have the same type as table.
- **`dz_dy_dem`** – A Tensor. Must have the same type as table.
- **`descriptor`** – A Tensor. Must have the same type as table.
- **`name`** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

`deepmd.env.op_module.`**`TabulateFusionSeT`**(table, table_info, em_x, em, last_layer_size, name=None)

TODO: add doc.

Parameters

- **`table`** – A Tensor. Must be one of the following types: float32, float64.
- **`table_info`** – A Tensor. Must have the same type as table.
- **`em_x`** – A Tensor. Must have the same type as table.
- **`em`** – A Tensor. Must have the same type as table.
- **`last_layer_size`** – An int.
- **`name`** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

`deepmd.env.op_module.`**`TabulateFusionSeTGrad`**(table, table_info, em_x, em, dy, descriptor, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table.

`deepmd.env.op_module.`**`TabulateFusionSeTGradGrad`**(table, table_info, em_x, em, dz_dy_dem_x, dz_dy_dem, descriptor, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as table.

`deepmd.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.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.`**`UaggregatedDy2DxS`**(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_nvnmd`(x, w, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.copy_flt_nvnmd`(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.descrpt_norot`(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.descrpt_se_a`(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.`**`descrpt_se_a_ef`**(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.`**`descrpt_se_a_ef_para`**(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.

- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.`**`descrpt_se_a_ef_vert`**(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.`**`descrpt_se_a_mask`**(coord, type, mask, box, natoms, mesh, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.

- **type** – A Tensor of type int32.
- **mask** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **natoms** – A Tensor of type int32.
- **mesh** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.`**`descrpt_se_r`**(coord, type, natoms, box, mesh, davg, dstd, rcut, rcut_smth, sel, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut** – A float.
- **rcut_smth** – A float.
- **sel** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.`**`dotmul_flt_nvnmd`**(x, w, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.`**`ewald_recp`**(coord, charge, natoms, box, ewald_beta, ewald_h, name=None)
TODO: add doc.

Parameters

- **`coord`** – A Tensor. Must be one of the following types: float32, float64.
- **`charge`** – A Tensor. Must have the same type as coord.
- **`natoms`** – A Tensor of type int32.
- **`box`** – A Tensor. Must have the same type as coord.
- **`ewald_beta`** – A float.
- **`ewald_h`** – A float.
- **`name`** – A name for the operation (optional).

Returns

A tuple of Tensor objects (energy, force, virial).

energy: A Tensor. Has the same type as coord. force: A Tensor. Has the same type as coord. virial: A Tensor. Has the same type as coord.

`deepmd.env.op_module.`**`flt_nvnmd`**(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.`**`gelu_custom`**(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_grad`**(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.**gelu_grad_custom**(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.**gelu_grad_grad**(dy, dy_, x, name=None)
TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as dy.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as dy.

deepmd.env.op_module.**gelu_grad_grad_custom**(dy, dy_, x, name=None)
TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as dy.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as dy.

deepmd.env.op_module.**map_aparam**(aparam, nlist, natoms, n_a_sel, n_r_sel, name=None)
TODO: add doc.

Parameters

- **aparam** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.

- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as aparam.

`deepmd.env.op_module.`**`map_flt_nvnmd`**(x, table, table_grad, table_info, name=None)
TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **table** – A Tensor. Must have the same type as x.
- **table_grad** – A Tensor. Must have the same type as x.
- **table_info** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

`deepmd.env.op_module.`**`matmul_fitnet_nvnmd`**(x, w, nbitx, nbitw, normw, name=None)
TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **nbitx** – An int.
- **nbitw** – An int.
- **normw** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

`deepmd.env.op_module.`**`matmul_flt2fix_nvnmd`**(x, w, nbit, name=None)
TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **nbit** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

`deepmd.env.op_module.`**`matmul_flt_nvnmd`**(x, w, normx, normw, name=None)
TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.

- **w** – A Tensor. Must have the same type as x.
- **normx** – An int.
- **normw** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

`deepmd.env.op_module.`**`mul_flt_nvnmd`**(x, w, name=None)

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as x.

`deepmd.env.op_module.`**`neighbor_stat`**(coord, type, natoms, box, mesh, rcut, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **rcut** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (max_nbor_size, min_nbor_dist).

max_nbor_size: A Tensor of type int32. min_nbor_dist: A Tensor. Has the same type as coord.

`deepmd.env.op_module.`**`pair_tab`**(table_info, table_data, type, rij, nlist, natoms, scale, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **table_info** – A Tensor of type float64.
- **table_data** – A Tensor of type float64.
- **type** – A Tensor of type int32.
- **rij** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.

- **scale** – A Tensor. Must have the same type as rij.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (atom_energy, force, atom_virial).

atom_energy: A Tensor. Has the same type as rij. force: A Tensor. Has the same type as rij. atom_virial: A Tensor. Has the same type as rij.

`deepmd.env.op_module.`**`parallel_prod_force_se_a`**`(net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel, parallel=False, start_frac=0, end_frac=1, name=None)`

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **parallel** – An optional bool. Defaults to False.
- **start_frac** – An optional float. Defaults to 0.
- **end_frac** – An optional float. Defaults to 1.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net_deriv.

`deepmd.env.op_module.`**`prod_env_mat_a`**`(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)`

Compute the environment matrix for descriptor se_e2_a.

Each row of the environment matrix $\mathcal{R}^i$ can be constructed as follows

$$(\mathcal{R}^i)_j = [\begin{array}{c} s(r_{ji}) \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{array}]$$

In the above equation, $\mathrm{R}_{ji} = \mathrm{R}_j - \mathrm{R}_i = (x_{ji}, y_{ji}, z_{ji})$ is the relative coordinate and $r_{ji} = \|\mathrm{R}_{ji}\|$ is its norm. The switching function $s(r)$ is defined as:

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

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]: 2 <= i < Ntypes+2, number of type i atoms.
- **box** – A Tensor. Must have the same type as coord. The box of frames.
- **mesh** – A Tensor of type int32. Gor historical reasons, only the length of the Tensor matters. If size of mesh == 6, pbc is assumed. If size of mesh == 0, no-pbc is assumed.
- **davg** – A Tensor. Must have the same type as coord. Average value of the environment matrix for normalization.
- **dstd** – A Tensor. Must have the same type as coord. Standard deviation of the environment matrix for normalization.
- **rcut_a** – A float. This argument is not used.
- **rcut_r** – A float. The cutoff radius for the environment matrix.
- **rcut_r_smth** – A float. From where the environment matrix should be smoothed.
- **sel_a** – A list of ints. sel_a[i] specifies the maxmum number of type i atoms in the cut-off radius.
- **sel_r** – A list of ints. This argument is not used.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. The environment matrix. descrpt_deriv: A Tensor. Has the same type as coord. The derivative of the environment matrix. rij: A Tensor. Has the same type as coord. The distance between the atoms. nlist: A Tensor of type int32. The neighbor list of each atom.

`deepmd.env.op_module.`**`prod_env_mat_a_mix`**(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

Compute the environment matrix mixing the atom types.

The sorting of neighbor atoms depends not on atom types, but on the distance and index. The atoms in nlist matrix will gather forward and thus save space for gaps of types in ProdEnvMatA, resulting in optimized and relative small sel_a.

The additional outputs are listed as following:

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist, ntype, nmask).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. ntype: A Tensor of type int32. The corresponding atom types in nlist. nmask: A Tensor of type bool. The atom mask in nlist.

`deepmd.env.op_module.`**`prod_env_mat_a_nvnmd_quantize`**(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.`**`prod_env_mat_r`**(coord, type, natoms, box, mesh, davg, dstd, rcut, rcut_smth, sel, name=None)

TODO: add doc.

Parameters

- **`coord`** – A Tensor. Must be one of the following types: float32, float64.
- **`type`** – A Tensor of type int32.
- **`natoms`** – A Tensor of type int32.
- **`box`** – A Tensor. Must have the same type as coord.
- **`mesh`** – A Tensor of type int32.
- **`davg`** – A Tensor. Must have the same type as coord.
- **`dstd`** – A Tensor. Must have the same type as coord.
- **`rcut`** – A float.
- **`rcut_smth`** – A float.
- **`sel`** – A list of ints.
- **`name`** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.`**`prod_force`**(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.`**`prod_force_norot`**(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.`**`prod_force_se_a`**(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.`**`prod_force_se_a_mask`**(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.`**`prod_force_se_r`**(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.`**`prod_virial`**(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.`**`prod_virial_norot`**(net_deriv, in_deriv, rij, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **`net_deriv`** – A Tensor. Must be one of the following types: float32, float64.
- **`in_deriv`** – A Tensor. Must have the same type as net_deriv.
- **`rij`** – A Tensor. Must have the same type as net_deriv.
- **`nlist`** – A Tensor of type int32.
- **`natoms`** – A Tensor of type int32.
- **`n_a_sel`** – An int.
- **`n_r_sel`** – An int.

- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

`deepmd.env.op_module.`**`prod_virial_se_a`**(net_deriv, in_deriv, rij, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **rij** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

`deepmd.env.op_module.`**`prod_virial_se_r`**(net_deriv, in_deriv, rij, nlist, natoms, name=None)

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **rij** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

`deepmd.env.op_module.`**`quantize_nvnmd`**(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.soft_min_force`(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.soft_min_switch`(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.`**`soft_min_virial`**`(`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.`**`tabulate_fusion`**`(`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.`**`tabulate_fusion_grad`**`(`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.`**`tabulate_fusion_grad_grad`**(table, table_info, em_x, em, dz_dy_dem_x, dz_dy_dem, descriptor, name=None)

TODO: add doc.

Parameters

- **`table`** – A Tensor. Must be one of the following types: float32, float64.
- **`table_info`** – A Tensor. Must have the same type as table.
- **`em_x`** – A Tensor. Must have the same type as table.
- **`em`** – A Tensor. Must have the same type as table.
- **`dz_dy_dem_x`** – A Tensor. Must have the same type as table.
- **`dz_dy_dem`** – A Tensor. Must have the same type as table.
- **`descriptor`** – A Tensor. Must have the same type as table.
- **`name`** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

`deepmd.env.op_module.`**`tabulate_fusion_se_a`**(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.`**`tabulate_fusion_se_a_grad`**(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.`**`tabulate_fusion_se_a_grad_grad`**(table, table_info, em_x, em, dz_dy_dem_x, dz_dy_dem, descriptor, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

`deepmd.env.op_module.`**`tabulate_fusion_se_r`**(table, table_info, em, last_layer_size, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

`deepmd.env.op_module.`**`tabulate_fusion_se_r_grad`**(table, table_info, em, dy, descriptor, name=None)

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.

- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as table.

`deepmd.env.op_module.`**`tabulate_fusion_se_r_grad_grad`**(table, table_info, em, dz_dy_dem, descriptor, name=None)

TODO: add doc.

Parameters
- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as table.

`deepmd.env.op_module.`**`tabulate_fusion_se_t`**(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.`**`tabulate_fusion_se_t_grad`**(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.`**`tabulate_fusion_se_t_grad_grad`**`(table, table_info, em_x, em, dz_dy_dem_x, dz_dy_dem, descriptor, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

`deepmd.env.op_module.`**`tanh4_flt_nvnmd`**`(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.`**`unaggregated_dy2_dx`**`(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.`**`unaggregated_dy2_dx_s`**(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.`**`unaggregated_dy_dx`**(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.`**`unaggregated_dy_dx_s`**(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.

18.2 op_grads_module

Python wrappers around TensorFlow ops.

This file is MACHINE GENERATED! Do not edit.

`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.
- **`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.`**`ProdForceSeAGrad`**(grad, net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **`grad`** – A Tensor. Must be one of the following types: float32, float64.
- **`net_deriv`** – A Tensor. Must have the same type as grad.
- **`in_deriv`** – A Tensor. Must have the same type as grad.
- **`nlist`** – A Tensor of type int32.
- **`natoms`** – A Tensor of type int32.
- **`n_a_sel`** – An int.
- **`n_r_sel`** – An int.
- **`name`** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.`**`ProdForceSeAMaskGrad`**(grad, net_deriv, in_deriv, mask, nlist, total_atom_num, name=None)

TODO: add doc.

Parameters

- **`grad`** – A Tensor. Must be one of the following types: float32, float64.

- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **mask** – A Tensor of type int32.
- **nlist** – A Tensor of type int32.
- **total_atom_num** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.`**`ProdForceSeRGrad`**(grad, net_deriv, in_deriv, nlist, natoms, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.`**`ProdVirialGrad`**(grad, net_deriv, in_deriv, rij, nlist, axis, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

`deepmd.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.

`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.

`deepmd.env.op_grads_module.`**`prod_force_grad`**`(grad, net_deriv, in_deriv, nlist, axis, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **`grad`** – A Tensor. Must be one of the following types: float32, float64.
- **`net_deriv`** – A Tensor. Must have the same type as grad.
- **`in_deriv`** – A Tensor. Must have the same type as grad.
- **`nlist`** – A Tensor of type int32.
- **`axis`** – A Tensor of type int32.
- **`natoms`** – A Tensor of type int32.
- **`n_a_sel`** – An int.
- **`n_r_sel`** – An int.
- **`name`** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.`**`prod_force_se_a_grad`**`(grad, net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **`grad`** – A Tensor. Must be one of the following types: float32, float64.
- **`net_deriv`** – A Tensor. Must have the same type as grad.

- **in_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.`**`prod_force_se_a_mask_grad`**(grad, net_deriv, in_deriv, mask, nlist, total_atom_num, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **mask** – A Tensor of type int32.
- **nlist** – A Tensor of type int32.
- **total_atom_num** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.`**`prod_force_se_r_grad`**(grad, net_deriv, in_deriv, nlist, natoms, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.`**`prod_virial_grad`**(grad, net_deriv, in_deriv, rij, nlist, axis, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.

- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.`**`prod_virial_se_a_grad`**(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.`**`prod_virial_se_r_grad`**(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.soft_min_force_grad`(grad, du, sw_deriv, nlist, natoms, n_a_sel, n_r_sel, name=None)

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **du** – A Tensor. Must have the same type as grad.
- **sw_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns
A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.soft_min_virial_grad`(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.

CHAPTER

NINETEEN

C++ API

19.1 Class Hierarchy

- Namespace deepmd
    - Struct deepmd_exception
    - Struct NeighborListData
    - Struct tf_exception
    - Class AtomMap
    - Class DeepPot
    - Class DeepPotModelDevi
    - Class DeepTensor
    - Class DipoleChargeModifier

19.2 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
            - file_source_api_cc_include_DeepPot.h
            - file_source_api_cc_include_DeepTensor.h
            - file_source_api_cc_include_tf_private.h
            - file_source_api_cc_include_tf_public.h

19.3 Full API

19.3.1 Namespaces

Namespace deepmd

Contents

Classes

- Struct deepmd_exception
- Struct NeighborListData
- Struct tf_exception
- Class AtomMap
- Class DeepPot
- Class DeepPotModelDevi
- Class DeepTensor
- Class DipoleChargeModifier

Functions

- Function deepmd::check_status
- Function deepmd::convert_pbtxt_to_pb
- Function deepmd::get_env_nthreads
- Function deepmd::load_op_library
- Function deepmd::model_compatable
- Function deepmd::name_prefix
- Function deepmd::print_summary
- Function deepmd::read_file_to_string
- Template Function deepmd::select_by_type
- Template Function deepmd::select_map(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<VALUETYPE>&, const int&, const std::vector<int>&, const std::vector<VALUETYPE>&, const double&, const std::vector<VALUETYPE>&, const std::vector<VALUETYPE>&, const deepmd::AtomMap&, const std::string)
- Template Function deepmd::session_input_tensors(std::vector<std::pair<std::string, tensorflow::Tensor>>&, const std::vector<VALUETYPE>&, const int&, const std::vector<int>&, const std::vector<VALUETYPE>&, InputNlist&, const std::vector<VALUETYPE>&, const std::vector<VALUETYPE>&, const deepmd::AtomMap&, const int, const int, const std::string)
- Template Function deepmd::session_input_tensors_mixed_type

Typedefs

- Typedef deepmd::ENERGYTYPE
- Typedef deepmd::STRINGTYPE

Namespace tensorflow

19.3.2 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 **`VALUETYPE`**>
void **`forward`**(typename std::vector<VALUETYPE>::iterator out, const typename std::vector<VALUETYPE>::const_iterator in, const int stride = 1, const int nframes = 1, const int nall = 0) const

template<typename **`VALUETYPE`**>
void **`backward`**(typename std::vector<VALUETYPE>::iterator out, const typename std::vector<VALUETYPE>::const_iterator in, const int stride = 1, const int nframes = 1, const int nall = 0) const

inline const std::vector<int> &**get_type**() const

inline const std::vector<int> &**get_fwd_map**() const

inline const std::vector<int> &**get_bkw_map**() const

Class DeepPot

- Defined in file_source_api_cc_include_DeepPot.h

Class Documentation

class **DeepPot**

Deep Potential.

Public Functions

DeepPot()

DP constructor without initialization.

~DeepPot()

DeepPot(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")

DP constructor with initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **file_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")

Initialize the DP.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **file_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

void **print_summary**(const std::string &pre) const

Print the DP summary to the screen.

