

---

# DeePMD-kit

DeepModeling

Jul 01, 2022



# GETTING STARTED

1	Getting Started	3
1.1	Easy install	3
1.1.1	Install off-line packages	3
1.1.2	Install with conda	4
1.1.3	Install with docker	4
1.2	Prepare data with dpdata	4
1.3	Train a model	5
1.3.1	Warning	6
1.4	Freeze a model	6
1.5	Test a model	7
1.6	Run MD with LAMMPS	7
2	Installation	9
2.1	Easy install	9
2.1.1	Install off-line packages	9
2.1.2	Install with conda	9
2.1.3	Install with docker	10
2.2	Install from source code	10
2.2.1	Install the python interface	10
2.2.2	Install the C++ interface	13
2.3	Install LAMMPS	15
2.3.1	Install LAMMPS's DeePMD-kit module (built-in mode)	15
2.3.2	Install LAMMPS (plugin mode)	16
2.4	Install i-PI	16
2.5	Install GROMACS with DeepMD	17
2.5.1	Patch source code of GROMACS	17
2.5.2	Compile GROMACS with deepmd-kit	17
2.6	Building conda packages	17
3	Data	19
3.1	System	19
3.2	Formats of a system	21
3.2.1	NumPy format	21
3.2.2	HDF5 format	21
3.2.3	Raw format and data conversion	22
3.3	Prepare data with dpdata	22
4	Model	25
4.1	Overall	25
4.2	Descriptor "se_e2_a"	26

4.3	Descriptor "se_e2_r" . . . . .	27
4.4	Descriptor "se_e3" . . . . .	27
4.5	Descriptor "hybrid" . . . . .	28
4.6	Determine sel . . . . .	28
4.7	Fit energy . . . . .	29
4.7.1	The fitting network . . . . .	29
4.7.2	Loss . . . . .	29
4.8	Fit tensor like Dipole and Polarizability . . . . .	30
4.8.1	The fitting Network . . . . .	30
4.8.2	Loss . . . . .	31
4.8.3	Training Data Preparation . . . . .	31
4.8.4	Train the Model . . . . .	31
4.9	Type embedding approach . . . . .	32
4.9.1	Type embedding net . . . . .	32
4.10	Deep potential long-range (DPLR) . . . . .	33
4.10.1	Train a deep Wannier model for Wannier centroids . . . . .	33
4.10.2	Train the DPLR model . . . . .	34
4.10.3	Molecular dynamics simulation with DPLR . . . . .	35
4.11	Deep Potential - Range Correction (DPRc) . . . . .	37
4.11.1	Training data . . . . .	37
4.11.2	Training the DPRc model . . . . .	37
4.11.3	Run MD simulations . . . . .	38
5	Training . . . . .	39
5.1	Train a model . . . . .	39
5.1.1	Warning . . . . .	40
5.2	Advanced options . . . . .	40
5.2.1	Learning rate . . . . .	40
5.2.2	Training parameters . . . . .	41
5.2.3	Options and environment variables . . . . .	43
5.2.4	Adjust sel of a frozen model . . . . .	44
5.3	Training Parameters . . . . .	44
5.4	Parallel training . . . . .	68
5.4.1	Tuning learning rate . . . . .	68
5.4.2	Scaling test . . . . .	68
5.4.3	How to use . . . . .	68
5.4.4	Logging . . . . .	69
5.5	TensorBoard Usage . . . . .	69
5.5.1	Highlighted features . . . . .	70
5.5.2	How to use Tensorboard with DeePMD-kit . . . . .	70
5.5.3	Examples . . . . .	71
5.5.4	Attention . . . . .	75
5.6	Known limitations of using GPUs . . . . .	75
6	Freeze and Compress . . . . .	77
6.1	Freeze a model . . . . .	77
6.2	Compress a model . . . . .	77
7	Test . . . . .	81
7.1	Test a model . . . . .	81
7.2	Calculate Model Deviation . . . . .	82
8	Inference . . . . .	83
8.1	Python interface . . . . .	83
8.2	C++ interface . . . . .	83

9	Command line interface	85
9.1	Named Arguments	85
9.2	Valid subcommands	85
9.3	Sub-commands:	85
9.3.1	config	85
9.3.2	transfer	86
9.3.3	train	86
9.3.4	freeze	87
9.3.5	test	88
9.3.6	compress	89
9.3.7	doc-train-input	90
9.3.8	model-devi	90
9.3.9	convert-from	91
9.3.10	neighbor-stat	91
9.3.11	train-nvnmd	92
10	Integrate with third-party packages	93
10.1	Use deep potential with ASE	93
10.2	Run MD with LAMMPS	93
10.3	LAMMPS commands	94
10.3.1	Enable DeePMD-kit plugin (plugin mode)	94
10.3.2	pair_style deepmd	94
10.3.3	Compute tensorial properties	95
10.3.4	Long-range interaction	95
10.3.5	Use of the centroid/stress/atom to get the full 3x3 “atomic-virial”	96
10.3.6	Computation of heat flux	96
10.4	Run path-integral MD with i-PI	97
10.5	Running MD with GROMACS	97
10.5.1	DP/MM Simulation	97
10.5.2	All-atom DP Simulation	100
10.6	Interfaces out of DeePMD-kit	101
10.6.1	dpdata	101
10.6.2	OpenMM plugin for DeePMD-kit	101
10.6.3	AMBER interface to DeePMD-kit	101
10.6.4	DP-GEN	101
10.6.5	MLatom	101
11	Use NVNMD	103
11.1	Introduction	103
11.2	Training	103
11.2.1	Input script	104
11.2.2	Training	106
11.3	Testing	106
11.4	Running MD	106
11.4.1	Account application	107
11.4.2	Adding task	107
11.4.3	Cancelling calculation	109
11.4.4	Downloading results	109
11.4.5	Deleting record	110
11.4.6	Clearing records	110
12	FAQs	111
12.1	How to tune Fitting/embedding-net size ?	111
12.1.1	Al2O3	111