Parameters

pre – [in] The prefix to each line.

template<typename **VALUETYPE**, typename **ENERGYVTYPE**>

void **compute**(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**, typename **ENERGYVTYPE**>
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 &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**, typename **ENERGYVTYPE**>
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.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**, typename **ENERGYVTYPE**>
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.

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**, typename **ENERGYVTYPE**>
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 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**, typename **ENERGYVTYPE**>

void **compute_mixed_type**(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy, std::vector<VALUETYPE> &atom_virial, const int &nframes, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, 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.

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.

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 DeepPotModelDevi

- Defined in file_source_api_cc_include_DeepPot.h

Class Documentation

class **DeepPotModelDevi**

Public Functions

DeepPotModelDevi()

DP model deviation constructor without initialization.

~DeepPotModelDevi()

DeepPotModelDevi(const std::vector<std::string> &models, const int &gpu_rank = 0, const std::vector<std::string> &file_contents = std::vector<std::string>())

DP model deviation constructor with initialization.

Parameters

- **models** – [in] The names of the frozen model files.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **file_contents** – [in] The contents of the model files. If it is not empty, DP will read from the strings instead of the files.

void **init**(const std::vector<std::string> &models, const int &gpu_rank = 0, const std::vector<std::string> &file_contents = std::vector<std::string>())

Initialize the DP model deviation contrcutor.

Parameters

- **models** – [in] The names of the frozen model files.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **file_contents** – [in] The contents of the model files. If it is not empty, DP will read from the strings instead of the files.

template<typename **VALUETYPE**>

void **compute**(std::vector<ENERGYTYPE> &all_ener, std::vector<std::vector<VALUETYPE>> &all_force, std::vector<std::vector<VALUETYPE>> &all_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using these DP models.

Parameters

- **all_ener** – [out] The system energies of all models.
- **all_force** – [out] The forces on each atom of all models.
- **all_virial** – [out] The virials of all models.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **lmp_list** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. dim_aparam. Then all frames and atoms are provided with the same aparam.

template<typename **VALUETYPE**>
void **compute**(std::vector<ENERGYTYPE> &all_ener, std::vector<std::vector<VALUETYPE>> &all_force, std::vector<std::vector<VALUETYPE>> &all_virial, std::vector<std::vector<VALUETYPE>> &all_atom_energy, std::vector<std::vector<VALUETYPE>> &all_atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using these DP models.

Parameters

- **all_ener** – [out] The system energies of all models.
- **all_force** – [out] The forces on each atom of all models.
- **all_virial** – [out] The virials of all models.
- **all_atom_energy** – [out] The atomic energies of all models.
- **all_atom_virial** – [out] The atomic virials of all models.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.

- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **lmp_list** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam. dim_aparam. Then all frames and atoms are provided with the same aparam.

inline double **cutoff()** const

Get the cutoff radius.

Returns
The cutoff radius.

inline int **numb_types()** const

Get the number of types.

Returns
The number of types.

inline int **numb_types_spin()** const

Get the number of types with spin.

Returns
The number of types with spin.

inline int **dim_fparam()** const

Get the dimension of the frame parameter.

Returns
The dimension of the frame parameter.

inline int **dim_aparam()** const

Get the dimension of the atomic parameter.

Returns
The dimension of the atomic parameter.

template<typename **VALUETYPE**>
void **compute_avg(**VALUETYPE &dener, const std::vector<VALUETYPE> &all_energy**)**

Compute the average energy.

Parameters

- **dener** – [out] The average energy.
- **all_energy** – [in] The energies of all models.

template<typename **VALUETYPE**>

void **compute_avg**(std::vector<VALUETYPE> &avg, const std::vector<std::vector<VALUETYPE>> &xx)

Compute the average of vectors.

Parameters

- **avg** – [out] The average of vectors.
- **xx** – [in] The vectors of all models.

template<typename **VALUETYPE**>
void **compute_std**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const std::vector<std::vector<VALUETYPE>> &xx, const int &stride)

Compute the standard deviation of vectors.

Parameters

- **std** – [out] The standard deviation of vectors.
- **avg** – [in] The average of vectors.
- **xx** – [in] The vectors of all models.
- **stride** – [in] The stride to compute the deviation.

template<typename **VALUETYPE**>
void **compute_relative_std**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const VALUETYPE eps, const int &stride)

Compute the relative standard deviation of vectors.

Parameters

- **std** – [out] The standard deviation of vectors.
- **avg** – [in] The average of vectors.
- **eps** – [in] The level parameter for computing the deviation.
- **stride** – [in] The stride to compute the deviation.

template<typename **VALUETYPE**>
void **compute_std_e**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const std::vector<std::vector<VALUETYPE>> &xx)

Compute the standard deviation of atomic energies.

Parameters

- **std** – [out] The standard deviation of atomic energies.
- **avg** – [in] The average of atomic energies.
- **xx** – [in] The vectors of all atomic energies.

template<typename **VALUETYPE**>
void **compute_std_f**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const std::vector<std::vector<VALUETYPE>> &xx)

Compute the standard deviation of forces.

Parameters

- **std** – [out] The standard deviation of forces.
- **avg** – [in] The average of forces.
- **xx** – [in] The vectors of all forces.

template<typename **VALUETYPE**>
void **compute_relative_std_f**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const VALUETYPE eps)

Compute the relative standard deviation of forces.

Parameters

- **std** – [out] The relative standard deviation of forces.
- **avg** – [in] The relative average of forces.
- **eps** – [in] The level parameter for computing the deviation.

Class DeepTensor

- Defined in file_source_api_cc_include_DeepTensor.h

Class Documentation

class **DeepTensor**

Deep Tensor.

Public Functions

DeepTensor()

Deep Tensor constructor without initialization.

~DeepTensor()

DeepTensor(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Deep Tensor constructor with initialization..

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] Name scopes of operations.

void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Initialize the Deep Tensor.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] Name scopes of operations.

void **print_summary**(const std::string &pre) const

Print the DP summary to the screen.

Parameters

pre – [in] The prefix to each line.

template<typename **VALUETYPE**>

void **compute**(std::vector<VALUETYPE> &value, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box)

Evaluate the value by using this model.

Parameters

- **value** – [out] The value to evalute, usually would be the atomic tensor.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.

template<typename **VALUETYPE**>
void **compute**(std::vector<VALUETYPE> &value, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &inlist)

Evaluate the value by using this model.

Parameters

- **value** – [out] The value to evalute, usually would be the atomic tensor.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.

template<typename **VALUETYPE**>
void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box)

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global_tensor** – [out] The global tensor to evalute.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.

template<typename **VALUETYPE**>
void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &inlist)

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global_tensor** – [out] The global tensor to evalute.

- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.

template<typename **VALUETYPE**>
void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_tensor, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box)

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global_tensor** – [out] The global tensor to evalute.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **atom_tensor** – [out] The atomic tensor value of the model, size natoms x odim.
- **atom_virial** – [out] The component-wise atomic virial of the global tensor, size odim x natoms x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.

template<typename **VALUETYPE**>
void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_tensor, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &inlist)

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global_tensor** – [out] The global tensor to evalute.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **atom_tensor** – [out] The atomic tensor value of the model, size natoms x odim.
- **atom_virial** – [out] The component-wise atomic virial of the global tensor, size odim x natoms x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.

- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.

inline double **cutoff()** const

Get the cutoff radius.

Returns

The cutoff radius.

inline int **numb_types()** const

Get the number of types.

Returns

The number of types.

inline int **output_dim()** const

Get the output dimension.

Returns

The output dimension.

inline const std::vector<int> &**sel_types()** const

Get the list of sel types.

Returns

The list of sel types.

Class DipoleChargeModifier

- Defined in file_source_api_cc_include_DataModifier.h

Class Documentation

class **DipoleChargeModifier**

Dipole charge modifier.

Public Functions

DipoleChargeModifier()

Dipole charge modifier without initialization.

DipoleChargeModifier(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = ””)

Dipole charge modifier without initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] The name scope.

~DipoleChargeModifier()

void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Initialize the dipole charge modifier.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The GPU rank. Default is 0.
- **name_scope** – [in] The name scope.

void **print_summary**(const std::string &pre) const

Print the DP summary to the screen.

Parameters

pre – [in] The prefix to each line.

template<typename **VALUETYPE**>
void **compute**(std::vector<VALUETYPE> &dfcorr_, std::vector<VALUETYPE> &dvcorr_, const std::vector<VALUETYPE> &dcoord_, const std::vector<int> &datype_, const std::vector<VALUETYPE> &dbox, const std::vector<std::pair<int, int>> &pairs, const std::vector<VALUETYPE> &delef_, const int nghost, const InputNlist &lmp_list)

Evaluate the force and virial correction by using this dipole charge modifier.

Parameters

- **dfcorr_** – [out] The force correction on each atom.
- **dvcorr_** – [out] The virial correction.
- **dcoord_** – [in] The coordinates of atoms. The array should be of size 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.

inline double **cutoff**() const

Get cutoff radius.

Returns

double cutoff radius.

inline int **numb_types**() const

Get the number of atom types.

Returns

int number of atom types.

inline std::vector<int> **sel_types**() const

Get the list of sel types.

Returns
The list of sel types.

19.3.3 Functions

Function deepmd::check_status

- Defined in file_source_api_cc_include_common.h

Function Documentation

void `deepmd::check_status`(const tensorflow::Status &status)
Check TensorFlow status. Exit if not OK.

Parameters
`status` – [in] TensorFlow status.

Function deepmd::convert_pbtxt_to_pb

- Defined in file_source_api_cc_include_common.h

Function Documentation

void `deepmd::convert_pbtxt_to_pb`(std::string fn_pb_txt, std::string fn_pb)
Convert pbtxt to pb.

Parameters

- **`fn_pb_txt`** – [in] Filename of the pb txt file.
- **`fn_pb`** – [in] Filename of the pb file.

Function deepmd::get_env_nthreads

- Defined in file_source_api_cc_include_common.h

Function Documentation

void `deepmd::get_env_nthreads`(int &num_intra_nthreads, int &num_inter_nthreads)
Get the number of threads from the environment variable.

A warning will be thrown if environmental variables are not set.

Parameters

- **`num_intra_nthreads`** – [out] The number of intra threads. Read from 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

void `deepmd::load_op_library()`

Dynamically load OP library. This should be called before loading graphs.

Function deepmd::model_compatable

- Defined in file_source_api_cc_include_common.h

Function Documentation

bool `deepmd::model_compatable`(std::string &model_version)

Check if the model version is supported.

Parameters
: `model_version` – [in] The model version.

Returns
: Whether the model is supported (true or false).

Function deepmd::name_prefix

- Defined in file_source_api_cc_include_common.h

Function Documentation

std::string `deepmd::name_prefix`(const std::string &name_scope)

Function deepmd::print_summary

- Defined in file_source_api_cc_include_common.h

Function Documentation

void `deepmd::print_summary`(const std::string &pre)

Print the summary of DeePMD-kit, including the version and the build information.

Parameters
: `pre` – [in] The prefix to each line.

Function deepmd::read_file_to_string

- Defined in file_source_api_cc_include_common.h

Function Documentation

void **deepmd::read_file_to_string**(std::string model, std::string &file_content)

Read model file to a string.

Parameters

- **model** – [in] Path to the model.
- **file_content** – [out] Content of the model file.

Template Function deepmd::select_by_type

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **VALUETYPE**>
void **deepmd::select_by_type**(std::vector<int> &fwd_map, std::vector<int> &bkw_map, int &nghost_real, const std::vector<VALUETYPE> &dcoord_, const std::vector<int> &datype_, const int &nghost, const std::vector<int> &sel_type_)

Get forward and backward map of selected atoms by atom types.

Parameters

- **fwd_map** – [out] The forward map with size natoms.
- **bkw_map** – [out] The backward map with size nreal.
- **nghost_real** – [out] The number of selected ghost atoms.
- **dcoord_** – [in] The coordinates of all atoms. Reserved for compatibility.
- **datype_** – [in] The atom types of all atoms.
- **nghost** – [in] The number of ghost atoms.
- **sel_type_** – [in] The selected atom types.

Template Function deepmd::select_map(std::vector<VT>&, const std::vector<VT>&, const std::vector<int>&, const int&, const int&, const int&, const int&)

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **VT**>
void **deepmd::select_map**(std::vector<VT> &out, const std::vector<VT> &in, const std::vector<int> &fwd_map, const int &stride, const int &nframes = 1, const int &nall1 = 0, const int &nall2 = 0)

Apply the given map to a vector.

Parameters

- **out** – [out] The output vector.
- **in** – [in] The input vector.
- **fwd_map** – [in] The map.
- **stride** – [in] The stride of the input vector.
- **nframes** – [in] The number of frames.
- **nall1** – [in] The number of atoms in the input vector.
- **nall2** – [in] The number of atoms in the output vector.

Template Function deepmd::select_map(typename std::vector<VT>::iterator, const typename std::vector<VT>::const_iterator, const std::vector<int>&, const int&, const int&, const int&, const int&)

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **VT**>
void **deepmd::select_map**(typename std::vector<VT>::iterator out, const typename std::vector<VT>::const_iterator in, const std::vector<int> &fwd_map, const int &stride, const int &nframes = 1, const int &nall1 = 0, const int &nall2 = 0)

Apply the given map to a vector.

Parameters

- **out** – [out] The output vector.
- **in** – [in] The input vector.
- **fwd_map** – [in] The map.
- **stride** – [in] The stride of the input vector.
- **nframes** – [in] The number of frames.
- **nall1** – [in] The number of atoms in the input vector.
- **nall2** – [in] The number of atoms in the output vector.

Template Function deepmd::select_map_inv(std::vector<VT>&, const std::vector<VT>&, const std::vector<int>&, const int&)

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **VT**>
void **deepmd::select_map_inv**(std::vector<VT> &out, const std::vector<VT> &in, const std::vector<int> &fwd_map, const int &stride)

Template Function deepmd::select_map_inv(typename std::vector<VT>::iterator, const typename std::vector<VT>::const_iterator, const std::vector<int>&, const int&)

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **VT**>
void **deepmd::select_map_inv**(typename std::vector<VT>::iterator out, const typename std::vector<VT>::const_iterator in, const std::vector<int> &fwd_map, const int &stride)

Template Function deepmd::select_real_atoms

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **VALUETYPE**>
void **deepmd::select_real_atoms**(std::vector<int> &fwd_map, std::vector<int> &bkw_map, int &nghost_real, const std::vector<VALUETYPE> &dcoord_, const std::vector<int> &datype_, const int &nghost, const int &ntypes)

Template Function deepmd::select_real_atoms_coord

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **VALUETYPE**>
void **deepmd::select_real_atoms_coord**(std::vector<VALUETYPE> &dcoord, std::vector<int> &datype, std::vector<VALUETYPE> &aparam, int &nghost_real, std::vector<int> &fwd_map, std::vector<int> &bkw_map, int &nall_real, int &nloc_real, const std::vector<VALUETYPE> &dcoord_, const std::vector<int> &datype_, const std::vector<VALUETYPE> &aparam_, const int &nghost, const int &ntypes, const int &nframes, const int &daparam, const int &nall)

Function deepmd::session_get_dtype

- Defined in file_source_api_cc_include_common.h

Function Documentation

int **deepmd::session_get_dtype**(tensorflow::Session *session, const std::string name, const std::string scope = "")

Get the type of a tensor.

Parameters

- **session** – [in] TensorFlow session.
- **name** – [in] The name of the tensor.
- **scope** – [in] The scope of the tensor.

Returns
The type of the tensor as int.

Template Function deepmd::session_get_scalar

- Defined in file_source_api_cc_include_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

- **session** – [in] TensorFlow session.
- **name** – [in] The name of the tensor.
- **scope** – [in] The scope of the tensor.

Returns
The value of the tensor.

Template Function deepmd::session_get_vector

- Defined in file_source_api_cc_include_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.
- **session** – [in] TensorFlow session.
- **name** – [in] The name of the tensor.
- **scope** – [in] The scope of the tensor.

Template Function deepmd::session_input_tensors(std::vector<std::pair<std::string, tensorflow::Tensor»&, const std::vector<VALUETYPE>&, const int&, const std::vector<int>&, const std::vector<VALUETYPE>&, const double&, const std::vector<VALUETYPE>&, const std::vector<VALUETYPE>&, const deepmd::AtomMap&, const std::string)

- 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> &datype_, const std::vector<VALUETYPE> &dbox, const double &cell_size, const std::vector<VALUETYPE> &fparam_, const std::vector<VALUETYPE> &aparam_, const deepmd::AtomMap &atommap, const std::string scope = "")

Get input tensors.

Parameters

- **input_tensors** – [out] Input tensors.
- **dcoord_** – [in] Coordinates of atoms.
- **ntypes** – [in] Number of atom types.
- **datype_** – [in] Atom types.
- **dbox** – [in] Box matrix.
- **cell_size** – [in] Cell size.
- **fparam_** – [in] Frame parameters.
- **aparam_** – [in] Atom parameters.

- **atommap** – [in] Atom map.
- **scope** – [in] The scope of the tensors.

Template Function deepmd::session_input_tensors(std::vector<std::pair<std::string, tensorflow::Tensor»&, const std::vector<VALUETYPE>&, const int&, const std::vector<int>&, const std::vector<VALUETYPE>&, InputNlist&, const std::vector<VALUETYPE>&, const std::vector<VALUETYPE>&, const deepmd::AtomMap&, const int, const int, const std::string)

- 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> &datype_, const std::vector<VALUETYPE> &dbox, InputNlist &dlist, const std::vector<VALUETYPE> &fparam_, const std::vector<VALUETYPE> &aparam_, const deepmd::AtomMap &atommap, const int nghost, const int ago, const std::string scope = "")

Get input tensors.

Parameters

- **input_tensors** – [out] Input tensors.
- **dcoord_** – [in] Coordinates of atoms.
- **ntypes** – [in] Number of atom types.
- **datype_** – [in] Atom types.
- **dlist** – [in] Neighbor list.
- **fparam_** – [in] Frame parameters.
- **aparam_** – [in] Atom parameters.
- **atommap** – [in] Atom map.
- **nghost** – [in] Number of ghost atoms.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **scope** – [in] The scope of the tensors.

Template Function deepmd::session_input_tensors_mixed_type

- Defined in file_source_api_cc_include_common.h

Function Documentation

template<typename **MODELTYPE**, typename **VALUETYPE**>
int **deepmd::session_input_tensors_mixed_type**(std::vector<std::pair<std::string, tensorflow::Tensor>> &input_tensors, const int &nframes, const std::vector<VALUETYPE> &dcoord_, const int &ntypes, const std::vector<int> &datype_, const std::vector<VALUETYPE> &dbox, const double &cell_size, const std::vector<VALUETYPE> &fparam_, const std::vector<VALUETYPE> &aparam_, const deepmd::AtomMap &atommap, const std::string scope = "")

Get input tensors for mixed type.

Parameters

- **input_tensors** – [out] Input tensors.
- **nframes** – [in] Number of frames.
- **dcoord_** – [in] Coordinates of atoms.
- **ntypes** – [in] Number of atom types.
- **datype_** – [in] Atom types.
- **dlist** – [in] Neighbor list.
- **fparam_** – [in] Frame parameters.
- **aparam_** – [in] Atom parameters.
- **atommap** – [in] Atom map.
- **nghost** – [in] Number of ghost atoms.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **scope** – [in] The scope of the tensors.

19.3.4 Typedefs

Typedef deepmd::ENERGYTYPE

- Defined in file_source_api_cc_include_common.h

Typedef Documentation

typedef double `deepmd::ENERGYTYPE`

Typedef deepmd::STRINGTYPE

- Defined in file_source_api_cc_include_tf_private.h

Typedef Documentation

typedef std::string `deepmd::STRINGTYPE`

CHAPTER

TWENTY

C API

20.1 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

20.2 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

20.3 Full API

20.3.1 Namespaces

Namespace deepmd

Namespaces

- Namespace deepmd::hpp

Namespace deepmd::hpp

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

20.3.2 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**

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`**

Struct DP_DeepTensor

- Defined in file_source_api_c_include_c_api_internal.h

Struct Documentation

struct **`DP_DeepTensor`**

Public Functions

`DP_DeepTensor`()

`DP_DeepTensor`(deepmd::DeepTensor &dt)

Public Members

deepmd::DeepTensor **`dt`**

std::string **`exception`**

Struct DP_DipoleChargeModifier

- Defined in file_source_api_c_include_c_api_internal.h

Struct Documentation

struct **`DP_DipoleChargeModifier`**

Public Functions

`DP_DipoleChargeModifier``()`

`DP_DipoleChargeModifier``(`deepmd::DipoleChargeModifier &dcm`)`

Public Members

deepmd::DipoleChargeModifier **`dcm`**

std::string **`exception`**

Struct DP_Nlist

- Defined in file_source_api_c_include_c_api_internal.h

Struct Documentation

struct **`DP_Nlist`**

Public Functions

`DP_Nlist()`

`DP_Nlist`(deepmd::InputNlist &nl)

Public Members

deepmd::InputNlist **`nl`**

std::string **`exception`**

Class DeepPot

- Defined in file_source_api_c_include_deepmd.hpp

Class Documentation

class **`DeepPot`**

Deep Potential.

Public Functions

inline **`DeepPot()`**

DP constructor without initialization.

inline **`~DeepPot()`**

inline **`DeepPot`**(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")

DP constructor with initialization.

Parameters

- **`model`** – [in] The name of the frozen model file.
- **`gpu_rank`** – [in] The GPU rank.
- **`file_content`** – [in] The content of the frozen model file.

inline void **`init`**(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")

Initialize the DP.

Parameters

- **`model`** – [in] The name of the frozen model file.
- **`gpu_rank`** – [in] The GPU rank.
- **`file_content`** – [in] The content of the frozen model file.

template<typename **`VALUETYPE`**, typename **`ENERGYVTYPE`**>

inline void **compute**(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**, typename **ENERGYVTYPE**>
inline void **compute**(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).

- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**, typename **ENERGYVTYPE**>
inline void **compute**(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using this DP with the neighbor list.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**, typename **ENERGYVTYPE**>
inline void **compute**(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP with the neighbor list.

Parameters

- **ener** – [out] The system energy.

- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**, typename **ENERGYVTYPE**>
inline void **compute_mixed_type**(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const int &nframes, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using this DP with the mixed type.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

template<typename **VALUETYPE**, typename **ENERGYVTYPE**>
inline void **compute_mixed_type**(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy, std::vector<VALUETYPE> &atom_virial, const int &nframes, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP with the mixed type.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim_fparam. dim_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim_aparam. natoms x dim_aparam. Then all frames are assumed to be provided with the same aparam.

inline double **cutoff**() const

Get the cutoff radius.

Returns
The cutoff radius.

inline int **numb_types**() const

Get the number of types.

Returns
The number of types.

inline int **numb_types_spin**() const

Get the number of types with spin.

Returns
The number of types with spin.

inline void **`get_type_map`**(std::string &type_map)

Get the type map (element name of the atom types) of this model.

Parameters
: **`type_map`** – [out] The type map of this model.

inline void **`print_summary`**(const std::string &pre) const

Print the summary of DeePMD-kit, including the version and the build information.

Parameters
: **`pre`** – [in] The prefix to each line.

inline int **`dim_fparam`**() const

Get the dimension of the frame parameter.

Returns
: The dimension of the frame parameter.

inline int **`dim_aparam`**() const

Get the dimension of the atomic parameter.

Returns
: The dimension of the atomic parameter.

Class DeepPotModelDevi

- Defined in file_source_api_c_include_deepmd.hpp

Class Documentation

class **`DeepPotModelDevi`**

Deep Potential model deviation.

Public Functions

inline **`DeepPotModelDevi`**()

DP model deviation constructor without initialization.

inline **`~DeepPotModelDevi`**()

inline **`DeepPotModelDevi`**(const std::vector<std::string> &models)

DP model deviation constructor with initialization.

Parameters
: **`models`** – [in] The names of the frozen model file.

inline void **`init`**(const std::vector<std::string> &models, const int &gpu_rank = 0, const std::vector<std::string> &file_content = std::vector<std::string>())

Initialize the DP model deviation.

Parameters

- **`model`** – [in] The name of the frozen model file.
- **`gpu_rank`** – [in] The GPU rank.

- **file_content** – [in] The content of the frozen model file.

template<typename **VALUETYPE**>
inline void **compute**(std::vector<double> &ener, std::vector<std::vector<VALUETYPE>> &force, std::vector<std::vector<VALUETYPE>> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const 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<typename **VALUETYPE**>
inline void **compute**(std::vector<double> &ener, std::vector<std::vector<VALUETYPE>> &force, std::vector<std::vector<VALUETYPE>> &virial, std::vector<std::vector<VALUETYPE>> &atom_energy, std::vector<std::vector<VALUETYPE>> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list, const int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP model deviation.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_energy** – [out] The atomic energy.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).

inline double **`cutoff()`** const

Get the cutoff radius.

Returns
The cutoff radius.

inline int **`numb_types()`** const

Get the number of types.

Returns
The number of types.

inline int **`numb_types_spin()`** const

Get the number of types with spin.

Returns
The number of types with spin.

inline int **`dim_fparam()`** const

Get the dimension of the frame parameter.

Returns
The dimension of the frame parameter.

inline int **`dim_aparam()`** const

Get the dimension of the atomic parameter.

Returns
The dimension of the atomic parameter.

template<typename **`VALUETYPE`**>
inline void **`compute_avg`**(std::vector<VALUETYPE> &avg, const std::vector<std::vector<VALUETYPE>> &xx)

Compute the average of vectors.

Parameters

- **`avg`** – [out] The average of vectors.
- **`xx`** – [in] The vectors of all models.

template<typename **`VALUETYPE`**>
inline void **`compute_std`**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const std::vector<std::vector<VALUETYPE>> &xx, const int &stride)

Compute the standard deviation of vectors.

Parameters

- **`std`** – [out] The standard deviation of vectors.
- **`avg`** – [in] The average of vectors.
- **`xx`** – [in] The vectors of all models.
- **`stride`** – [in] The stride to compute the deviation.

template<typename **`VALUETYPE`**>
inline void **`compute_relative_std`**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const VALUETYPE eps, const int &stride)

Compute the relative standard deviation of vectors.

Parameters

- **std** – [out] The standard deviation of vectors.
- **avg** – [in] The average of vectors.
- **eps** – [in] The level parameter for computing the deviation.
- **stride** – [in] The stride to compute the deviation.

template<typename **VALUETYPE**>
inline void **compute_std_f**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const std::vector<std::vector<VALUETYPE>> &xx)

Compute the standard deviation of forces.

Parameters

- **std** – [out] The standard deviation of forces.
- **avg** – [in] The average of forces.
- **xx** – [in] The vectors of all forces.

template<typename **VALUETYPE**>
inline void **compute_relative_std_f**(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const VALUETYPE eps)

Compute the relative standard deviation of forces.

Parameters

- **std** – [out] The relative standard deviation of forces.
- **avg** – [in] The relative average of forces.
- **eps** – [in] The level parameter for computing the deviation.

Class DeepTensor

- Defined in file_source_api_c_include_deepmd.hpp

Class Documentation

class **DeepTensor**

Deep Tensor.

Public Functions

inline **DeepTensor**()

Deep Tensor constructor without initialization.

inline **~DeepTensor**()

inline **DeepTensor**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

DeepTensor constructor with initialization.

Parameters

model – [in] The name of the frozen model file.

inline void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Initialize the DeepTensor.

Parameters

model – [in] The name of the frozen model file.

template<typename **VALUETYPE**>
inline void **compute**(std::vector<VALUETYPE> &tensor, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box)

Evaluate the tensor, force and virial by using this Deep Tensor.

Parameters

- **tensor** – [out] The atomic tensor.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).

template<typename **VALUETYPE**>
inline void **compute**(std::vector<VALUETYPE> &tensor, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list)

Evaluate the tensor, force and virial by using this Deep Tensor with the neighbor list.

Parameters

- **tensor** – [out] The tensor.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.

template<typename **VALUETYPE**>
inline void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box)

Evaluate the global tensor, force and virial by using this Deep Tensor.

Parameters

- **global_tensor** – [out] The global tensor.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.

- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).

template<typename **VALUETYPE**>
inline void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_tensor, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box)

Evaluate the global tensor, force, virial, atomic tensor, and atomic virial by using this Deep Tensor.

Parameters

- **global_tensor** – [out] The global tensor.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_tensor** – [out] The atomic tensor.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).

template<typename **VALUETYPE**>
inline void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list)

Evaluate the global tensor, force and virial by using this Deep Tensor with the neighbor list.

Parameters

- **global_tensor** – [out] The global tensor.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.

template<typename **VALUETYPE**>

inline void **compute**(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_tensor, std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost, const InputNlist &lmp_list)

Evaluate the global tensor, force, virial, atomic tensor, and atomic virial by using this Deep Tensor with the neighbor list.

Parameters

- **global_tensor** – [out] The global tensor.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom_tensor** – [out] The atomic tensor.
- **atom_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.

inline double **cutoff**() const

Get the cutoff radius.

Returns
The cutoff radius.

inline int **numb_types**() const

Get the number of types.

Returns
The number of types.

inline int **output_dim**() const

Get the output dimension.

Returns
The output dimension.

inline std::vector<int> **sel_types**() const

inline void **print_summary**(const std::string &pre) const

Print the summary of DeePMD-kit, including the version and the build information.

Parameters
pre – [in] The prefix to each line.

Class DipoleChargeModifier

- Defined in file_source_api_c_include_deepmd.hpp

Class Documentation

class **DipoleChargeModifier**

Public Functions

inline **DipoleChargeModifier**()

DipoleChargeModifier constructor without initialization.

inline **~DipoleChargeModifier**()

inline **DipoleChargeModifier**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

DipoleChargeModifier constructor with initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The rank of the GPU to be used.
- **name_scope** – [in] The name scope of the model.

inline void **init**(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")

Initialize the DipoleChargeModifier.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu_rank** – [in] The rank of the GPU to be used.
- **name_scope** – [in] The name scope of the model.

template<typename **VALUETYPE**>
inline void **compute**(std::vector<VALUETYPE> &dfcorr_, std::vector<VALUETYPE> &dvcorr_, const std::vector<VALUETYPE> &dcoord_, const std::vector<int> &datype_, const std::vector<VALUETYPE> &dbox, const std::vector<std::pair<int, int>> &pairs, const std::vector<VALUETYPE> &delef_, const int nghost, const InputNlist &lmp_list)

Evaluate the force and virial correction by using this dipole charge modifier.

Parameters

- **dfcorr_** – [out] The force correction on each atom.
- **dvcorr_** – [out] The virial correction.
- **dcoord_** – [in] The coordinates of atoms. The array should be of size 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.

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.

20.3.3 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

template<typename **FPTYPE**>
inline void **_DP_DeepTensorCompute**(DP_DeepTensor *dt, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, FPTYPE *global_tensor, FPTYPE *force, FPTYPE *virial, FPTYPE **atomic_energy, FPTYPE *atomic_virial, int *size_at)

Specialized Template Function _DP_DeepTensorCompute< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorCompute**<double>(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, double *global_tensor, double *force, double *virial, double **atomic_tensor, double *atomic_virial, int *size_at)

Specialized Template Function _DP_DeepTensorCompute< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorCompute**<float>(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, float *global_tensor, float *force, float *virial, float **atomic_tensor, float *atomic_virial, int *size_at)

Template Function _DP_DeepTensorComputeNList

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **FPTYPE**>
inline void **_DP_DeepTensorComputeNList**(DP_DeepTensor *dt, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, const int nghost, const DP_Nlist *nlist, FPTYPE *global_tensor, FPTYPE *force, FPTYPE *virial, FPTYPE **atomic_energy, FPTYPE *atomic_virial, int *size_at)

Specialized Template Function _DP_DeepTensorComputeNList< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorComputeNList**<double>(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, double *global_tensor, double *force, double *virial, double **atomic_tensor, double *atomic_virial, int *size_at)

Specialized Template Function _DP_DeepTensorComputeNList< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorComputeNList**<float>(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, float *global_tensor, float *force, float *virial, float **atomic_tensor, float *atomic_virial, int *size_at)

Template Function _DP_DeepTensorComputeTensor

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **FPTYPE**>
inline void **_DP_DeepTensorComputeTensor**(DP_DeepTensor *dt, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, FPTYPE **tensor, int *size)

Specialized Template Function _DP_DeepTensorComputeTensor< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorComputeTensor**<double>(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, double **tensor, int *size)

Specialized Template Function _DP_DeepTensorComputeTensor< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorComputeTensor**<float>(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, float **tensor, int *size)

Template Function _DP_DeepTensorComputeTensorNList

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **FPTYPE**>
inline void **_DP_DeepTensorComputeTensorNList**(DP_DeepTensor *dt, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, const int nghost, const DP_Nlist *nlist, FPTYPE **tensor, int *size)

Specialized Template Function _DP_DeepTensorComputeTensorNList< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorComputeTensorNList**<double>(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, double **tensor, int *size)

Specialized Template Function _DP_DeepTensorComputeTensorNList< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DeepTensorComputeTensorNList**<float>(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, float **tensor, int *size)

Template Function _DP_DipoleChargeModifierComputeNList

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **FPTYPE**>
inline void **_DP_DipoleChargeModifierComputeNList**(DP_DipoleChargeModifier *dcm, const int natom, const FPTYPE *coord, const int *atype, const FPTYPE *cell, const int *pairs, const int npairs, const FPTYPE *delef_, const int nghost, const DP_Nlist *nlist, FPTYPE *dfcorr_, FPTYPE *dvcorr_)

Specialized Template Function _DP_DipoleChargeModifierComputeNList< double >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DipoleChargeModifierComputeNList**<double>(DP_DipoleChargeModifier *dcm, const int natom, const double *coord, const int *atype, const double *cell, const int *pairs, const int npairs, const double *delef_, const int nghost, const DP_Nlist *nlist, double *dfcorr_, double *dvcorr_)

Specialized Template Function _DP_DipoleChargeModifierComputeNList< float >

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<>
inline void **_DP_DipoleChargeModifierComputeNList**<float>(DP_DipoleChargeModifier *dcm, const int natom, const float *coord, const int *atype, const float *cell, const int *pairs, const int npairs, const float *delef_, const int nghost, const DP_Nlist *nlist, float *dfcorr_, float *dvcorr_)

Function _DP_Get_Energy_Pointer(std::vector<double>&, const int)

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

inline double *`_DP_Get_Energy_Pointer`(std::vector<double> &vec, const int nframes)

Function _DP_Get_Energy_Pointer(double&, const int)

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

inline double *`_DP_Get_Energy_Pointer`(double &vec, const int nframes)

Function deepmd::hpp::convert_nlist

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

inline void `deepmd::hpp::convert_nlist`(InputNlist &to_nlist, std::vector<std::vector<int>> &from_nlist)

Convert int vector to InputNlist.