12.1.2	Cu . . . . .	112
12.1.3	Water . . . . .	113
12.1.4	Mg-Al . . . . .	114
12.2	How to control the number of nodes used by a job ? . . . . .	114
12.3	Do we need to set rcut < half boxsize ? . . . . .	115
12.4	How to set sel ? . . . . .	115
12.5	Installation . . . . .	115
12.5.1	Inadequate versions of gcc/g++ . . . . .	115
12.5.2	Build files left in DeePMD-kit . . . . .	115
12.6	The temperature undulates violently during early stages of MD . . . . .	116
12.7	MD: cannot run LAMMPS after installing a new version of DeePMD-kit . . . . .	116
12.8	Model compatibility . . . . .	116
13	Coding Conventions . . . . .	117
13.1	Preface . . . . .	117
13.2	Rules . . . . .	117
13.3	Whitespace . . . . .	118
13.4	General advice . . . . .	118
13.5	Writing documentation in the code . . . . .	118
13.6	Run pycodestyle on your code . . . . .	119
13.7	Run mypy on your code . . . . .	119
13.8	Run pydocstyle on your code . . . . .	119
13.9	Run black on your code . . . . .	119
14	Create a model . . . . .	121
14.1	Design a new component . . . . .	121
14.2	Register new arguments . . . . .	121
14.3	Package new codes . . . . .	122
15	Atom Type Embedding . . . . .	123
15.1	Overview . . . . .	123
15.2	Preliminary . . . . .	123
15.3	How to use . . . . .	124
15.4	Code Modification . . . . .	124
15.4.1	trainer (train/trainer.py) . . . . .	124
15.4.2	model (model/ener.py) . . . . .	124
15.4.3	embedding net (descriptor/se*.py) . . . . .	124
15.4.4	fitting net (fit/ener.py) . . . . .	125
16	Python API . . . . .	127
16.1	deepmd package . . . . .	127
16.1.1	Subpackages . . . . .	130
16.1.2	Submodules . . . . .	265
16.1.3	deepmd.calculator module . . . . .	265
16.1.4	deepmd.common module . . . . .	268
16.1.5	deepmd.env module . . . . .	273
17	C++ API . . . . .	275
17.1	Class Hierarchy . . . . .	275
17.2	File Hierarchy . . . . .	275
17.3	Full API . . . . .	276
17.3.1	Namespaces . . . . .	276
17.3.2	Classes and Structs . . . . .	277
17.3.3	Functions . . . . .	291
17.3.4	Typedefs . . . . .	296

18 License	299
19 Authors and Credits	301
19.1 Package Contributors . . . . .	301
19.2 Other Credits . . . . .	302
Bibliography	303
Python Module Index	305
Index	307





DeePMD-kit is a package written in Python/C++, designed to minimize the effort required to build deep learning based model of interatomic potential energy and force field and to perform molecular dynamics (MD). 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 this using 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.

---



## 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: 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](#)

#### 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
```

### 1.1.2 Install with conda

DeePMD-kit is available with [conda](#). Install [Anaconda](#) or [Miniconda](#) first.

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
```

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
```

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
```

One may enable the environment using

```
conda activate deepmd
```

### 1.1.3 Install with docker

A docker for installing the DeePMD-kit is available [here](#).

To pull the CPU version:

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

## 1.2 Prepare data with dpdata

One can use the 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/npz', '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 at the [official website](#).

## 1.3 Train a model

Several examples of training can be found at 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 worth special notice.

```
DEEPMD INFO  ---Summary of DataSystem: training  -----
↪ --
DEEPMD INFO  found 3 system(s):
DEEPMD INFO
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
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 by three data systems, while the validation data set is composed by one data system. The number of atoms, batch size, 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 `1curve.out`). The batch size can be set in the input script by the key `batch_size` in the corresponding sections for 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, 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. One can visualize this file by 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 prefix `save_ckpt` every `save_freq` training steps.

### 1.3.1 Warning

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

## 1.4 Freeze a model

The trained neural network is extracted from a checkpoint and dumped into a database. 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 database is called `graph.pb`.

## 1.5 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 file containing details of energy force and virial
                        accuracy
```

## 1.6 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     * *
```

where `graph.pb` is the file name of the frozen model. It should be noted that LAMMPS counts atom types starting from 1, therefore, all LAMMPS atom type will be firstly subtracted by 1, and then passed into the DeePMD-kit engine to compute the interactions.





## 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: 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](#)

#### 2.1.1 Install off-line packages

Both CPU and GPU version offline packages are available in [the Releases page](#).

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

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

#### 2.1.2 Install with conda

DeePMD-kit is available with [conda](#). Install [Anaconda](#) or [Miniconda](#) first.

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
```

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
```

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 lammmps horovod -c https://conda.
↳ deepmodeling.com
```

One may enable the environment using

```
conda activate deepmd
```

### 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
```

## 2.2 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 deepmd-kit
```

The `--recursive` option clones all [submodules](#) needed by DeePMD-kit.

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

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

### 2.2.1 Install the python interface

#### Install the 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 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 everytime 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 like python3.x, it can be specified by, for example

```
virtualenv -p python3.7 $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 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 requirement of 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.

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

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

then execute

```
dp -h
```

It will print the help information like

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

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

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

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

## Install horovod and mpi4py

Horovod and mpi4py is used for parallel training. For better performance on GPU, please follow 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 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:
```

(continues on next page)

(continued from previous page)

```

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

Available Controllers:
  [X] MPI
  [X] Gloo

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

```

From version 2.0.1, Horovod and mpi4py with MPICH support is 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 Lammmps or I-Pi, then the python interface installed in the previous section does everything and he/she can safely skip this section.

### Install the Tensorflow's C++ interface

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 requirement of 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 the DeePMD-kit's C++ interface

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

```

cd $deepmd_source_dir/source
mkdir build
cd build

```

I assume you want to install DeePMD-kit into path `$deepmd_root`, then execute cmake

```

cmake -DTENSORFLOW_ROOT=$tensorflow_root -DCMAKE_INSTALL_PREFIX=$deepmd_root ..

```

where the variable `tensorflow_root` stores the location where the TensorFlow's C++ interface is installed.