Parameters

- `to_nlist` – [out] InputNlist.
- `from_nlist` – [in] 2D int vector. The first axis represents the centeral atoms and the second axis represents the neighbor atoms.

Function deepmd::hpp::convert_pbtxt_to_pb

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

inline void `deepmd::hpp::convert_pbtxt_to_pb`(std::string fn_pb_txt, std::string fn_pb)

Convert pbtxt to pb.

Parameters

- `fn_pb_txt` – [in] Filename of the pb txt file.
- `fn_pb` – [in] Filename of the pb file.

Function deepmd::hpp::read_file_to_string

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

inline void **`deepmd::hpp::read_file_to_string`**(std::string model, std::string &file_content)

Read model file to a string.

Parameters

- **`model`** – [in] Path to the model.
- **`file_content`** – [out] Content of the model file.

Template Function deepmd::hpp::select_by_type

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **`VALUETYPE`**>
void **`deepmd::hpp::select_by_type`**(std::vector<int> &fwd_map, std::vector<int> &bkw_map, int &nghost_real, const std::vector<VALUETYPE> &dcoord_, const std::vector<int> &datype_, const int &nghost, const std::vector<int> &sel_type_)

Get forward and backward map of selected atoms by atom types.

Parameters

- **`fwd_map`** – [out] The forward map with size natoms.
- **`bkw_map`** – [out] The backward map with size nreal.
- **`nghost_real`** – [out] The number of selected ghost atoms.
- **`dcoord_`** – [in] The coordinates of all atoms. Reserved for compatibility.
- **`datype_`** – [in] The atom types of all atoms.
- **`nghost`** – [in] The number of ghost atoms.
- **`sel_type_`** – [in] The selected atom types.

Template Function deepmd::hpp::select_map

- Defined in file_source_api_c_include_deepmd.hpp

Function Documentation

template<typename **VT**>
void **deepmd::hpp::select_map**(std::vector<VT> &out, const std::vector<VT> &in, const std::vector<int> &fwd_map, const int &stride)

Apply the given map to a vector. Assume nframes is 1.

Template Parameters
VT – The value type of the vector. Only support int.

Parameters

- **out** – [out] The output vector.
- **in** – [in] The input vector.
- **fwd_map** – [in] The map.
- **stride** – [in] The stride of the input vector.

Function DP_ConvertPbtxtToPb

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_ConvertPbtxtToPb**(const char *c_pbtxt, const char *c_pb)

Convert PBtxt to PB.

Parameters

- **c_pbtxt** – [in] The name of the PBtxt file.
- **c_pb** – [in] The name of the PB file.

Function DP_DeepPotCheckOK

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char ***DP_DeepPotCheckOK**(DP_DeepPot *dp)

Check if there is any exceptions throw.

Parameters
dp – The DP to use.

Returns
const char* error message.

Function DP_DeepPotCompute

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **`DP_DeepPotCompute`**(DP_DeepPot *dp, const int natom, const double *coord, const int *atype, const double *cell, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP. (double version)

Attention
The number of frames is assumed to be 1.

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **`dp`** – [in] The DP to use.
- **`natoms`** – [in] The number of atoms.
- **`coord`** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **`atype`** – [in] The atom types. The array should contain natoms ints.
- **`box`** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **`energy`** – [out] Output energy.
- **`force`** – [out] Output force. The array should be of size natoms x 3.
- **`virial`** – [out] Output virial. The array should be of size 9.
- **`atomic_energy`** – [out] Output atomic energy. The array should be of size natoms.
- **`atomic_virial`** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotCompute2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotCompute2**(DP_DeepPot *dp, const int nframes, const int natom, const double *coord, const int *atype, const double *cell, const double *fparam, const double *aparam, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP. (double version)

Version
2

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputef

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotComputef**(DP_DeepPot *dp, const int natom, const float *coord, const int *atype, const float *cell, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP. (float version)

Attention
The number of frames is assumed to be 1.

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- `dp` – [in] The DP to use.
- `natoms` – [in] The number of atoms.
- `coord` – [in] The coordinates of atoms. The array should be of size natoms x 3.
- `atype` – [in] The atom types. The array should contain natoms ints.
- `box` – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- `energy` – [out] Output energy.
- `force` – [out] Output force. The array should be of size natoms x 3.
- `virial` – [out] Output virial. The array should be of size 9.
- `atomic_energy` – [out] Output atomic energy. The array should be of size natoms.
- `atomic_virial` – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputef2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotComputef2**(DP_DeepPot *dp, const int nframes, const int natom, const float *coord, const int *atype, const float *cell, const float *fparam, const float *aparam, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP. (float version)

Version
2

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputeMixedType

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotComputeMixedType**(DP_DeepPot *dp, const int nframes, const int natoms, const double *coord, const int *atype, const double *cell, const double *fparam, const double *aparam, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP with the mixed type. (double version)

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain nframes x natoms ints.

- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputeMixedTypef

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotComputeMixedTypef**(DP_DeepPot *dp, const int nframes, const int natoms, const float *coord, const int *atype, const float *cell, const float *fparam, const float *aparam, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP with the mixed type. (float version)

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain nframes x natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputeNList

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotComputeNList**(DP_DeepPot *dp, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, const int ago, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP with the neighbor list. (double version)

Attention

The number of frames is assumed to be 1.

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputeNList2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeepPotComputeNList2`(DP_DeepPot *dp, const int nframes, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, const int ago, const double *fparam, const double *aparam, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP with the neighbor list. (double version)

Version
2

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputeNListf

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **`DP_DeepPotComputeNListf`**(DP_DeepPot *dp, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, const int ago, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP with the neighbor list. (float version)

Attention
The number of frames is assumed to be 1.

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **`dp`** – [in] The DP to use.
- **`natoms`** – [in] The number of atoms.
- **`coord`** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **`atype`** – [in] The atom types. The array should contain natoms ints.
- **`box`** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **`nghost`** – [in] The number of ghost atoms.
- **`nlist`** – [in] The neighbor list.
- **`ago`** – [in] Update the internal neighbour list if ago is 0.
- **`energy`** – [out] Output energy.
- **`force`** – [out] Output force. The array should be of size natoms x 3.
- **`virial`** – [out] Output virial. The array should be of size 9.
- **`atomic_energy`** – [out] Output atomic energy. The array should be of size natoms.
- **`atomic_virial`** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotComputeNListf2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeepPotComputeNListf2`(DP_DeepPot *dp, const int nframes, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, const int ago, const float *fparam, const float *aparam, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP with the neighbor list. (float version)

Version
2

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **`dp`** – [in] The DP to use.
- **`nframes`** – [in] The number of frames.
- **`natoms`** – [in] The number of atoms.
- **`coord`** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **`atype`** – [in] The atom types. The array should contain natoms ints.
- **`box`** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **`nghost`** – [in] The number of ghost atoms.
- **`nlist`** – [in] The neighbor list.
- **`ago`** – [in] Update the internal neighbour list if ago is 0.
- **`fparam`** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **`aparam`** – [in] The atom parameters. The array can be of size nframes x dim_aparam.
- **`energy`** – [out] Output energy.
- **`force`** – [out] Output force. The array should be of size natoms x 3.
- **`virial`** – [out] Output virial. The array should be of size 9.
- **`atomic_energy`** – [out] Output atomic energy. The array should be of size natoms.
- **`atomic_virial`** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotGetCutoff

- Defined in file_source_api_c_include_c_api.h

Function Documentation

double **`DP_DeepPotGetCutoff`**(DP_DeepPot *dp)

Get the type map of a DP.

Parameters
: **`dp`** – [in] The DP to use.

Returns
: The cutoff radius.

Function DP_DeepPotGetDimAParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int **`DP_DeepPotGetDimAParam`**(DP_DeepPot *dp)

Get the dimension of atomic parameters of a DP.

Parameters
: **`dp`** – [in] The DP to use.

Returns
: The dimension of atomic parameters of the DP.

Function DP_DeepPotGetDimFParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int **`DP_DeepPotGetDimFParam`**(DP_DeepPot *dp)

Get the dimension of frame parameters of a DP.

Parameters
: **`dp`** – [in] The DP to use.

Returns
: The dimension of frame parameters of the DP.

Function DP_DeepPotGetNumbTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int **`DP_DeepPotGetNumbTypes`**(DP_DeepPot *dp)

Get the number of types of a DP.

Parameters
: **`dp`** – [in] The DP to use.

Returns
: The number of types of the DP.

Function DP_DeepPotGetNumbTypesSpin

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int **`DP_DeepPotGetNumbTypesSpin`**(DP_DeepPot *dp)

Get the number of types with spin of a DP.

Parameters
: **`dp`** – [in] The DP to use.

Returns
: The number of types with spin of the DP.

Function DP_DeepPotGetTypeMap

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char ***`DP_DeepPotGetTypeMap`**(DP_DeepPot *dp)

Get the type map of a DP.

Parameters
: **`dp`** – [in] The DP to use.

Returns
: The type map of the DP.

Function DP_DeepPotModelDeviCheckOK

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char *`DP_DeepPotModelDeviCheckOK`(DP_DeepPotModelDevi *dp)

Check if there is any exceptions throw.

Parameters
: **dp** – The DP model deviation to use.

Returns
: const char* error message.

Function DP_DeepPotModelDeviComputeNList

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeepPotModelDeviComputeNList`(DP_DeepPotModelDevi *dp, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, const int ago, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (double version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP model deviation to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.

- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotModelDeviComputeNList2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotModelDeviComputeNList2**(DP_DeepPotModelDevi *dp, const int nframes, const int natoms, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, const int ago, const double *fparam, const double *aparam, double *energy, double *force, double *virial, double *atomic_energy, double *atomic_virial)

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (double version)

Version
2

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP model deviation to use.
- **nframes** – [in] The number of frames. Only support 1 for now.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x natoms x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.

- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotModelDeviComputeNListf

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotModelDeviComputeNListf**(DP_DeepPotModelDevi *dp, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, const int ago, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (float version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP model deviation to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotModelDeviComputeNListf2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepPotModelDeviComputeNListf2**(DP_DeepPotModelDevi *dp, const int nframes, const int natoms, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, const int ago, const float *fparam, const float *aparam, double *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (float version)

Version
2

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP model deviation to use.
- **nframes** – [in] The number of frames. Only support 1 for now.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x natoms x dim_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

Function DP_DeepPotModelDeviGetCutoff

- Defined in file_source_api_c_include_c_api.h

Function Documentation

double `DP_DeepPotModelDeviGetCutoff`(DP_DeepPotModelDevi *dp)

Get the type map of a DP model deviation.

Parameters
: `dp` – [in] The DP model deviation to use.

Returns
: The cutoff radius.

Function DP_DeepPotModelDeviGetDimAParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int `DP_DeepPotModelDeviGetDimAParam`(DP_DeepPotModelDevi *dp)

Get the dimension of atomic parameters of a DP Model Deviation.

Parameters
: `dp` – [in] The DP Model Deviation to use.

Returns
: The dimension of atomic parameters of the DP Model Deviation.

Function DP_DeepPotModelDeviGetDimFParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int `DP_DeepPotModelDeviGetDimFParam`(DP_DeepPotModelDevi *dp)

Get the dimension of frame parameters of a DP Model Deviation.

Parameters
: `dp` – [in] The DP Model Deviation to use.

Returns
: The dimension of frame parameters of the DP Model Deviation.

Function DP_DeepPotModelDeviGetNumbTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int **DP_DeepPotModelDeviGetNumbTypes**(DP_DeepPotModelDevi *dp)

Get the number of types of a DP model deviation.

Parameters
: **dp** – [in] The DP model deviation to use.

Returns
: The number of types of the DP model deviation.

Function DP_DeepPotModelDeviGetNumbTypesSpin

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int **DP_DeepPotModelDeviGetNumbTypesSpin**(DP_DeepPotModelDevi *dp)

Get the number of types with spin of a DP model deviation.

Parameters
: **dp** – [in] The DP model deviation to use.

Returns
: The number of types with spin of the DP model deviation.

Function DP_DeepTensorCheckOK

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char ***DP_DeepTensorCheckOK**(DP_DeepTensor *dt)

Check if there is any exceptions throw.

Parameters
: **dt** – The Deep Tensor to use.

Returns
: const char* error message.

Function DP_DeepTensorCompute

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **`DP_DeepTensorCompute`**(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, double *global_tensor, double *force, double *virial, double **atomic_tensor, double *atomic_virial, int *size_at)

Evaluate the global tensor, force and virial by using a DP. (double version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **`dt`** – [in] The Deep Tensor to use.
- **`natoms`** – [in] The number of atoms.
- **`coord`** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **`atype`** – [in] The atom types. The array should contain natoms ints.
- **`box`** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **`global_tensor`** – [out] Output global tensor.
- **`force`** – [out] Output force. The array should be of size natoms x 3.
- **`virial`** – [out] Output virial. The array should be of size 9.
- **`atomic_tensor`** – [out] Output atomic tensor. The array should be of size natoms.
- **`atomic_virial`** – [out] Output atomic virial. The array should be of size natoms x 9.
- **`size_at`** – [out] Output size of atomic tensor.

Function DP_DeepTensorComputef

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **`DP_DeepTensorComputef`**(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, float *global_tensor, float *force, float *virial, float **atomic_tensor, float *atomic_virial, int *size_at)

Evaluate the global tensor, force and virial by using a DP. (float version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **global_tensor** – [out] Output global tensor.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic_tensor** – [out] Output atomic tensor. The array should be of size natoms.
- **atomic_virial** – [out] Output atomic virial. The array should be of size natoms x 9.
- **size_at** – [out] Output size of atomic tensor.

Function DP_DeepTensorComputeNList

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepTensorComputeNList**(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, double *global_tensor, double *force, double *virial, double **atomic_tensor, double *atomic_virial, int *size_at)

Evaluate the global tensor, force and virial by using a DP with the neighbor list. (double version)

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **global_tensor** – [out] Output global tensor.
- **force** – [out] Output force. The array should be of size natoms x 3.

- **`virial`** – [out] Output virial. The array should be of size 9.
- **`atomic_tensor`** – [out] Output atomic tensor. The array should be of size natoms.
- **`atomic_virial`** – [out] Output atomic virial. The array should be of size natoms x 9.
- **`size_at`** – [out] Output size of atomic tensor.

Function DP_DeepTensorComputeNListf

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **`DP_DeepTensorComputeNListf`**(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, float *global_tensor, float *force, float *virial, float **atomic_tensor, float *atomic_virial, int *size_at)

Evaluate the global tensor, force and virial by using a DP with the neighbor list. (float version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **`dt`** – [in] The Deep Tensor to use.
- **`natoms`** – [in] The number of atoms.
- **`coord`** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **`atype`** – [in] The atom types. The array should contain natoms ints.
- **`box`** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **`nghost`** – [in] The number of ghost atoms.
- **`nlist`** – [in] The neighbor list.
- **`global_tensor`** – [out] Output global tensor.
- **`force`** – [out] Output force. The array should be of size natoms x 3.
- **`virial`** – [out] Output virial. The array should be of size 9.
- **`atomic_tensor`** – [out] Output atomic tensor. The array should be of size natoms.
- **`atomic_virial`** – [out] Output atomic virial. The array should be of size natoms x 9.
- **`size_at`** – [out] Output size of atomic tensor.

Function DP_DeepTensorComputeTensor

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeepTensorComputeTensor`(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, double **tensor, int *size)

Evaluate the tensor by using a DP. (double version)

Parameters

- **`dt`** – [in] The Deep Tensor to use.
- **`natoms`** – [in] The number of atoms.
- **`coord`** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **`atype`** – [in] The atom types. The array should contain natoms ints.
- **`box`** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **`tensor`** – [out] Output tensor.

Function DP_DeepTensorComputeTensorf

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_DeepTensorComputeTensorf`(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, float **tensor, int *size)

Evaluate the tensor by using a DP. (float version)

Parameters

- **`dt`** – [in] The Deep Tensor to use.
- **`natoms`** – [in] The number of atoms.
- **`coord`** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **`atype`** – [in] The atom types. The array should contain natoms ints.
- **`box`** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **`tensor`** – [out] Output tensor.
- **`size`** – [out] Output size of the tensor.

Function DP_DeepTensorComputeTensorNList

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepTensorComputeTensorNList**(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype, const double *cell, const int nghost, const DP_Nlist *nlist, double **tensor, int *size)

Evaluate the tensor by using a DP with the neighbor list. (double version)

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **tensor** – [out] Output tensor.
- **size** – [out] Output size of the tensor.

Function DP_DeepTensorComputeTensorNListf

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_DeepTensorComputeTensorNListf**(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist, float **tensor, int *size)

Evaluate the tensor by using a DP with the neighbor list. (float version)

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.

- **tensor** – [out] Output tensor.
- **size** – [out] Output size of the tensor.

Function DP_DeepTensorGetCutoff

- Defined in file_source_api_c_include_c_api.h

Function Documentation

double **DP_DeepTensorGetCutoff**(DP_DeepTensor *dt)

Get the type map of a Deep Tensor.

Parameters

dt – [in] The Deep Tensor to use.

Returns

The cutoff radius.

Function DP_DeepTensorGetNumbSelTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int **DP_DeepTensorGetNumbSelTypes**(DP_DeepTensor *dt)

Get the number of sel types of a Deep Tensor.

Parameters

dt – [in] The Deep Tensor to use.

Returns

The number of sel types

Function DP_DeepTensorGetNumbTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int **DP_DeepTensorGetNumbTypes**(DP_DeepTensor *dt)

Get the type map of a Deep Tensor.

Parameters

dt – [in] The Deep Tensor to use.

Returns

The number of types of the Deep Tensor.

Function DP_DeepTensorGetOutputDim

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int **`DP_DeepTensorGetOutputDim`**(DP_DeepTensor *dt)

Get the output dimension of a Deep Tensor.

Parameters
: **`dt`** – [in] The Deep Tensor to use.

Returns
: The output dimension of the Deep Tensor.

Function DP_DeepTensorGetSelTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int ***`DP_DeepTensorGetSelTypes`**(DP_DeepTensor *dt)

Get sel types of a Deep Tensor.

Parameters
: **`dt`** – [in] The Deep Tensor to use.

Returns
: The sel types

Function DP_DipoleChargeModifierCheckOK

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char ***`DP_DipoleChargeModifierCheckOK`**(DP_DipoleChargeModifier *dcm)

Check if there is any exceptions throw.

Parameters
: **`dcm`** – The DipoleChargeModifier to use.

Returns
: const char* error message.

Function DP_DipoleChargeModifierComputeNList

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **`DP_DipoleChargeModifierComputeNList`**(DP_DipoleChargeModifier *dcm, const int natom, const double *coord, const int *atype, const double *cell, const int *pairs, const int npairs, const double *delef_, const int nghost, const DP_Nlist *nlist, double *dfcorr_, double *dvcorr_)

Evaluate the force and virial correction by using a dipole charge modifier with the neighbor list. (double version)

> Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **`dcm`** – [in] The dipole charge modifier to use.
- **`natoms`** – [in] The number of atoms.
- **`coord`** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **`atype`** – [in] The atom types. The array should contain natoms ints.
- **`cell`** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **`pairs`** – [in] The pairs of atoms. The list should contain npairs pairs of ints.
- **`npairs`** – [in] The number of pairs.
- **`delef_`** – [in] The electric field on each atom. The array should be of size nframes x natoms x 3.
- **`nghost`** – [in] The number of ghost atoms.
- **`nlist`** – [in] The neighbor list.
- **`dfcorr_`** – [out] Output force correction. The array should be of size natoms x 3.
- **`dvcorr_`** – [out] Output virial correction. The array should be of size 9.

Function DP_DipoleChargeModifierComputeNListf

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **`DP_DipoleChargeModifierComputeNListf`**(DP_DipoleChargeModifier *dcm, const int natom, const float *coord, const int *atype, const float *cell, const int *pairs, const int npairs, const float *delef_, const int nghost, const DP_Nlist *nlist, float *dfcorr_, float *dvcorr_)

Evaluate the force and virial correction by using a dipole charge modifier with the neighbor list. (float version)

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **`dcm`** – [in] The dipole charge modifier to use.
- **`natoms`** – [in] The number of atoms.
- **`coord`** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **`atype`** – [in] The atom types. The array should contain natoms ints.
- **`cell`** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **`pairs`** – [in] The pairs of atoms. The list should contain npairs pairs of ints.
- **`npairs`** – [in] The number of pairs.
- **`delef_`** – [in] The electric field on each atom. The array should be of size nframes x natoms x 3.
- **`nghost`** – [in] The number of ghost atoms.
- **`nlist`** – [in] The neighbor list.
- **`dfcorr_`** – [out] Output force correction. The array should be of size natoms x 3.
- **`dvcorr_`** – [out] Output virial correction. The array should be of size 9.

Function DP_DipoleChargeModifierGetCutoff

- Defined in file_source_api_c_include_c_api.h

Function Documentation

double **`DP_DipoleChargeModifierGetCutoff`**(DP_DipoleChargeModifier *dt)

Get the type map of a DipoleChargeModifier.

Parameters

`dcm` – [in] The DipoleChargeModifier to use.

Returns

The cutoff radius.

Function DP_DipoleChargeModifierGetNumbSelTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int **`DP_DipoleChargeModifierGetNumbSelTypes`**(DP_DipoleChargeModifier *dt)

Get the number of sel types of a DipoleChargeModifier.

Parameters
: **`dcm`** – [in] The DipoleChargeModifier to use.

Returns
: The number of sel types

Function DP_DipoleChargeModifierGetNumbTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int **`DP_DipoleChargeModifierGetNumbTypes`**(DP_DipoleChargeModifier *dt)

Get the type map of a DipoleChargeModifier.

Parameters
: **`dcm`** – [in] The DipoleChargeModifier to use.

Returns
: The number of types of the DipoleChargeModifier.

Function DP_DipoleChargeModifierGetSelTypes

- Defined in file_source_api_c_include_c_api.h

Function Documentation

int ***`DP_DipoleChargeModifierGetSelTypes`**(DP_DipoleChargeModifier *dt)

Get sel types of a DipoleChargeModifier.

Parameters
: **`dcm`** – [in] The DipoleChargeModifier to use.

Returns
: The sel types

Function DP_NewDeepPot

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DeepPot ***DP_NewDeepPot**(const char *c_model)

DP constructor with initialization.

Parameters
`c_model` – [in] The name of the frozen model file.

Returns
A pointer to the deep potential.

Function DP_NewDeepPotModelDevi

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DeepPotModelDevi ***DP_NewDeepPotModelDevi**(const char **c_models, int n_models)

DP model deviation constructor with initialization.

Parameters

- `c_models` – [in] The array of the name of the frozen model file.
- `nmodels` – [in] The number of models.

Function DP_NewDeepPotModelDeviWithParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DeepPotModelDevi ***DP_NewDeepPotModelDeviWithParam**(const char **c_model, const int n_models, const int gpu_rank, const char **c_file_contents, const int n_file_contents, const int *size_file_contents)

DP model deviation constructor with initialization.

Parameters

- `c_models` – [in] The array of the name of the frozen model file.
- `nmodels` – [in] The number of models.
- `gpu_rank` – [in] The rank of the GPU.
- `c_file_contents` – [in] The contents of the model file.
- `n_file_contents` – [in] The number of the contents of the model file.

- **size_file_contents** – [in] The sizes of the contents of the model file.

Returns
DP_DeepPotModelDevi* A pointer to the deep potential model deviation.

Function DP_NewDeepPotWithParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DeepPot ***DP_NewDeepPotWithParam**(const char *c_model, const int gpu_rank, const char *c_file_content)

DP constructor with initialization.

Parameters

- **c_model** – The name of the frozen model file.
- **gpu_rank** – The rank of the GPU.
- **c_file_content** – Broken implementation. Use DP_NewDeepPotWithParam2 instead.

Returns
DP_DeepPot* A pointer to the deep potential.

Function DP_NewDeepPotWithParam2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DeepPot ***DP_NewDeepPotWithParam2**(const char *c_model, const int gpu_rank, const char *c_file_content, const int size_file_content)

DP constructor with initialization.

Version
2

Parameters

- **c_model** – The name of the frozen model file.
- **gpu_rank** – The rank of the GPU.
- **c_file_content** – The content of the model file.
- **size_file_content** – The size of the model file.

Returns
DP_DeepPot* A pointer to the deep potential.

Function DP_NewDeepTensor

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DeepTensor ***DP_NewDeepTensor**(const char *c_model)

Deep Tensor constructor with initialization.

Parameters
: **c_model** – [in] The name of the frozen model file.

Returns
: A pointer to the deep tensor.

Function DP_NewDeepTensorWithParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DeepTensor ***DP_NewDeepTensorWithParam**(const char *c_model, const int gpu_rank, const char *c_name_scope)

Deep Tensor constructor with initialization.

Parameters

- **c_model** – The name of the frozen model file.
- **gpu_rank** – The rank of the GPU.
- **c_name_scope** – The name scope.

Returns
: DP_DeepTensor* A pointer to the deep tensor.

Function DP_NewDipoleChargeModifier

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DipoleChargeModifier ***DP_NewDipoleChargeModifier**(const char *c_model)

Dipole charge modifier constructor with initialization.

Parameters
: **c_model** – [in] The name of the frozen model file.

Returns
: A pointer to the dipole charge modifier.

Function DP_NewDipoleChargeModifierWithParam

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_DipoleChargeModifier ***DP_NewDipoleChargeModifierWithParam**(const char *c_model, const int gpu_rank, const char *c_name_scope)

Dipole charge modifier constructor with initialization.

Parameters

- **c_model** – The name of the frozen model file.
- **gpu_rank** – The rank of the GPU.
- **c_name_scope** – The name scope.

Returns

DP_DipoleChargeModifier* A pointer to the dipole charge modifier.

Function DP_NewNlist

- Defined in file_source_api_c_include_c_api.h

Function Documentation

DP_Nlist ***DP_NewNlist**(int inum_, int *ilist_, int *numneigh_, int **firstneigh_)

Create a new neighbor list.

Parameters

- **inum_** – [in] Number of core region atoms
- **Array** – [in] stores the core region atom's index
- **Array** – [in] stores the core region atom's neighbor atom number
- **Array** – [in] stores the core region atom's neighbor index

Returns

A pointer to the neighbor list.

Function DP_NlistCheckOK

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char ***DP_NlistCheckOK**(DP_Nlist *dp)

Check if there is any exceptions throw.

Parameters
: **dp** – The neighbor list to use.

Returns
: const char* error message.

Function DP_PrintSummary

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void **DP_PrintSummary**(const char *c_pre)

Print the summary of DeePMD-kit, including the version and the build information.

Parameters
: **c_pre** – [in] The prefix to each line.

Function DP_ReadFileToChar

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char ***DP_ReadFileToChar**(const char *c_model)

Read a file to a char array.

Parameters
: **c_model** – [in] The name of the file.

Returns
: const char* The char array.

Function DP_ReadFileToChar2

- Defined in file_source_api_c_include_c_api.h

Function Documentation

const char *`DP_ReadFileToChar2`(const char *c_model, int *size)

Read a file to a char array. This version can handle string with '\0'.

Version

2

Parameters

- **`c_model`** – [in] The name of the file.
- **`size`** – [out] The size of the char array.

Returns

const char* The char array.

Function DP_SelectByType

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_SelectByType`(const int natoms, const int *atype, const int nghost, const int nsel_type, const int *sel_type, int *fwd_map, int *nreal, int *bkw_map, int *nghost_real)

Get forward and backward map of selected atoms by atom types.

Parameters

- **`natoms`** – [in] The number of atoms.
- **`atype`** – [in] The atom types of all atoms.
- **`nghost`** – [in] The number of ghost atoms.
- **`nsel_type`** – [in] The number of selected atom types.
- **`sel_type`** – [in] The selected atom types.
- **`fwd_map`** – [out] The forward map with size natoms.
- **`nreal`** – [out] The number of selected real atoms.
- **`bkw_map`** – [out] The backward map with size nreal.
- **`nghost_real`** – [out] The number of selected ghost atoms.

Function DP_SelectMapInt

- Defined in file_source_api_c_include_c_api.h

Function Documentation

void `DP_SelectMapInt`(const int *in, const int *fwd_map, const int stride, const int nall1, const int nall2, int *out)

Apply the given map to a vector. Assume nframes is 1.

Parameters

- **`in`** – [in] The input vector.
- **`fwd_map`** – [in] The map.
- **`stride`** – [in] The stride of the input vector.
- **`nall1`** – [in] The number of atoms in the input vector.
- **`nall2`** – [out] The number of atoms in the output vector.
- **`out`** – [out] The output vector.

20.3.4 Defines

Define DP_CHECK_OK

- Defined in file_source_api_c_include_deepmd.hpp

Define Documentation

`DP_CHECK_OK`(check_func, dp)

Check if any exceptions throw in the C++ API. Throw if possible.

Define DP_NEW_OK

- Defined in file_source_api_c_include_c_api_internal.h

Define Documentation

`DP_NEW_OK`(dpcls, xx)

Define DP_REQUIRES_OK

- Defined in file_source_api_c_include_c_api_internal.h

Define Documentation

`DP_REQUIRES_OK`(dp, xx)

20.3.5 Typedefs

Typedef DP_DeepPot

- Defined in file_source_api_c_include_c_api.h

Typedef Documentation

typedef struct DP_DeepPot `DP_DeepPot`

The deep potential.

Typedef DP_DeepPotModelDevi

- Defined in file_source_api_c_include_c_api.h

Typedef Documentation

typedef struct DP_DeepPotModelDevi `DP_DeepPotModelDevi`

The deep potential model deviation.

Typedef DP_DeepTensor

- Defined in file_source_api_c_include_c_api.h

Typedef Documentation

typedef struct DP_DeepTensor `DP_DeepTensor`

The deep tensor.

Typedef DP_DipoleChargeModifier

- Defined in file_source_api_c_include_c_api.h

Typedef Documentation

typedef struct DP_DipoleChargeModifier `DP_DipoleChargeModifier`

The dipole charge modifier.

Typedef DP_Nlist

- Defined in file_source_api_c_include_c_api.h

Typedef Documentation

typedef struct DP_Nlist `DP_Nlist`

Neighbor list.

CHAPTER

TWENTYONE

CORE API

21.1 Class Hierarchy

- Namespace deepmd
  - Struct deepmd_exception
  - Struct deepmd_exception_oom
  - Template Struct EwaldParameters
  - Struct InputNlist
  - Template Struct Region
- Template Struct DescrptSeRGPUExecuteFunctor
- Template Struct GeluGPUExecuteFunctor
- Template Struct GeluGradGPUExecuteFunctor
- Template Struct GeluGradGradGPUExecuteFunctor
- Template Struct ProdForceSeAGPUExecuteFunctor
- Template Struct ProdForceSeRGPUExecuteFunctor
- Template Struct ProdVirialSeAGPUExecuteFunctor
- Template Struct ProdVirialSeRGPUExecuteFunctor
- Template Struct TabulateCheckerGPUExecuteFunctor
- Template Struct TabulateFusionGPUExecuteFunctor
- Template Struct TabulateFusionGradGPUExecuteFunctor
- Template Class SimulationRegion
- Union U_Flt64_Int64

21.2 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_pair_tab.h
      - file_source_lib_include_prod_env_mat.h
      - file_source_lib_include_prod_env_mat_nvnmd.h
      - file_source_lib_include_prod_force.h
      - file_source_lib_include_prod_force_grad.h
      - file_source_lib_include_prod_virial.h
      - file_source_lib_include_prod_virial_grad.h
      - file_source_lib_include_region.h
      - file_source_lib_include_SimulationRegion.h
      - file_source_lib_include_SimulationRegion_Impl.h
      - file_source_lib_include_soft_min_switch.h
      - file_source_lib_include_soft_min_switch_force.h
      - file_source_lib_include_soft_min_switch_force_grad.h
      - file_source_lib_include_soft_min_switch_virial.h
      - file_source_lib_include_soft_min_switch_virial_grad.h
      - file_source_lib_include_switcher.h
      - file_source_lib_include_tabulate.h

- file_source_lib_include_utilities.h

21.3 Full API

21.3.1 Namespaces

Namespace deepmd

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

- 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_cuda
- Template Function deepmd::tabulate_fusion_se_a_grad_cpu
- Template Function deepmd::tabulate_fusion_se_a_grad_gpu_cuda
- Template Function deepmd::tabulate_fusion_se_a_grad_grad_cpu
- Template Function deepmd::tabulate_fusion_se_a_grad_grad_gpu_cuda
- Template Function deepmd::tabulate_fusion_se_r_cpu
- Template Function deepmd::tabulate_fusion_se_r_gpu_cuda
- Template Function deepmd::tabulate_fusion_se_r_grad_cpu
- Template Function deepmd::tabulate_fusion_se_r_grad_gpu_cuda
- Template Function deepmd::tabulate_fusion_se_r_grad_grad_cpu
- Template Function deepmd::tabulate_fusion_se_r_grad_grad_gpu_cuda
- Template Function deepmd::tabulate_fusion_se_t_cpu
- Template Function deepmd::tabulate_fusion_se_t_gpu_cuda
- Template Function deepmd::tabulate_fusion_se_t_grad_cpu
- Template Function deepmd::tabulate_fusion_se_t_grad_gpu_cuda
- Template Function deepmd::tabulate_fusion_se_t_grad_grad_cpu
- Template Function deepmd::tabulate_fusion_se_t_grad_grad_gpu_cuda
- Template Function deepmd::test_encoding_decoding_nbor_info_gpu_cuda
- Function deepmd::use_nei_info_cpu
- Function deepmd::use_nei_info_gpu
- Function deepmd::use_nlist_map
- Template Function deepmd::volume_cpu
- Template Function deepmd::volume_gpu

Variables

- Variable deepmd::ElectrostaticConvertion

Namespace std

21.3.2 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

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

inline **`deepmd_exception`**`()`

inline **`deepmd_exception`**`(`const std::string &msg`)`

Struct deepmd_exception_oom

- Defined in file_source_lib_include_errors.h

Inheritance Relationships

Base Type

- `public deepmd::deepmd_exception` (Struct deepmd_exception)

Struct Documentation

struct **`deepmd_exception_oom`** : public deepmd::deepmd_exception

Public Functions

inline **`deepmd_exception_oom`**`()`

inline **`deepmd_exception_oom`**`(`const std::string &msg`)`

Template Struct EwaldParameters

- Defined in file_source_lib_include_ewald.h

Struct Documentation

template<typename **`VALUETYPE`**>

struct **`EwaldParameters`**

Public Members

VALUETYPE **`rcut`** = 6.0

VALUETYPE **`beta`** = 2

VALUETYPE **`spacing`** = 4

Struct InputNlist

- Defined in file_source_lib_include_neighbor_list.h

Struct Documentation

struct **`InputNlist`**

Construct InputNlist with the input LAMMPS nbor list info.