One may add the following arguments to cmake:

CMake Arguments	Allowed value	Default value	Usage
- <code>DTENSORFLOW_ROOT=</code> <code>&lt;value&gt;</code>	Path	-	The Path to TensorFlow's C++ interface.
- <code>DCMAKE_INSTALL_PREFIX=</code> <code>&lt;value&gt;</code>	Path	-	The Path where DeePMD-kit will be installed.
- <code>DUSE_CUDA_TOOLKIT=</code> <code>&lt;value&gt;</code>	TRUE FALSE	FALSE	If TRUE, Build GPU support with CUDA toolkit.
- <code>DCUDA_TOOLKIT_ROOT=</code> <code>&lt;value&gt;</code>	Path	De- tected auto- mati- cally	The path to the CUDA toolkit directory. CUDA 7.0 or later is supported. NVCC is required.
- <code>DUSE_ROCM_TOOLKIT=</code> <code>&lt;value&gt;</code>	TRUE FALSE	FALSE	If TRUE, Build GPU support with ROCM toolkit.
- <code>DROCM_ROOT=</code> <code>&lt;value&gt;</code>	Path	De- tected auto- mati- cally	The path to the ROCM toolkit directory.
- <code>DLAMMPS_VERSION_OR_NUMBER=</code> <code>&lt;value&gt;</code>	Num- ber	20220723	Only necessary for LAMMPS built-in mode. The version number of LAMMPS (yyyymmdd). LAMMPS 29Oct2020 (20201029) or later is supported.
- <code>DLAMMPS_SOURCE_ROOT=</code> <code>&lt;value&gt;</code>	Path	-	Only necessary for LAMMPS plugin mode. The path to the LAMMPS source code. LAMMPS 8Apr2021 or later is supported. If not assigned, the plugin mode will not be enabled.

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

```
make -j4
make install
```

The 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 following executable and libraries installed in `$deepmd_root/bin` and `$deepmd_root/lib`

```
$ ls $deepmd_root/bin
dp_ipi      dp_ipi_low
$ ls $deepmd_root/lib
libdeepmd_cc_low.so  libdeepmd_ipi_low.so  libdeepmd_lmp_low.so  libdeepmd_low.so
↳ libdeepmd_op_cuda.so  libdeepmd_op.so
libdeepmd_cc.so      libdeepmd_ipi.so      libdeepmd_lmp.so      libdeepmd_op_cuda_low.so
↳ libdeepmd_op_low.so  libdeepmd.so
```

## 2.3 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.

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

Before following this section, [DeePMD-kit C++ interface](#) should have been installed.

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

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

DeePMD-kit will generate a module called `USER-DEEPM` in the build directory. If you need the low precision version, move `env_low.sh` to `env.sh` in the directory. Now download the LAMMPS code, and uncompress it. The LAMMPS version should be the same as what is specified as the CMAKE argument `LAMMPS_VERSION_NUMBER`.

```
cd /some/workspace
wget https://github.com/lammps/lammps/archive/stable_23Jun2022.tar.gz
tar xf stable_23Jun2022.tar.gz
```

The source code of LAMMPS is stored in directory `lammps-stable_23Jun2022`. Now go into the LAMMPS code and copy the DeePMD-kit module like this

```
cd lammps-stable_23Jun2022/src/
cp -r $deepmd_source_dir/source/build/USER-DEEPM .
make yes-kSPACE
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 LAMMPS source code by

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

## 2.3.2 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.tar.gz
tar xf stable_23Jun2022.tar.gz
```

The source code of LAMMPS is stored in directory `lammps-stable_23Jun2022`. 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/build/
cd lammps-stable_23Jun2022/build/
```

Now build LAMMPS. Note that `PLUGIN` and `KSPACE` package 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
```

## 2.4 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, force and virial). 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
```



## 2.5 Install GROMACS with DeePMD

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

### 2.5.1 Patch source code of GROMACS

Download 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, only support 2020.2 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.

### 2.5.2 Compile GROMACS with deepmd-kit

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

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

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

make -j
make install
```

## 2.6 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 executing

```
./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
```

**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 organize data in **systems**. Each **system** is composed by a number of **frames**. One may roughly view a **frame** as a snap shot on an MD trajectory, but it does not necessary 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 virial. 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 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 H<sub>2</sub>O). To contains data with different formula, one need to divide data into multiple systems.

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

ID	Property	Raw file	Re-quired/Optional	Shape	Description
type	Atom type indexes	type.raw	Required	Natoms	Integers that start with 0
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
nopbc	Non-periodic system	nopbc	Optional	1	If True, this system is non-periodic; otherwise it's periodic

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

ID	Property	Raw file	Unit	Re- quired/Optional	Shape	Description
co- ord	Atomic coordi- nates	co- ord.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	

The labeled frame properties is 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 * 3	in the order XX XY XZ YX YY YZ ZX ZY ZZ
atom_ener	Atomic ener- gies	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 polariz- ability	polarizabil- ity.raw	Any	Nframes * 9	in the order XX XY XZ YX YY YZ ZX ZY ZZ
atomic_polarizability	Atomic polariz- ability	atomic_polarizabil- ity.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 `.npz` 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.npz
set.000/energy.npz
set.000/force.npz
set.001/coord.npz
set.001/energy.npz
set.001/force.npz
```

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 name. The mapping can be given by the file `type_map.raw`. For example

```
$ cat type_map.raw
0 H
```

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

### 3.2.2 HDF5 format

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

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

Here, `/path/to/data.hdf5` is the path and `H2O` is the key. There should be some data in the `H2O` group, such as `H2O/type.raw` and `H2O/set.000/force.npz`.

A 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 are 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 contains 2000 frames with the Numpy format.

## 3.3 Prepare data with dpdata

One can use the 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/npz', '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 at the [official website](#).





## 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 with the actual name of the element) of the corresponding atom types. A model for water, 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. `loc_frame`: Defines a local frame at each atom, and the compute the descriptor as local coordinates under this frame.
5. `hybrid`: Concatenate a list of descriptors to form a new descriptor.

The fitting of the following physical properties are 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](#).

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]` denote 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 input end to the output end, respectively. If the outer layer is of twice size as 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`, then descriptor will consider the types of neighbor atoms. Otherwise, both the types of centric and neighbor atoms are considered.
- The `axis_neuron` specifies the size of 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 "hybrid"

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

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
```