Public Functions

inline **InputNlist**()

inline **InputNlist**(int inum_, int *ilist_, int *numneigh_, int **firstneigh_)

inline **~InputNlist**()

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**

Public Functions

Region()

~Region()

Public Members

FPTYPE ***boxt**

FPTYPE ***rec_boxt**

Template Struct DescrptSeRGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **DescrptSeRGPUExecuteFunctor**

Public Functions

void **operator()**(const FPTYPE *coord, const int *type, const int *ilist, const int *jrange, const int *jlist, int *array_int, unsigned long long *array_longlong, const FPTYPE *avg, const FPTYPE *std, FPTYPE *descript, FPTYPE *descript_deriv, FPTYPE *rij, int *nlist, const int nloc, const int nall, const int nnei, const int ndescrpt, const float rcut_r, const float rcut_r_smth, const std::vector<int> sec_a, const bool fill_nei_a, const int MAGIC_NUMBER)

Template Struct GeluGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **GeluGPUExecuteFunctor**

Public Functions

void **operator()**(const FPTYPE *in, FPTYPE *out, const int size)

Template Struct GeluGradGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **GeluGradGPUExecuteFunctor**

Public Functions

void **operator()**(const FPTYPE *dy, const FPTYPE *in, FPTYPE *out, const int size)

Template Struct GeluGradGradGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **GeluGradGradGPUExecuteFunctor**

Public Functions

void **operator()**(const FPTYPE *dy, const FPTYPE *dy_, const FPTYPE *in, FPTYPE *out, const int size)

Template Struct ProdForceSeAGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **ProdForceSeAGPUExecuteFunctor**

Public Functions

void **operator()**(FPTYPE *force, const FPTYPE *net_derive, const FPTYPE *in_deriv, const int *nlist, const int nloc, const int nall, const int nnei, const int ndescrpt, const int n_a_sel, const int n_a_shift)

Template Struct ProdForceSeRGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **ProdForceSeRGPUExecuteFunctor**

Public Functions

void **operator()**(FPTYPE *force, const FPTYPE *net_derive, const FPTYPE *in_deriv, const int *nlist, const int nloc, const int nall, const int nnei, const int ndescrpt)

Template Struct ProdVirialSeAGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **ProdVirialSeAGPUExecuteFunctor**

Public Functions

void **operator()**(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE *in_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int nnei, const int ndescrpt, const int n_a_sel, const int n_a_shift)

Template Struct ProdVirialSeRGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **ProdVirialSeRGPUExecuteFunctor**

Public Functions

void **operator()**(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE *in_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int nnei, const int ndescrpt)

Template Struct TabulateCheckerGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **TabulateCheckerGPUExecuteFunctor**

Public Functions

void **operator()**(const FPTYPE *table_info, const FPTYPE *in, int *out, const int nloc, const int nnei)

Template Struct TabulateFusionGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **TabulateFusionGPUExecuteFunctor**

Public Functions

void **operator()**(const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *in, const FPTYPE *ff, const int nloc, const int nnei, const int last_layer_size, FPTYPE *out)

Template Struct TabulateFusionGradGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename **FPTYPE**>

struct **TabulateFusionGradGPUExecuteFunctor**

Public Functions

void **operator()**(const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *in, const FPTYPE *ff, const FPTYPE *dy, const int nloc, const int nnei, const int last_layer_size, FPTYPE *dy_dx, FPTYPE *dy_df)

Template Class SimulationRegion

- Defined in file_source_lib_include_SimulationRegion.h

Class Documentation

template<typename **VALUETYPE**>

class **SimulationRegion**

Public Functions

inline void **reinitBox**(const double *boxv)

inline void **affineTransform**(const double *affine_map)

inline void **reinitOrigin**(const double *orig)

inline void **reinitOrigin**(const std::vector<double> &orig)

void **backup**()

void **recover**()

SimulationRegion()

~SimulationRegion()

inline double ***getBoxTensor**()

inline const double ***getBoxTensor**() const

inline double ***getRecBoxTensor**()

inline const double ***getRecBoxTensor**() const

inline double ***getBoxOrigin**()

inline const double ***getBoxOrigin**() const

inline double **getVolume**() const

inline void **toFaceDistance**(double *dd) const

inline void **phys2Inter**(double *i_v, const VALUETYPE *p_v) const

inline void **inter2Phys**(VALUETYPE *p_v, const double *i_v) const

inline bool **isPeriodic**(const int dim) const inline double *`getShiftVec`(const int index = 0)

inline const double *`getShiftVec`(const int index = 0) const

inline int `getShiftIndex`(const int *idx) const

inline int `getNullShiftIndex`() const

inline void `shiftCoord`(const int *idx, VALUETYPE &x, VALUETYPE &y, VALUETYPE &z) const

inline void `diffNearestNeighbor`(const VALUETYPE *r0, const VALUETYPE *r1, VALUETYPE *phys) const

inline virtual void `diffNearestNeighbor`(const VALUETYPE x0, const VALUETYPE y0, const VALUETYPE z0, const VALUETYPE x1, const VALUETYPE y1, const VALUETYPE z1, VALUETYPE &dx, VALUETYPE &dy, VALUETYPE &dz) const

inline virtual void `diffNearestNeighbor`(const VALUETYPE x0, const VALUETYPE y0, const VALUETYPE z0, const VALUETYPE x1, const VALUETYPE y1, const VALUETYPE z1, VALUETYPE &dx, VALUETYPE &dy, VALUETYPE &dz, int &shift_x, int &shift_y, int &shift_z) const

inline virtual void `diffNearestNeighbor`(const VALUETYPE x0, const VALUETYPE y0, const VALUETYPE z0, const VALUETYPE x1, const VALUETYPE y1, const VALUETYPE z1, VALUETYPE &dx, VALUETYPE &dy, VALUETYPE &dz, VALUETYPE &shift_x, VALUETYPE &shift_y, VALUETYPE &shift_z) const

Public Static Functions

static inline int `compactIndex`(const int *idx)

static inline int `getNumbShiftVec`()

static inline int `getShiftVecTotalSize`()

Protected Functions

void `computeShiftVec`()

inline double *`getInterShiftVec`(const int index = 0)

inline const double *`getInterShiftVec`(const int index = 0) const

Protected Attributes

double **shift_vec**[shift_vec_size]

double **inter_shift_vec**[shift_vec_size]

Protected Static Functions

static inline int **index3to1**(const int tx, const int ty, const int tz)

Protected Static Attributes

static const int **SPACENDIM** = 3

static const int **DBOX_XX** = 1

static const int **DBOX_YY** = 1

static const int **DBOX_ZZ** = 1

static const int **NBOX_XX** = DBOX_XX * 2 + 1

static const int **NBOX_YY** = DBOX_YY * 2 + 1

static const int **NBOX_ZZ** = DBOX_ZZ * 2 + 1

static const int **shift_info_size** = NBOX_XX * NBOX_YY * NBOX_ZZ

static const int **shift_vec_size** = SPACENDIM * shift_info_size

21.3.3 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`**

21.3.4 Functions

Template Function add_flt_nvnmd

- Defined in file_source_lib_include_env_mat_nvnmd.h

Function Documentation

template<class **`T`**>
void **`add_flt_nvnmd`**(T &y, T x1, T x2)

Function build_nlist(std::vector<std::vector<int»&, std::vector<std::vector<int»&, const std::vector<double>&, const int&, const double&, const double&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const SimulationRegion<double>&, const std::vector<int>&)

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **`build_nlist`**(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const std::vector<double> &coord, const int &nloc, const double &rc0, const double &rc1, const std::vector<int> &nat_stt_, const std::vector<int> &nat_end_, const std::vector<int> &ext_stt_, const std::vector<int> &ext_end_, const SimulationRegion<double> &region, const std::vector<int> &global_grid)

Function build_nlist(std::vector<std::vector<int»&, std::vector<std::vector<int»&, const std::vector<double>&, const double&, const double&, const std::vector<int>&, const SimulationRegion<double>&)

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **build_nlist**(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const std::vector<double> &coord, const double &rc0, const double &rc1, const std::vector<int> &grid, const SimulationRegion<double> &region)

Function build_nlist(std::vector<std::vector<int»&, std::vector<std::vector<int»&, const std::vector<double>&, const std::vector<int>&, const std::vector<int>&, const double&, const double&, const std::vector<int>&, const SimulationRegion<double>&)

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **build_nlist**(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const std::vector<double> &coord, const std::vector<int> &sel0, const std::vector<int> &sel1, const double &rc0, const double &rc1, const std::vector<int> &grid, const SimulationRegion<double> &region)

Function build_nlist(std::vector<std::vector<int»&, std::vector<std::vector<int»&, const std::vector<double>&, const double&, const double&, const SimulationRegion<double> *)

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **build_nlist**(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const std::vector<double> &coord, const double &rc0_, const double &rc1_, const SimulationRegion<double> *region = NULL)

Function compute_descriptor(std::vector<double>&, std::vector<double>&, std::vector<double>&, const std::vector<double>&, const int&, const std::vector<int>&, const SimulationRegion<double>&, const bool&, const int&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const int, const int, const int, const int)

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

inline void **compute_descriptor**(std::vector<double> &descrpt_a, std::vector<double> &descrpt_r, std::vector<double> &rot_mat, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const int &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int> &fmt_nlist_r, const std::vector<int> &sec_a, const std::vector<int> &sec_r, const int axis0_type, const int axis0_idx, const int axis1_type, const int axis1_idx)

Function compute_descriptor(std::vector<double>&, std::vector<double>&, std::vector<double>&, std::vector<double>&, std::vector<double>&, std::vector<double>&, std::vector<double>&, const std::vector<double>&, const int&, const std::vector<int>&, const SimulationRegion<double>&, const bool&, const int&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const int, const int, const int, const int)

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

inline void **compute_descriptor**(std::vector<double> &descrpt_a, std::vector<double> &descrpt_a_deriv, std::vector<double> &descrpt_r, std::vector<double> &descrpt_r_deriv, std::vector<double> &rij_a, std::vector<double> &rij_r, std::vector<double> &rot_mat, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const int &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int> &fmt_nlist_r, const std::vector<int> &sec_a, const std::vector<int> &sec_r, const int axis0_type, const int axis0_idx, const int axis1_type, const int axis1_idx)

Function compute_descriptor_se_a_ef_para

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

inline void **compute_descriptor_se_a_ef_para**(std::vector<double> &descrpt_a, std::vector<double> &descrpt_a_deriv, std::vector<double> &rij_a, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const std::vector<double> &efield, const int &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int> &sec_a, const double &rmin, const double &rmax)

Function compute_descriptor_se_a_ef_vert

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

inline void **compute_descriptor_se_a_ef_vert**(std::vector<double> &descrpt_a, std::vector<double> &descrpt_a_deriv, std::vector<double> &rij_a, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const std::vector<double> &efield, const int &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int> &sec_a, const double &rmin, const double &rmax)

Function compute_descriptor_se_a_extf

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

inline void **compute_descriptor_se_a_extf**(std::vector<double> &descrpt_a, std::vector<double> &descrpt_a_deriv, std::vector<double> &rij_a, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const std::vector<double> &efield, const int &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int> &sec_a, const double &rmin, const double &rmax)

Function compute_dRdT

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "compute_dRdT" with arguments (double (*), const double*, const double*, const double*) in doxygen xml output for project "core" from directory: _build/core/xml/. Potential matches:

```
- void compute_dRdT(double (*dRdT)[9], const double *r1, const double *r2, const double *rot)
```

Function compute_dRdT_1

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "compute_dRdT_1" with arguments (double (*), const double*, const double*, const double*) in doxygen xml output for project "core" from directory: _build/core/xml/. Potential matches:

```
- void compute_dRdT_1(double (*dRdT)[9], const double *r1, const double *r2, const double *rot)
```

Function compute_dRdT_2

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "compute_dRdT_2" with arguments (double (*), const double*, const double*, const double*) in doxygen xml output for project "core" from directory: _build/core/xml/. Potential matches:

```
- void compute_dRdT_2(double (*dRdT)[9], const double *r1, const double *r2, const double *rot)
```

Function copy_coord

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **copy_coord**(std::vector<double> &out_c, std::vector<int> &out_t, std::vector<int> &mapping, std::vector<int> &ncell, std::vector<int> &ngcell, const std::vector<double> &in_c, const std::vector<int> &in_t, const double &rc, const SimulationRegion<double> &region)

Template Function deepmd::build_nlist_cpu

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

template<typename **FPTYPE**>
int **deepmd::build_nlist_cpu**(InputNlist &nlist, int *max_list_size, const FPTYPE *c_cpy, const int &nloc, const int &nall, const int &mem_size, const float &rcut)

Template Function deepmd::build_nlist_gpu

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

template<typename **FPTYPE**>
int **deepmd::build_nlist_gpu**(InputNlist &nlist, int *max_list_size, int *nlist_data, const FPTYPE *c_cpy, const int &nloc, const int &nall, const int &mem_size, const float &rcut)

Template Function deepmd::compute_cell_info

- Defined in file_source_lib_include_coord.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::compute_cell_info**(int *cell_info, const float &rcut, const deepmd::Region<FPTYPE> &region)

Function deepmd::convert_nlist

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **deepmd::convert_nlist**(InputNlist &to_nlist, std::vector<std::vector<int>> &from_nlist)

Construct the InputNlist with a two-dimensional vector.

Parameters

- **to_nlist** – InputNlist struct which stores the neighbor information of the core region atoms.
- **from_nlist** – Vector which stores the neighbor information of the core region atoms.

Function deepmd::convert_nlist_gpu_device

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **deepmd::convert_nlist_gpu_device**(InputNlist &gpu_nlist, InputNlist &cpu_nlist, int *&gpu_memory, const int &max_nbor_size)

Convert the a host memory InputNlist to a device memory InputNlist.