## 4.6 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 0 H
```

where `data` is the directory of data, `6.0` is the cutoff radius, and `0` 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 to `[46, 92]` in the water example.

## 4.7 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.

### 4.7.1 The fitting network

The construction of the fitting net is give 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.

### 4.7.2 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, force and virial, respectively.  $p_e$ ,  $p_f$ , and  $p_v$  give the prefactors of the energy, force and virial losses. The prefactors may not be a constant, rather it changes linearly with the learning rate. Taking the force prefactor for example, at training step  $t$ , it is given by

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

where  $\alpha(t)$  denotes the learning rate at step  $t$ .  $p_f^0$  and  $p_f^\infty$  specifies the  $p_f$  at the start of the training and at the limit of  $t \rightarrow \infty$  (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.

## 4.8 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 them for 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 remains the same as `ener` mode, and their meaning can be found [here](#). To fit a tensor, one need to modify `model/fitting_net` and `loss`.

### 4.8.1 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 water system, `sel_type` is `[0]` since 0 represents for atom O. If left unset, all type of atoms will be fitted.
- The rest `args` has the same meaning as they do in `ener` mode.

## 4.8.2 Loss

DP supports a combinational training of 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  $1 \times 9$  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 a atomic system, each atom in `sel_type` has a `tensor` label. For example, when fitting dipole, each frame will provide a `#sel_atom`  $\times$  3 matrix, where `#sel_atom` is the number of atoms whose type are in `sel_type`.

The `loss` section tells DP the weight of this two kind 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, system with corresponding label will NOT be included in the training process.

## 4.8.3 Training Data Preparation

In tensor mode, the identification of label's type (global or atomic) is derived from the file name. The global label should be named as `dipole.npy/raw` or `polarizability.npy/raw`, while the atomic label should be named as `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 label.

## 4.8.4 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

(continues on next page)

(continued from previous page)

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 local loss and global loss will be 0.0. This is because our training data and validation data consist of global system and atomic system, i.e.

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

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

## 4.9 Type embedding approach

We generate specific 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.

### 4.9.1 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" : {
    ...
  }
}
```

(continues on next page)



(continued from previous page)

```
}
}
```

Model will automatically apply type embedding approach and generate type embedding vectors. If type embedding vector is detected, descriptor and fitting net would take it as a part of 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 input end to the output end, respectively. It takes one-hot vector as input and output dimension equals to the last dimension of the `neuron` list. If the outer layer is of twice size as 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 find in the directory.

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

See [here](#) for further explanation of type embedding.

P.S.: You can't apply compression method while using atom type embedding

## 4.10 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 training in two steps.

### 4.10.1 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 to 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 associate to 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 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
```

#### 4.10.2 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  $\text{\AA}^{-1}$ ) gives the spread parameter controls the spread of Gaussian charges, and `ewald_h` (unit  $\text{\AA}$ ) assigns the grid size of Fourier transform. The DPLR model can be trained and frozen by (from the example directory)

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

### 4.10.3 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 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 to 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_style "zero" is used.
pair_style      deepmd ener.pb
pair_coeff      * *
bond_style      zero
bond_coeff      *
special_bonds   lj/coul 1 1 1 angle no
```

Type 1 and 2 (O and H) are `real_atoms`, while type 3 (WCs) are `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    ppm/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 ppm/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 atoms.

```
# 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 a very short training steps, thus is of poor quality.

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

## 4.11 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:

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

See the [JCTC paper](#) for details.

### 4.11.1 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  $E_{\text{MM}}$  is eliminated.

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

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

(continues on next page)

(continued from previous page)

```

        "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`.

### 4.11.3 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.

## TRAINING

### 5.1 Train a model

Several examples of training can be found at 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 worth special notice.

```
DEEPMD INFO    ---Summary of DataSystem: training  -----
↪--
DEEPMD INFO    found 3 system(s):
DEEPMD INFO
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
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 by three data systems, while the validation data set is composed by one data system. The number of atoms, batch size, 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 `1curve.out`). The batch size can be set in

the input script by the key `batch_size` in the corresponding sections for 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, 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. One can visualize this file by 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 prefix `save_ckpt` every `save_freq` training steps.

### 5.1.1 Warning

It is warned that the example water data (in folder `examples/water/data`) is of very limited amount, is provided only for testing purpose, 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,  $\alpha$  is the learning rate,  $\alpha_0$  is the starting learning rate (set by `start_lr`),  $\lambda$  is the decay rate, and  $\tau$  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 pick `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 optional 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 in 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 are divided into blocks. The block *i* has systems ranging from *sidx\_i* to *eidx\_i*. The probability of using a system from block *i* is in proportional to *w\_i*. Within one block, the probability of using a system is in proportional to its size.
- An example of using "auto\_prob" is given as 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 as `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` that 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 the mixed precision training:
- Only `se_e2_a` type descriptor is supported by the mixed precision training workflow.
- The precision of embedding net and 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 check point.

### 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 get the best performance, one should control the number of threads used by DeePMD-kit. This is achieved by three environmental variables: `OMP_NUM_THREADS`, `TF_INTRA_OP_PARALLELISM_THREADS` and `TF_INTER_OP_PARALLELISM_THREADS`. `OMP_NUM_THREADS` controls the multithreading of DeePMD-kit implemented operations. `TF_INTRA_OP_PARALLELISM_THREADS` and `TF_INTER_OP_PARALLELISM_THREADS` controls `intra_op_parallelism_threads` and `inter_op_parallelism_threads`, which are Tensorflow configurations for multithreading. An explanation is found [here](#).

For example 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
```

It is encouraged to adjust the configurations after empirical testing.

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	ON	0	Enable auto parallelization for CPU operators.

### 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 need 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* enable, or hybrid composed of 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 further 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.

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.

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  $(N_t + 1) * N_t / 2 + 1$  columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

smin\_alpha:

type: float, optional  
argument path: model/smin\_alpha

The short-range tabulated interaction will be 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: [2, 4, 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”.. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version.

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”. Default follows the interface precision.

trainable:

type: bool, optional, default: True

argument path: model/type\_embedding/trainable

If the parameters in the embedding net are trainable

seed:

type: int | NoneType, optional

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*

The type of the descriptor. See explanation below.

- *loc\_frame*: Defines a local frame at each atom, and the compute the descriptor as local coordinates under this frame.

- `se_e2_a`: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor.
- `se_e2_r`: Used by the smooth edition of Deep Potential. Only the distance between atoms is used to construct the descriptor.
- `se_e3`: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Three-body embedding will be used by this descriptor.
- `se_a_tpe`: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Type embedding will be used by this descriptor.
- `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 first 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]`: 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.

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

`sel`:

type: `list` | `str`, 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 wrapped 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”.. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version.

`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

Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  embedding nets

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”. 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: int | NoneType, 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: list | str, 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 wrapped 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”.. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version.

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”. 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: int | NoneType, 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: `list` | `str`, 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 wrapped 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”.. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version.

`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`

Try to build  $N\_types$  embedding nets. Otherwise, building  $N\_types^2$  embedding nets

`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”. 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: `int` | `NoneType`, 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: `list` | `str`, 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 wrapped 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”.. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version.

`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`

Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  embedding nets

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”. 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: `int` | `NoneType`, 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

fitting\_net:

type: `dict`

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*, *dipole*, *polar*

The type of the fitting. See explanation below.

- ener: Fit an energy model (potential energy surface).
- dipole: Fit an atomic dipole model. Global dipole labels or atomic dipole labels for all the selected atoms (see sel\_type) should be provided by dipole.npy in each data system. The file either has number of frames lines and 3 times of number of selected atoms columns, or has number of frames lines and 3 columns. See loss parameter.
- polar: Fit an atomic polarizability model. Global polarizability labels or atomic polarizability labels for all the selected atoms (see sel\_type) should be provided by polarizability.npy in each data system. The file either has number of frames lines and 9 times of number of selected atoms columns, or has number of frames lines and 9 columns. See loss parameter.

When `type` is set to `ener`:

`numb_fparam`:

type: `int`, optional, default: 0

argument path: `model/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”.. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version.

`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”. 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: `list | bool`, 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 this 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 initial energy shift for each type of atoms.

seed:

type: `int | NoneType`, 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

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”.. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version.

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”. Default follows the interface precision.



`sel_type`:

type: `list | NoneType | int`, 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: `int | NoneType`, 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”.. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version.

`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”. 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: `list | float`, 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: `list` | `NoneType` | `int`, 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: `int` | `NoneType`, optional

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

Random seed for parameter initialization of the fitting net

`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  $\text{\AA}^{-1}$

`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  $\text{\AA}$

`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.

`learning_rate`:

type: dict

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.

`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`, `tensor`

The type of the loss. When the fitting type is `ener`, the loss type should be set to `ener` or left unset. When the fitting type is `dipole` or `polar`, the loss type should be set to `tensor`. .

When `type` is set to `ener`:

`start_pref_e`:

type: `int` | `float`, optional, default: 0.02  
argument path: `loss[ener]/start_pref_e`

The prefactor of energy loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the energy label should be provided by file `energy.npy` in each data system. If both `start_pref_energy` and `limit_pref_energy` are set to 0, then the energy will be ignored.

`limit_pref_e`:

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

argument path: `loss[ener]/limit_pref_e`

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

start\_pref\_f:

type: `int | float`, optional, default: 1000

argument path: `loss[ener]/start_pref_f`

The prefactor of force loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the force label should be provided by file `force.npy` in each data system. If both `start_pref_force` and `limit_pref_force` are set to 0, then the force will be ignored.

limit\_pref\_f:

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

argument path: `loss[ener]/limit_pref_f`

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

start\_pref\_v:

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

argument path: `loss[ener]/start_pref_v`

The prefactor of virial loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the virial label should be provided by file `virial.npy` in each data system. If both `start_pref_virial` and `limit_pref_virial` are set to 0, then the virial will be ignored.

limit\_pref\_v:

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

argument path: `loss[ener]/limit_pref_v`

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

start\_pref\_ae:

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

argument path: `loss[ener]/start_pref_ae`

The prefactor of atom\_ener loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the atom\_ener label should be provided by file `atom_ener.npy` in each data system. If both `start_pref_atom_ener` and `limit_pref_atom_ener` are set to 0, then the atom\_ener will be ignored.

limit\_pref\_ae:

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

argument path: `loss[ener]/limit_pref_ae`

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

start\_pref\_pf:

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

argument path: `loss[ener]/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 non-zero value, the atom\_pref label should be provided by file atom\_pref.npy in each data system. If both start\_pref\_atom\_pref and limit\_pref\_atom\_pref are set to 0, then the atom\_pref will be ignored.

limit\_pref\_pf:

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

argument path: `loss[ener]/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: `NoneType` | `float`, 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.  $DF_i / (\|F\| + \text{relative\_f})$  with  $DF$  denoting the difference between prediction and label and  $\|F\|$  denoting the L2 norm of the label.

enable\_atom\_ener\_coeff:

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

argument path: `loss[ener]/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 `tensor`:

pref:

type: `int` | `float`

argument path: `loss[tensor]/pref`

The prefactor of the weight of global loss. It should be larger than or equal to 0. It controls the weight of loss corresponding to global label, i.e. 'polarizability.npy' or dipole.npy, whose shape should be #frames x [9 or 3]. If it's larger than 0.0, this npy should be included.

pref\_atomic:

type: `int` | `float`

argument path: `loss[tensor]/pref_atomic`

The prefactor of the weight of atomic loss. It should be larger than or equal to 0. It controls the weight of loss corresponding to atomic label, i.e. atomic\_polarizability.npy or atomic\_dipole.npy, whose shape should be #frames x ([9 or 3] x #selected atoms). If it's larger than 0.0, this npy should be included. Both pref and pref\_atomic should be provided, and either can be set to 0.0.

training:

type: `dict`

argument path: `training`

The training options.

training\_data:

type: `dict`

argument path: `training/training_data`

Configurations of training data.

systems:

type: list | str

argument path: training/training\_data/systems

The data systems for training. This key can be provided with a list that specifies the systems, or be provided with a string by which the prefix of all systems are given and the list of the systems is automatically generated.

set\_prefix:

type: str, optional, default: set

argument path: training/training\_data/set\_prefix

The prefix of the sets in the [systems](#).

batch\_size:

type: list | int | str, 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.

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  $1.0/\text{self.get\_nsystems}()$
- “prob\_sys\_size” : the probability of a system is proportional to the number of batches in the system
- “prob\_sys\_size;stt\_idx:end\_idx:weight;stt\_idx:end\_idx:weight;...” : the list of systems is divided into blocks. A block is specified by stt\_idx:end\_idx:weight, where stt\_idx is the starting index of the system, end\_idx is then ending (not including) index of the system, the probabilities of the systems in this block sums up to weight, and the relatively probabilities within this block is proportional to the number of batches in the system.

sys\_probs:

type: list | NoneType, optional, default: None, alias: sys\_weights

argument path: training/training\_data/sys\_probs

A list of float if specified. Should be of the same length as systems, specifying the probability of each system.

validation\_data:

type: `NoneType | dict`, 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: `list | str`

argument path: `training/validation_data/systems`

The data systems for validation. This key can be provided with a list that specifies the systems, or be provided with a string by which the prefix of all systems are given and the list of the systems is automatically generated.

set\_prefix:

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

argument path: `training/validation_data/set_prefix`

The prefix of the sets in the [systems](#).

batch\_size:

type: `list | int | str`, optional, default: `auto`

argument path: `training/validation_data/batch_size`

This key can be

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

auto\_prob:

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

argument path: `training/validation_data/auto_prob`

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

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

sys\_probs:

type: `list | NoneType`, optional, default: `None`, alias: `sys_weights`

argument path: `training/validation_data/sys_probs`



A list of float if specified. Should be of the same length as systems, specifying the probability of each system.