Parameters

- **cpu_nlist** – Host memory InputNlist struct which stores the neighbor information of the core region atoms
- **gpu_nlist** – Device memory InputNlist struct which stores the neighbor information of the core region atoms
- **gpu_memory** – Device array which stores the elements of gpu_nlist
- **max_nbor_size** –

Template Function deepmd::convert_to_inter_cpu

- Defined in file_source_lib_include_region.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::convert_to_inter_cpu**(FPTYPE *ri, const Region<FPTYPE> &region, const FPTYPE *rp)

Template Function deepmd::convert_to_inter_gpu

- Defined in file_source_lib_include_region.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::convert_to_inter_gpu**(FPTYPE *ri, const Region<FPTYPE> &region, const FPTYPE *rp)

Template Function deepmd::convert_to_phys_cpu

- Defined in file_source_lib_include_region.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::convert_to_phys_cpu**(FPTYPE *rp, const Region<FPTYPE> &region, const FPTYPE *ri)

Template Function deepmd::convert_to_phys_gpu

- Defined in file_source_lib_include_region.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::convert_to_phys_gpu**(FPTYPE *rp, const Region<FPTYPE> &region, const FPTYPE *ri)

Template Function deepmd::copy_coord_cpu

- Defined in file_source_lib_include_coord.h

Function Documentation

template<typename **FPTYPE**>
int **deepmd::copy_coord_cpu**(FPTYPE *out_c, int *out_t, int *mapping, int *nall, const FPTYPE *in_c, const int *in_t, const int &nloc, const int &mem_nall, const float &rcut, const deepmd::Region<FPTYPE> &region)

Template Function deepmd::copy_coord_gpu

- Defined in file_source_lib_include_coord.h

Function Documentation

template<typename **FPTYPE**>
int **deepmd::copy_coord_gpu**(FPTYPE *out_c, int *out_t, int *mapping, int *nall, int *int_data, const FPTYPE *in_c, const int *in_t, const int &nloc, const int &mem_nall, const int &loc_cellnum, const int &total_cellnum, const int *cell_info, const deepmd::Region<FPTYPE> &region)

Function deepmd::cos_switch(const double&, const double&, const double&)

- Defined in file_source_lib_include_switcher.h

Function Documentation

inline double `deepmd::cos_switch`(const double &xx, const double &rmin, const double &rmax)

Function deepmd::cos_switch(double&, double&, const double&, const double&, const double&)

- Defined in file_source_lib_include_switcher.h

Function Documentation

inline void `deepmd::cos_switch`(double &vv, double &dd, const double &xx, const double &rmin, const double &rmax)

Template Function deepmd::cprod

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<typename `TYPE`>
inline void `deepmd::cprod`(const TYPE *r0, const TYPE *r1, TYPE *r2)

Function deepmd::cum_sum

- Defined in file_source_lib_include_utilities.h

Function Documentation

void `deepmd::cum_sum`(std::vector<int> &sec, const std::vector<int> &n_sel)

Template Function deepmd::delete_device_memory

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::delete_device_memory**(FPTYPE *&device)

Template Function deepmd::dot1

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<typename **TYPE**>
inline TYPE **deepmd::dot1**(const TYPE *r0, const TYPE *r1)

Template Function deepmd::dot2

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<typename **TYPE**>
inline TYPE **deepmd::dot2**(const TYPE *r0, const TYPE *r1)

Template Function deepmd::dot3

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<typename **TYPE**>
inline TYPE **deepmd::dot3**(const TYPE *r0, const TYPE *r1)

Template Function deepmd::dot4

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<typename **TYPE**>
inline TYPE **deepmd::dot4**(const TYPE *r0, const TYPE *r1)

Template Function deepmd::dotmv3

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<typename **TYPE**>
inline void **deepmd::dotmv3**(TYPE *vec_o, const TYPE *tensor, const TYPE *vec_i)

Function deepmd::DPGetDeviceCount

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

inline void **deepmd::DPGetDeviceCount**(int &gpu_num)

Function deepmd::DPSetDevice

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

inline cudaError_t **deepmd::DPSetDevice**(int rank)

Template Function deepmd::env_mat_a_cpu

- Defined in file_source_lib_include_env_mat.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::env_mat_a_cpu**(std::vector<FPTYPE> &descrpt_a, std::vector<FPTYPE> &descrpt_a_deriv, std::vector<FPTYPE> &rij_a, const std::vector<FPTYPE> &posi, const std::vector<int> &type, const int &i_idx, const std::vector<int> &fmt_nlist, const std::vector<int> &sec, const float &rmin, const float &rmax)

Template Function deepmd::env_mat_a_nvnmd_quantize_cpu

- Defined in file_source_lib_include_env_mat_nvnmd.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::env_mat_a_nvnmd_quantize_cpu**(std::vector<FPTYPE> &descrpt_a, std::vector<FPTYPE> &descrpt_a_deriv, std::vector<FPTYPE> &rij_a, const std::vector<FPTYPE> &posi, const std::vector<int> &type, const int &i_idx, const std::vector<int> &fmt_nlist, const std::vector<int> &sec, const float &rmin, const float &rmax)

Function deepmd::env_mat_nbor_update

- Defined in file_source_lib_include_prod_env_mat.h

Function Documentation

void **deepmd::env_mat_nbor_update**(InputNlist &inlist, InputNlist &gpu_inlist, int &max_nbor_size, int *&nbor_list_dev, const int *mesh, const int size)

Template Function deepmd::env_mat_r_cpu

- Defined in file_source_lib_include_env_mat.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::env_mat_r_cpu**(std::vector<FPTYPE> &descrpt_a, std::vector<FPTYPE> &descrpt_a_deriv, std::vector<FPTYPE> &rij_a, const std::vector<FPTYPE> &posi, const std::vector<int> &type, const int &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int> &sec_a, const float &rmin, const float &rmax)

Template Function deepmd::ewald_recp

- Defined in file_source_lib_include_ewald.h

Function Documentation

template<typename **VALUETYPE**>
void **deepmd::ewald_recp**(VALUETYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<VALUETYPE> &charge, const deepmd::Region<VALUETYPE> &region, const EwaldParameters<VALUETYPE> &param)

Function deepmd::filter_ftype_gpu_cuda

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void **deepmd::filter_ftype_gpu_cuda**(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_cuda

- Defined in file_source_lib_include_fmt_nlist.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::format_nbor_list_gpu_cuda**(int *nlist, const FPTYPE *coord, const int *type, const deepmd::InputNlist &gpu_inlist, int *array_int, uint_64 *array_longlong, const int max_nbor_size, const int nloc, const int nall, const float rcut, const std::vector<int> sec)

Template Function deepmd::format_nlist_cpu

- Defined in file_source_lib_include_fmt_nlist.h

Function Documentation

template<typename **FPTYPE**>
void deepmd::**format_nlist_cpu**(int *nlist, const InputNlist &in_nlist, const FPTYPE *coord, const int *type, const int nloc, const int nall, const float rcut, const std::vector<int> sec)

Function deepmd::free_nlist_gpu_device

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void deepmd::**free_nlist_gpu_device**(InputNlist &gpu_nlist)

Reclaim the allocated device memory of struct InputNlist.

Parameters

gpu_nlist – Device memory InputNlist struct which stores the neighbor information of the core region atoms

Template Function deepmd::gelu_cpu

- Defined in file_source_lib_include_gelu.h

Function Documentation

template<typename **FPTYPE**>
void deepmd::**gelu_cpu**(FPTYPE *out, const FPTYPE *xx, const int_64 size)

Template Function deepmd::gelu_gpu_cuda

- Defined in file_source_lib_include_gelu.h

Function Documentation

template<typename **FPTYPE**>
void deepmd::**gelu_gpu_cuda**(FPTYPE *out, const FPTYPE *xx, const int_64 size)

Template Function deepmd::gelu_grad_cpu

- Defined in file_source_lib_include_gelu.h

Function Documentation

template<typename **FPTYPE**>
void deepmd::**gelu_grad_cpu**(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const int_64 size)

Template Function deepmd::gelu_grad_gpu_cuda

- Defined in file_source_lib_include_gelu.h

Function Documentation

template<typename **FPTYPE**>
void deepmd::**gelu_grad_gpu_cuda**(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const int_64 size)

Template Function deepmd::gelu_grad_grad_cpu

- Defined in file_source_lib_include_gelu.h

Function Documentation

template<typename **FPTYPE**>
void deepmd::**gelu_grad_grad_cpu**(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const FPTYPE *dy_2, const int_64 size)

Template Function deepmd::gelu_grad_grad_gpu_cuda

- Defined in file_source_lib_include_gelu.h

Function Documentation

template<typename **FPTYPE**>
void deepmd::**gelu_grad_grad_gpu_cuda**(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const FPTYPE *dy_2, const int_64 size)

Template Function deepmd::init_region_cpu

- Defined in file_source_lib_include_region.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::init_region_cpu**(Region<FPTYPE> &region, const FPTYPE *boxt)

Template Function deepmd::invsqrt

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<typename **TYPE**>
inline TYPE **deepmd::invsqrt**(const TYPE x)

Specialized Template Function deepmd::invsqrt< double >

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<>
inline double **deepmd::invsqrt**<double>(const double x)

Specialized Template Function deepmd::invsqrt< float >

- Defined in file_source_lib_include_utilities.h

Function Documentation

template<>
inline float **deepmd::invsqrt**<float>(const float x)

Template Function deepmd::malloc_device_memory(FPTYPE *&, const std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::malloc_device_memory**(FPTYPE *&device, const std::vector<FPTYPE> &host)

Template Function deepmd::malloc_device_memory(FPTYPE *&, const int)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::malloc_device_memory**(FPTYPE *&device, const int size)

Template Function deepmd::malloc_device_memory(FPTYPE *&, std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_rocm.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::malloc_device_memory**(FPTYPE *&device, std::vector<FPTYPE> &host)

Template Function deepmd::malloc_device_memory_sync(FPTYPE *&, const std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::malloc_device_memory_sync**(FPTYPE *&device, const std::vector<FPTYPE> &host)

Template Function deepmd::malloc_device_memory_sync(FPTYPE *&, const FPTYPE *, const int)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::malloc_device_memory_sync**(FPTYPE *&device, const FPTYPE *host, const int size)

Template Function deepmd::malloc_device_memory_sync(FPTYPE *&, std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_rocm.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::malloc_device_memory_sync**(FPTYPE *&device, std::vector<FPTYPE> &host)

Template Function deepmd::map_aparam_cpu

- Defined in file_source_lib_include_map_aparam.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::map_aparam_cpu**(FPTYPE *output, const FPTYPE *aparam, const int *nlist, const int &nloc, const int &nnei, const int &numb_aparam)

Function deepmd::max_numneigh

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

int **deepmd::max_numneigh**(const InputNlist &to_nlist)

Compute the max number of neighbors within the core region atoms.

Parameters
: **to_nlist** – InputNlist struct which stores the neighbor information of the core region atoms.

Return values
: **max** – number of neighbors

Returns
: integer

Template Function deepmd::memcpy_device_to_host(const FPTYPE *, std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::memcpy_device_to_host**(const FPTYPE *device, std::vector<FPTYPE> &host)

Template Function deepmd::memcpy_device_to_host(const FPTYPE *, FPTYPE *, const int)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::memcpy_device_to_host**(const FPTYPE *device, FPTYPE *host, const int size)

Template Function deepmd::memcpy_device_to_host(FPTYPE *, std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_rocm.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::memcpy_device_to_host**(FPTYPE *device, std::vector<FPTYPE> &host)

Template Function deepmd::memcpy_host_to_device(FPTYPE *, const std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::memcpy_host_to_device**(FPTYPE *device, const std::vector<FPTYPE> &host)

Template Function deepmd::memcpy_host_to_device(FPTYPE *, const FPTYPE *, const int)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::memcpy_host_to_device**(FPTYPE *device, const FPTYPE *host, const int size)

Template Function deepmd::memcpy_host_to_device(FPTYPE *, std::vector<FPTYPE>&)

- Defined in file_source_lib_include_gpu_rocm.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::memcpy_host_to_device**(FPTYPE *device, std::vector<FPTYPE> &host)

Template Function deepmd::memset_device_memory

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::memset_device_memory**(FPTYPE *device, const int var, const int size)

Template Function deepmd::normalize_coord_cpu

- Defined in file_source_lib_include_coord.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::normalize_coord_cpu**(FPTYPE *coord, const int natom, const deepmd::Region<FPTYPE> &region)

Template Function deepmd::normalize_coord_gpu

- Defined in file_source_lib_include_coord.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::normalize_coord_gpu**(FPTYPE *coord, const int natom, const deepmd::Region<FPTYPE> &region)

Template Function deepmd::pair_tab_cpu

- Defined in file_source_lib_include_pair_tab.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::pair_tab_cpu**(FPTYPE *energy, FPTYPE *force, FPTYPE *virial, const double *table_info, const double *table_data, const FPTYPE *rij, const FPTYPE *scale, const int *type, const int *nlist, const int *natoms, const std::vector<int> &sel_a, const std::vector<int> &sel_r)

Template Function deepmd::prod_env_mat_a_cpu

- Defined in file_source_lib_include_prod_env_mat.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_env_mat_a_cpu**(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const FPTYPE *coord, const int *type, const InputNlist &inlist, const int max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int nloc, const int nall, const float rcut, const float rcut_smth, const std::vector<int> sec, const int *f_type = NULL)

Template Function deepmd::prod_env_mat_a_gpu_cuda

- Defined in file_source_lib_include_prod_env_mat.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_env_mat_a_gpu_cuda**(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const FPTYPE *coord, const int *type, const InputNlist &gpu_inlist, int *array_int, unsigned long long *array_longlong, const int max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int nloc, const int nall, const float rcut, const float rcut_smth, const std::vector<int> sec, const int *f_type = NULL)

Template Function deepmd::prod_env_mat_a_nvnmd_quantize_cpu

- Defined in file_source_lib_include_prod_env_mat_nvnmd.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_env_mat_a_nvnmd_quantize_cpu**(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const FPTYPE *coord, const int *type, const InputNlist &inlist, const int max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int nloc, const int nall, const float rcut, const float rcut_smth, const std::vector<int> sec)

Template Function deepmd::prod_env_mat_r_cpu

- Defined in file_source_lib_include_prod_env_mat.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_env_mat_r_cpu**(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const FPTYPE *coord, const int *type, const InputNlist &inlist, const int max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int nloc, const int nall, const float rcut, const float rcut_smth, const std::vector<int> sec)

Template Function deepmd::prod_env_mat_r_gpu_cuda

- Defined in file_source_lib_include_prod_env_mat.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_env_mat_r_gpu_cuda**(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

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 start_index = 0)

Template Function deepmd::prod_force_a_gpu_cuda

- Defined in file_source_lib_include_prod_force.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_force_a_gpu_cuda**(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE *in_deriv, const int *nlist, const int nloc, const int nall, const int nnei)

Template Function deepmd::prod_force_grad_a_cpu

- Defined in file_source_lib_include_prod_force_grad.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_force_grad_a_cpu**(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const int *nlist, const int nloc, const int nnei)

Template Function deepmd::prod_force_grad_a_gpu_cuda

- Defined in file_source_lib_include_prod_force_grad.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_force_grad_a_gpu_cuda**(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const int *nlist, const int nloc, const int nnei)

Template Function deepmd::prod_force_grad_r_cpu

- Defined in file_source_lib_include_prod_force_grad.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_force_grad_r_cpu**(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const int *nlist, const int nloc, const int nnei)

Template Function deepmd::prod_force_grad_r_gpu_cuda

- Defined in file_source_lib_include_prod_force_grad.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_force_grad_r_gpu_cuda**(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const int *nlist, const int nloc, const int nnei)

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)

Template Function deepmd::prod_force_r_gpu_cuda

- Defined in file_source_lib_include_prod_force.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_force_r_gpu_cuda**(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE *in_deriv, const int *nlist, const int nloc, const int nall, const int nnei)

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_cuda

- Defined in file_source_lib_include_prod_virial.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_virial_a_gpu_cuda**(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int nnei)

Template Function deepmd::prod_virial_grad_a_cpu

- Defined in file_source_lib_include_prod_virial_grad.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_virial_grad_a_cpu**(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nnei)

Template Function deepmd::prod_virial_grad_a_gpu_cuda

- Defined in file_source_lib_include_prod_virial_grad.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_virial_grad_a_gpu_cuda**(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nnei)

Template Function deepmd::prod_virial_grad_r_cpu

- Defined in file_source_lib_include_prod_virial_grad.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_virial_grad_r_cpu**(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nnei)

Template Function deepmd::prod_virial_grad_r_gpu_cuda

- Defined in file_source_lib_include_prod_virial_grad.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_virial_grad_r_gpu_cuda**(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nnei)

Template Function deepmd::prod_virial_r_cpu

- Defined in file_source_lib_include_prod_virial.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_virial_r_cpu**(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int nnei)

Template Function deepmd::prod_virial_r_gpu_cuda

- Defined in file_source_lib_include_prod_virial.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::prod_virial_r_gpu_cuda**(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int nnei)

Template Function deepmd::soft_min_switch_cpu

- Defined in file_source_lib_include_soft_min_switch.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::soft_min_switch_cpu**(FPTYPE *sw_value, FPTYPE *sw_deriv, const FPTYPE *rij, const int *nlist, const int &nloc, const int &nnei, const FPTYPE &alpha, const FPTYPE &rmin, const FPTYPE &rmax)

Template Function deepmd::soft_min_switch_force_cpu

- Defined in file_source_lib_include_soft_min_switch_force.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::soft_min_switch_force_cpu**(FPTYPE *force, const FPTYPE *du, const FPTYPE *sw_deriv, const int *nlist, const int nloc, const int nall, const int nnei)

Template Function deepmd::soft_min_switch_force_grad_cpu

- Defined in file_source_lib_include_soft_min_switch_force_grad.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::soft_min_switch_force_grad_cpu**(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *sw_deriv, const int *nlist, const int nloc, const int nnei)

Template Function deepmd::soft_min_switch_virial_cpu

- Defined in file_source_lib_include_soft_min_switch_virial.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::soft_min_switch_virial_cpu**(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *du, const FPTYPE *sw_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int nnei)

Template Function deepmd::soft_min_switch_virial_grad_cpu

- Defined in file_source_lib_include_soft_min_switch_virial_grad.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::soft_min_switch_virial_grad_cpu**(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE *sw_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nnei)

Function deepmd::spline3_switch

- Defined in file_source_lib_include_switcher.h

Function Documentation

inline void **deepmd::spline3_switch**(double &vv, double &dd, const double &xx, const double &rmin, const double &rmax)

Template Function deepmd::spline5_switch

- Defined in file_source_lib_include_switcher.h

Function Documentation

template<typename **FPTYPE**>
inline void **deepmd::spline5_switch**(FPTYPE &vv, FPTYPE &dd, const FPTYPE &xx, const float &rmin, const float &rmax)