`numb_btch:`

type: `int`, optional, default: 1, alias: `numb_batch`

argument path: `training/validation_data/numb_btch`

An integer that specifies the number of systems 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 only currently.

`numb_steps:`

type: `int`, alias: `stop_batch`

argument path: `training/numb_steps`

Number of training batch. Each training uses one batch of data.

`seed:`

type: `int` | `NoneType`, optional

argument path: `training/seed`

The random seed for getting frames from the training data set.

`disp_file:`

type: `str`, optional, default: `lcurve.out`

argument path: `training/disp_file`

The file for printing learning curve.

`disp_freq:`

type: `int`, optional, default: 1000

argument path: `training/disp_freq`

The frequency of printing learning curve.

`save_freq:`

type: `int`, optional, default: 1000

argument path: `training/save_freq`

The frequency of saving check point.

save\_ckpt:

type: `str`, optional, default: `model.ckpt`

argument path: `training/save_ckpt`

The 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 during training.

profiling:

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

argument path: `training/profiling`

Profiling during training.

profiling\_file:

type: `str`, optional, default: `timeline.json`

argument path: `training/profiling_file`

Output file for profiling.

enable\_profiler:

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

argument path: `training/enable_profiler`

Enable TensorFlow Profiler (available in TensorFlow 2.3) to analyze performance. The log will be saved to `tensorboard_log_dir`.

tensorboard:

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

argument path: `training/tensorboard`

Enable tensorboard

tensorboard\_log\_dir:

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

argument path: `training/tensorboard_log_dir`

The log directory of tensorboard outputs

tensorboard\_freq:

type: `int`, optional, default: `1`

argument path: `training/tensorboard_freq`

The frequency of writing tensorboard events.

nvnmd:

type: dict, optional  
argument path: `nvnmmd`

The nvnmmd options.

net\_size:

- type: `int`
- argument path: `nvnmmd/net_size`
- configuration the number of nodes of fitting\_net, just can be set as 128

map\_file:

- type: `str`
- argument path: `nvnmmd/map_file`
- A file containing the mapping tables to replace the calculation of embedding nets

config\_file:

- type: `str`
- argument path: `nvnmmd/config_file`
- A file containing the parameters about how to implement the model in certain hardware

weight\_file:

- type: `str`
- argument path: `nvnmmd/weight_file`
- a \*.numpy file containing the weights of the model

enable:

- type: `bool`
- argument path: `nvnmmd/enable`
- enable the nvnmmd training

restore\_descriptor:

- type: `bool`
- argument path: `nvnmmd/restore_descriptor`
- enable to restore the parameter of embedding\_net from weight.npy

restore\_fitting\_net:

- type: `bool`
- argument path: `nvnmmd/restore_fitting_net`
- enable to restore the parameter of fitting\_net from weight.npy

quantize\_descriptor:

- type: `bool`
- argument path: `nvnmmd/quantize_descriptor`
- enable the quantization of descriptor

quantize\_fitting\_net:

- type: `bool`
- argument path: `nvnmmd/quantize_fitting_net`

enable the quantization of fitting\_net

## 5.4 Parallel training

Currently, parallel training is enabled in a synchronized way with help of [Horovod](#). Depend on the number of training processes (according to MPI context) and 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 is 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 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 scale 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 a 8-GPU host, linear acceleration can be observed with 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, environment variable `CUDA_VISIBLE_DEVICES` must be set to control parallelism on the occupied host where one process is bound to one GPU card.

Note that `OMP_NUM_THREADS`, `TF_INTRA_OP_PARALLELISM_THREADS`, and `TF_INTER_OP_PARALLELISM_THREADS` should be carefully adjusted to achieve the best performance.

When using MPI with Horovod, `horovodrun` is a simple wrapper around `mpirun`. In the case where fine-grained control over options 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 on HPC environment.

Whether distributed workers are initiated can be observed at 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 TensorBoard Usage

TensorBoard provides the visualization and tooling needed for machine learning experimentation. A full instruction of tensorboard can be found [here](#).

### 5.5.1 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

### 5.5.2 How to use Tensorboard with DeePMD-kit

Before running TensorBoard, make sure you have generated summary data in a log directory by modifying the the input script, set `tensorboard` to true in 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`.

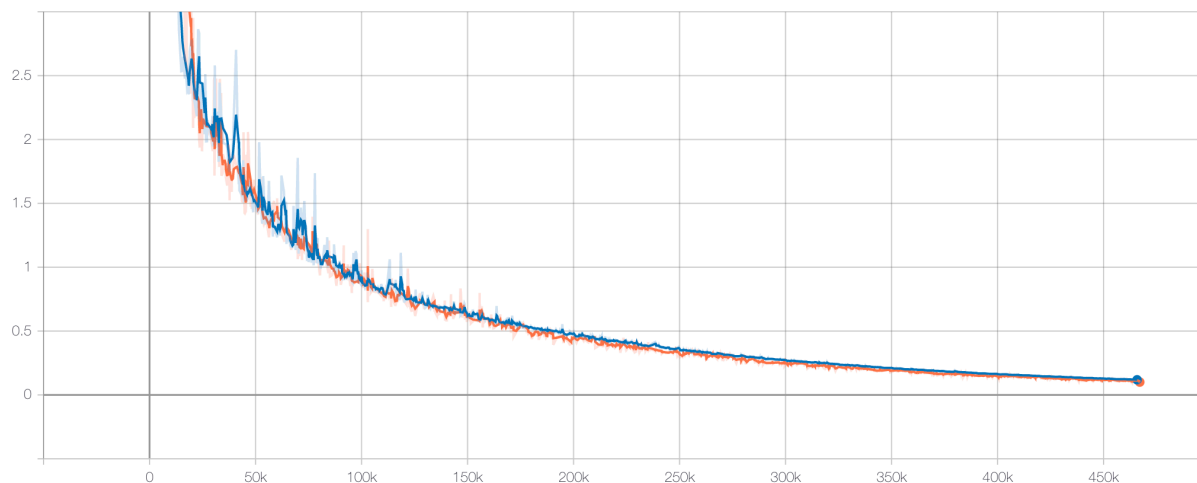
TensorBoard requires a logdir to read logs from. For info on configuring TensorBoard, run `tensorboard -help`. One can easily change the log name with “`tensorboard_log_dir`” and the sampling frequency with “`tensorboard_freq`”.

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

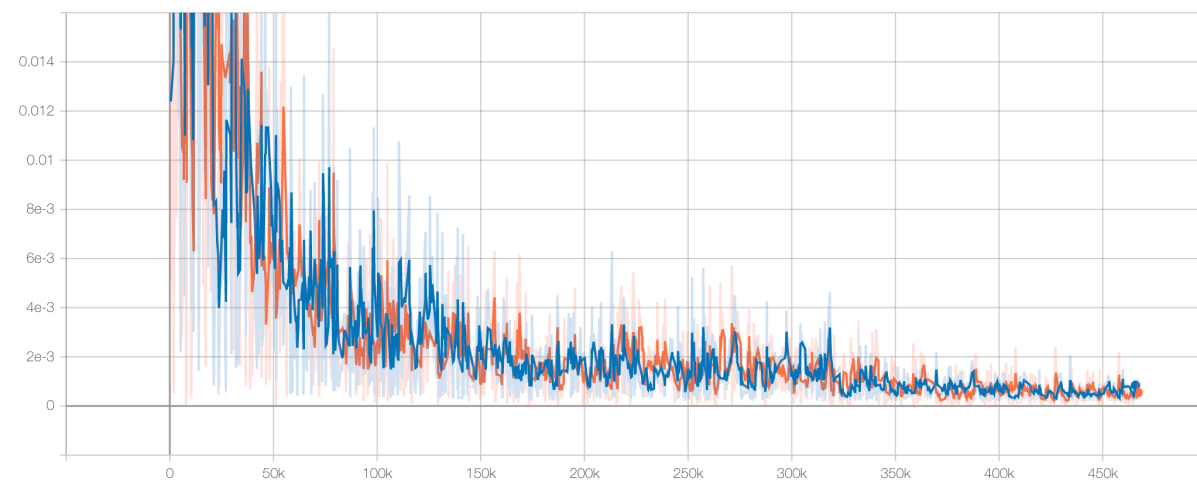
### 5.5.3 Examples

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

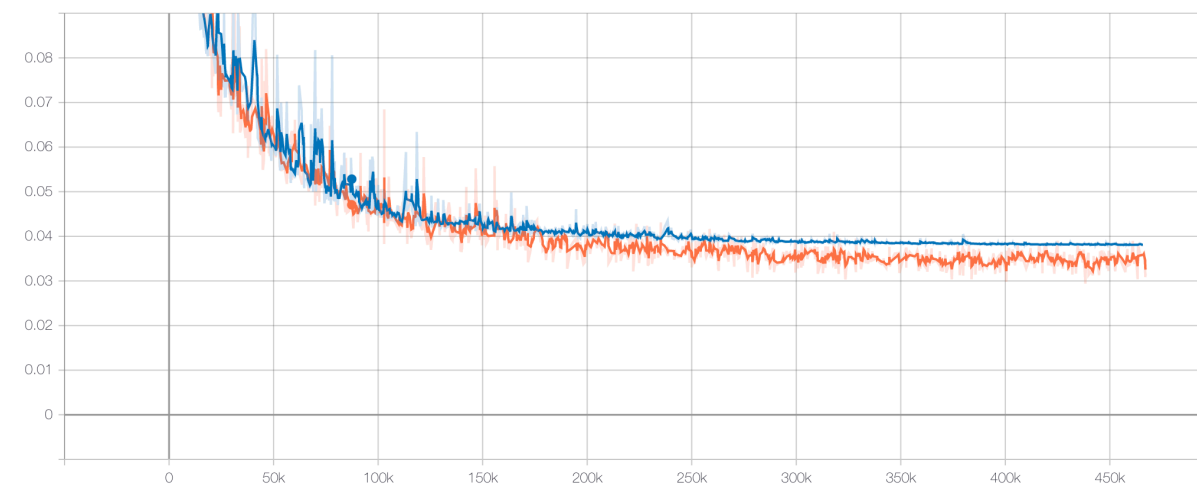
l2\_loss



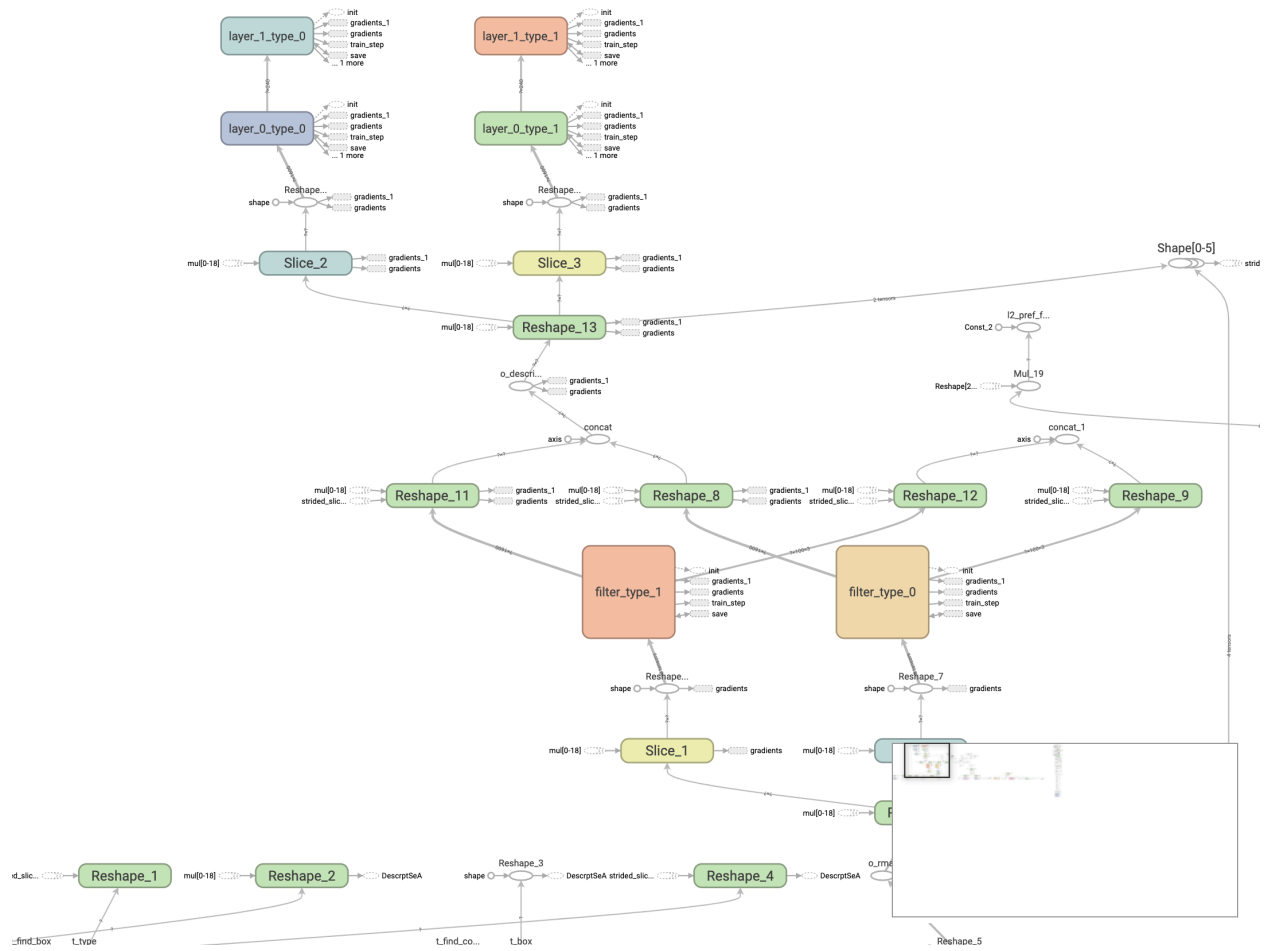
l2\_ener\_loss



l2\_force\_loss

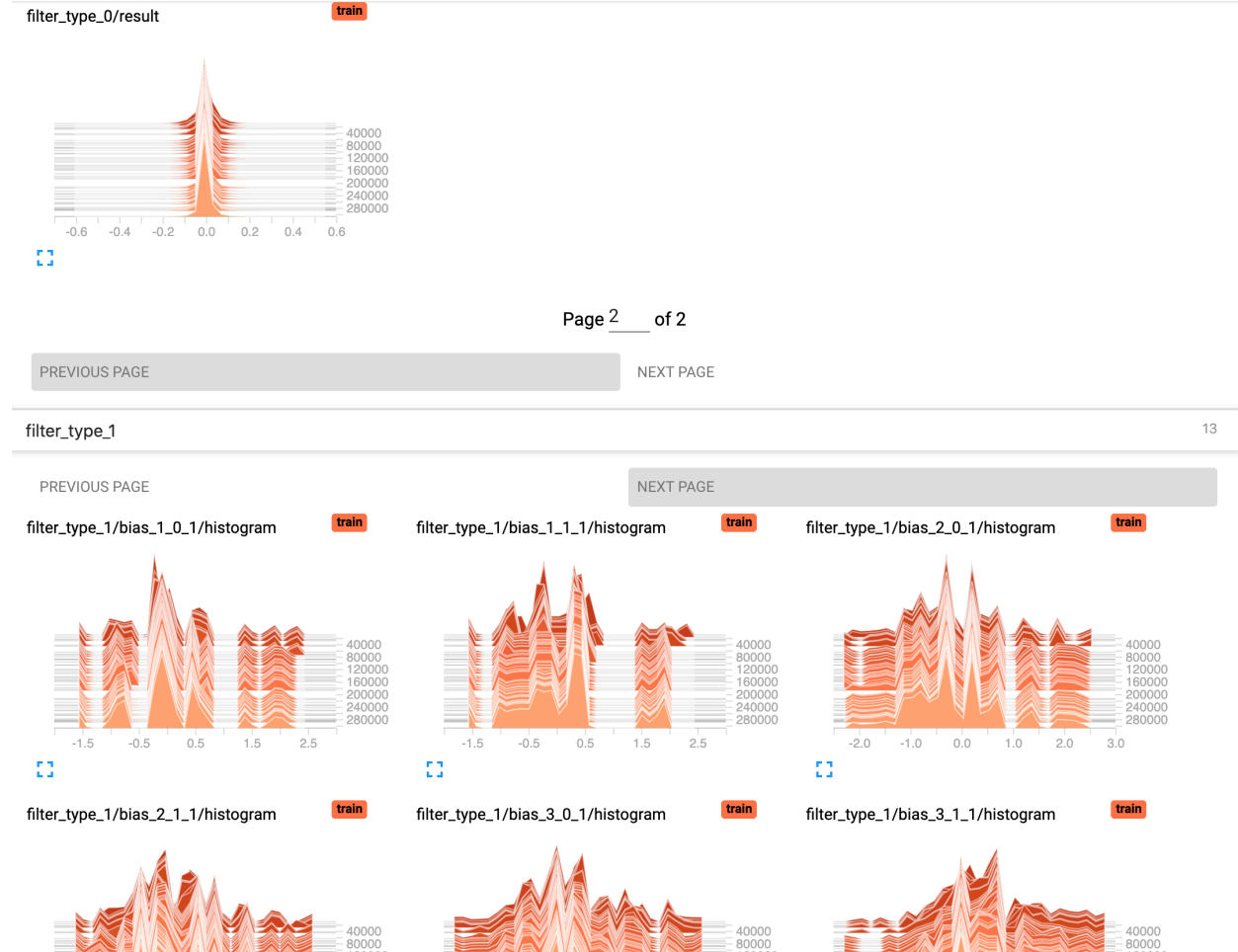