Template Function deepmd::tabulate_fusion_se_a_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_a_cpu**(FPTYPE *out, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const int nloc, const int nnei, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_a_gpu_cuda

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_a_gpu_cuda**(FPTYPE *out, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const int nloc, const int nnei, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_a_grad_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_a_grad_cpu**(FPTYPE *dy_dem_x, FPTYPE *dy_dem, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *dy, const int nloc, const int nnei, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_a_grad_gpu_cuda

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_a_grad_gpu_cuda**(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, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_a_grad_grad_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_a_grad_grad_cpu**(FPTYPE *dz_dy, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *dz_dy_dem_x, const FPTYPE *dz_dy_dem, const int nloc, const int nnei, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_a_grad_grad_gpu_cuda

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_a_grad_grad_gpu_cuda**(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, const int last_layer_size)

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_cuda

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_r_gpu_cuda**(FPTYPE *out, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em, const int nloc, const int nnei, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_r_grad_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_r_grad_cpu**(FPTYPE *dy_dem, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em, const FPTYPE *dy, const int nloc, const int nnei, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_r_grad_gpu_cuda

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_r_grad_gpu_cuda**(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_cuda

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_r_grad_grad_gpu_cuda**(FPTYPE *dz_dy, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em, const FPTYPE *dz_dy_dem, const int nloc, const int nnei, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_t_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_t_cpu**(FPTYPE *out, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const int nloc, const int nnei_i, const int nnei_j, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_t_gpu_cuda

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_t_gpu_cuda**(FPTYPE *out, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const int nloc, const int nnei_i, const int nnei_j, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_t_grad_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_t_grad_cpu**(FPTYPE *dy_dem_x, FPTYPE *dy_dem, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *dy, const int nloc, const int nnei_i, const int nnei_j, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_t_grad_gpu_cuda

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_t_grad_gpu_cuda**(FPTYPE *dy_dem_x, FPTYPE *dy_dem, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *dy, const int nloc, const int nnei_i, const int nnei_j, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_t_grad_grad_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_t_grad_grad_cpu**(FPTYPE *dz_dy, const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *em_x, const FPTYPE *em, const FPTYPE *dz_dy_dem_x, const FPTYPE *dz_dy_dem, const int nloc, const int nnei_i, const int nnei_j, const int last_layer_size)

Template Function deepmd::tabulate_fusion_se_t_grad_grad_gpu_cuda

- Defined in file_source_lib_include_tabulate.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::tabulate_fusion_se_t_grad_grad_gpu_cuda**(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_cuda

- Defined in file_source_lib_include_fmt_nlist.h

Function Documentation

template<typename **FPTYPE**>
void deepmd::**test_encoding_decoding_nbor_info_gpu_cuda**(uint_64 *key, int *out_type, int *out_index, const int *in_type, const FPTYPE *in_dist, const int *in_index, const int size_of_array)

Function deepmd::use_nei_info_cpu

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void deepmd::**use_nei_info_cpu**(int *nlist, int *ntype, bool *nmask, const int *type, const int *nlist_map, const int nloc, const int nnei, const int ntypes, const bool b_nlist_map)

Function deepmd::use_nei_info_gpu

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void deepmd::**use_nei_info_gpu**(int *nlist, int *ntype, bool *nmask, const int *type, const int *nlist_map, const int nloc, const int nnei, const int ntypes, const bool b_nlist_map)

Function deepmd::use_nlist_map

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

void deepmd::**use_nlist_map**(int *nlist, const int *nlist_map, const int nloc, const int nnei)

Template Function deepmd::volume_cpu

- Defined in file_source_lib_include_region.h

Function Documentation

template<typename **FPTYPE**>
FPTYPE **deepmd::volume_cpu**(const Region<FPTYPE> &region)

Template Function deepmd::volume_gpu

- Defined in file_source_lib_include_region.h

Function Documentation

template<typename **FPTYPE**>
void **deepmd::volume_gpu**(FPTYPE *volume, const Region<FPTYPE> &region)

Template Function dotmul_flt_nvnmd

- Defined in file_source_lib_include_env_mat_nvnmd.h

Function Documentation

template<class **T**>
void **dotmul_flt_nvnmd**(T &y, T *x1, T *x2, int64_t M)

Function DPAssert(cudaError_t, const char *, int, bool)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

inline void **DPAssert**(cudaError_t code, const char *file, int line, bool abort = true)

Function DPAssert(hipError_t, const char *, int, bool)

- Defined in file_source_lib_include_gpu_rocm.h

Function Documentation

inline void **DPAssert**(hipError_t code, const char *file, int line, bool abort = true)

Function env_mat_a

- Defined in file_source_lib_include_env_mat.h

Function Documentation

void **env_mat_a**(std::vector<double> &descrpt_a, std::vector<double> &descrpt_a_deriv, std::vector<double> &rij_a, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const int &i_idx, const std::vector<int> &fmt_nlist, const std::vector<int> &sec, const double &rmin, const double &rmax)

Function env_mat_r

- Defined in file_source_lib_include_env_mat.h

Function Documentation

void **env_mat_r**(std::vector<double> &descrpt_r, std::vector<double> &descrpt_r_deriv, std::vector<double> &rij_r, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const int &i_idx, const std::vector<int> &fmt_nlist, const std::vector<int> &sec, const double &rmin, const double &rmax)

Template Function find_max_expo(int64_t&, T *, int64_t)

- Defined in file_source_lib_include_env_mat_nvnmd.h

Function Documentation

template<class **T**>
void **find_max_expo**(int64_t &max_expo, T *x, int64_t M)

Template Function find_max_expo(int64_t&, T *, int64_t, int64_t)

- Defined in file_source_lib_include_env_mat_nvnmd.h

Function Documentation

template<class **T**>
void **find_max_expo**(int64_t &max_expo, T *x, int64_t N, int64_t M)

Template Function format_nlist_i_cpu

- Defined in file_source_lib_include_fmt_nlist.h

Function Documentation

template<typename **FPTYPE**>
int **format_nlist_i_cpu**(std::vector<int> &fmt_nei_idx_a, const std::vector<FPTYPE> &posi, const std::vector<int> &type, const int &i_idx, const std::vector<int> &nei_idx_a, const float &rcut, const std::vector<int> &sec_a)

Function format_nlist_i_fill_a

- Defined in file_source_lib_include_fmt_nlist.h

Function Documentation

int **format_nlist_i_fill_a**(std::vector<int> &fmt_nei_idx_a, std::vector<int> &fmt_nei_idx_r, const std::vector<double> &posi, const int &ntypes, const std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc, const int &i_idx, const std::vector<int> &nei_idx_a, const std::vector<int> &nei_idx_r, const double &rcut, const std::vector<int> &sec_a, const std::vector<int> &sec_r)

Template Function mul_flt_nvnmd

- Defined in file_source_lib_include_env_mat_nvnmd.h

Function Documentation

template<class **T**>
void **mul_flt_nvnmd**(T &y, T x1, T x2)

Function nborAssert(cudaError_t, const char *, int, bool)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

inline void **`nborAssert`**(cudaError_t code, const char *file, int line, bool abort = true)

Function nborAssert(hipError_t, const char *, int, bool)

- Defined in file_source_lib_include_gpu_rocm.h

Function Documentation

inline void **`nborAssert`**(hipError_t code, const char *file, int line, bool abort = true)

Function omp_get_num_threads

- Defined in file_source_lib_include_ewald.h

Function Documentation

int **`omp_get_num_threads`**()

Function omp_get_thread_num

- Defined in file_source_lib_include_ewald.h

Function Documentation

int **`omp_get_thread_num`**()

Template Function split_flt

- Defined in file_source_lib_include_env_mat_nvnmd.h

Function Documentation

template<class **T**>
void **split_flt**(T x, int64_t &sign, int64_t &expo, int64_t &mant)

21.3.5 Variables

Variable deepmd::ElectrostaticConvertion

- Defined in file_source_lib_include_ewald.h

Variable Documentation

const double deepmd::**ElectrostaticConvertion** = 14.39964535475696995031

21.3.6 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 MOASPNDIM

- Defined in file_source_lib_include_SimulationRegion.h

Define Documentation

`MOASPNDIM`

Define NBIT_CUTF

- Defined in file_source_lib_include_env_mat_nvnmd.h

Define Documentation

`NBIT_CUTF`

Define NBIT_FLTF

- Defined in file_source_lib_include_env_mat_nvnmd.h

Define Documentation

NBIT_FLTF

Define nborErrcheck

- Defined in file_source_lib_include_gpu_cuda.h

Define Documentation

nborErrcheck(res)

Define nborErrcheck

- Defined in file_source_lib_include_gpu_rocm.h

Define Documentation

nborErrcheck(res)

Define SQRT_2_PI

- Defined in file_source_lib_include_device.h

Define Documentation

SQRT_2_PI

Define TPB

- Defined in file_source_lib_include_device.h

Define Documentation

`TPB`

21.3.7 Typedefs

Typedef int_64

- Defined in file_source_lib_include_device.h

Typedef Documentation

typedef long long **`int_64`**

Typedef uint_64

- Defined in file_source_lib_include_device.h

Typedef Documentation

typedef unsigned long long **`uint_64`**

CHAPTER TWENTYTWO

LICENSE

The project DeePMD-kit is licensed under GNU LGPLv3.0.

CHAPTER

TWENTYTHREE

AUTHORS AND CREDITS

23.1 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) 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,

23.2 Package Contributors

- AngelJia
- AnguseZhang
- Anurag Kumar Singh
- Chenxing Luo
- Chun Cai
- Davide Tisi

- Denghui Lu
- Duo
- Eisuke Kawashima
- GeiduanLiu
- Han Wang
- Harvey Que
- HuangJiameng
- Jia-Xin Zhu
- Jiequn Han
- Jingchao Zhang
- Jinzhe Zeng
- Koki MURAOKA
- LiangWenshuo1118
- Linfeng Zhang
- LiuGroupHNU
- Lu
- Marián Rynik
- Nick Lin
- Rhys Goodall
- Shaochen Shi
- TrellixVulnTeam
- 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
- marian-code
- mingzhong15
- njzjz
- pkulzy
- pre-commit-ci[bot]
- readthedocs-assistant
- sigbjobo
- tuoping
- wsyxbcl
- ziyao

23.3 Other Credits

- Zhang ZiXuan for designing the Deepmodeling logo.
- Everyone on the Deepmodeling mailing list for contributing to many discussions and decisions!

CHAPTER

TWENTYFOUR

LOGO

The logo of DeePMD-kit is a beaver. Beavers were widely distributed in Europe and Asia but became nearly extinct due to hunting. Listed as a first-class state-protected animal in China, the population of beavers in China is less than the giant pandas. We hope that users of DeePMD-kit can enhance the awareness to protect beavers.

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] Jinzhe Zeng, Timothy J. Giese,¸Sölen Ekesan, Darrin M. York, Development of Range-Corrected Deep Learning Potentials for Fast, Accurate Quantum Mechanical/molecular Mechanical Simulations of Chemical Reactions in Solution, J. Chem. Theory Comput., 2021, 17 (11), 6993-7009.

[1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 4441–4451.

[1] Jinzhe Zeng, Timothy J. Giese,¸Sölen Ekesan, Darrin M. York, Development of Range-Corrected Deep Learning Potentials for Fast, Accurate Quantum Mechanical/molecular Mechanical Simulations of Chemical Reactions in Solution, J. Chem. Theory Comput., 2021, 17 (11), 6993-7009.

[1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 4441–4451.

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identitymappings in deep residual networks. InComputer Vision – ECCV 2016,pages 630–645. Springer International Publishing, 2016.

[WZHE18] Han Wang, Linfeng Zhang, Jiequn Han, and Weinan E. DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics. Comput. Phys. Comm., 228:178–184, jul 2018. doi:10.1016/j.cpc.2018.03.016.

[ZZL+23] Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li′ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, and Han Wang. DeePMD-kit v2: A software package for Deep Potential models. 2023. doi:10.48550/arXiv.2304.09409.

[LWC+21] Denghui Lu, Han Wang, Mohan Chen, Lin Lin, Roberto Car, Weinan E, Weile Jia, and Linfeng Zhang. 86 PFLOPS Deep Potential Molecular Dynamics simulation of 100 million atoms with ab initio accuracy. Comput. Phys. Comm., 259:107624, 2021. doi:10.1016/j.cpc.2020.107624.

[ZHW+18] Linfeng Zhang, Jiequn Han, Han Wang, Roberto Car, and Weinan E. Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics. Phys. Rev. Lett., 120(14):143001, 2018. doi:10.1103/PhysRevLett.120.143001.

[ZHW+18] Linfeng Zhang, Jiequn Han, Han Wang, Wissam Saidi, Roberto Car, and Weinan E. End-to-end Symmetry Preserving Inter-atomic Potential Energy Model for Finite and Extended Systems. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems 31, pages 4436–4446. Curran Associates, Inc., 2018. URL: https://dl.acm.org/doi/10.5555/3327345.3327356.

[WWZ+22] Xiaoyang Wang, Yinan Wang, Linfeng Zhang, Fuzhi Dai, and Han Wang. A tungsten deep neural-network potential for simulating mechanical property degradation under fusion service environment. Nucl. Fusion, 62:126013, 2022. doi:10.1088/1741-4326/ac888b.

[ZBD+22] Duo Zhang, Hangrui Bi, Fu-Zhi Dai, Wanrun Jiang, Linfeng Zhang, and Han Wang. DPA-1: Pretraining of Attention-based Deep Potential Model for Molecular Simulation. 2022. doi:10.48550/arXiv.2208.08236.

[ZCW+20] Linfeng Zhang, Mohan Chen, Xifan Wu, Han Wang, Weinan E, and Roberto Car. Deep neural network for the dielectric response of insulators. Phys. Rev. B, 102(4):041121, 2020. doi:10.1103/PhysRevB.102.041121.

[SAZ+20] Grace M Sommers, Marcos F Calegari Andrade, Linfeng Zhang, Han Wang, and Roberto Car. Raman spectrum and polarizability of liquid water from deep neural networks. Phys. Chem. Chem. Phys., 22(19):10592–10602, 2020. doi:10.1039/D0CP01893G.

[ZCY+22] Qiyu Zeng, Bo Chen, Xiaoxiang Yu, Shen Zhang, Dongdong Kang, Han Wang, and Jiayu Dai. Towards large-scale and spatiotemporally resolved diagnosis of electronic density of states by deep learning. Phys. Rev. B, 105:174109, 2022. doi:10.1103/PhysRevB.105.174109.

[ZTGY23] Jinzhe Zeng, Yujun Tao, Timothy J Giese, and Darrin M York. QD\pi : A Quantum Deep Potential Interaction Model for Drug Discovery. J. Chem. Theory Comput., 19:1261–1275, 2023. doi:10.1021/acs.jctc.2c01172.

[ZWM+22] Linfeng Zhang, Han Wang, Maria Carolina Muniz, Athanassios Z Panagiotopoulos, Roberto Car, and Weinan E. A deep potential model with long-range electrostatic interactions. J. Chem. Phys., 156:124107, 2022. doi:10.1063/5.0083669.

[ZGEY21] Jinzhe Zeng, Timothy J Giese, Şölen Ekesan, and Darrin M York. Development of Range-Corrected Deep Learning Potentials for Fast, Accurate Quantum Mechanical/molecular Mechanical Simulations of Chemical Reactions in Solution. J. Chem. Theory Comput., 17:6993–7009, 2021. doi:10.1021/acs.jctc.1c00201.

[WGZ+19] Hao Wang, Xun Guo, Linfeng Zhang, Han Wang, and Jianming Xue. Deep learning inter-atomic potential model for accurate irradiation damage simulations. Appl. Phys. Lett., 114(24):244101, 2019. doi:10.1063/1.5098061.

[LJC+22] Denghui Lu, Wanrun Jiang, Yixiao Chen, Linfeng Zhang, Weile Jia, Han Wang, and Mohan Chen. DP Compress: A Model Compression Scheme for Generating Efficient Deep Potential Models. J. Chem. Theory Comput., 18:5555–5567, 2022. doi:10.1021/acs.jctc.2c00102.

[ZLW+19] Linfeng Zhang, De-Ye Lin, Han Wang, Roberto Car, and Weinan E. Active learning of uniformly accurate interatomic potentials for materials simulation. Phys. Rev. Mater., 3:23804, 2019. doi:10.1103/PhysRevMaterials.3.023804.

[ZZWZ21] Jinzhe Zeng, Linfeng Zhang, Han Wang, and Tong Zhu. Exploring the Chemical Space of Linear Alkane Pyrolysis via Deep Potential GENerator. Energy & Fuels, 35(1):762–769, 2021. doi:10.1021/acs.energyfuels.0c03211.

[MLZ+22] Pinghui Mo, Chang Li, Dan Zhao, Yujia Zhang, Mengchao Shi, Junhua Li, and Jie Liu. Accurate and efficient molecular dynamics based on machine learning and non von Neumann architecture. npj Comput. Mater., 8:107, 2022. doi:10.1038/s41524-022-00773-z.

PYTHON MODULE INDEX

INDEX