## Visualizing deepmd-kit model graph





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

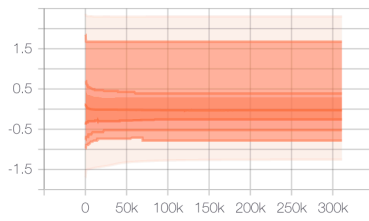


filter\_type\_0

PREVIOUS PAGE

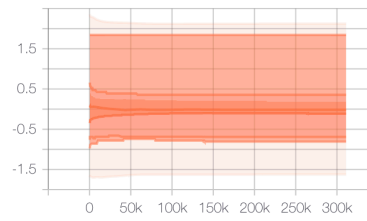
NEXT PAGE

filter\_type\_0/bias\_1\_0\_1/histogram



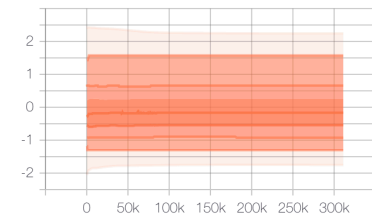
train

filter\_type\_0/bias\_1\_1\_1/histogram



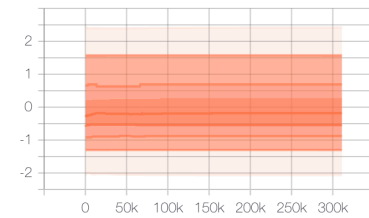
train

filter\_type\_0/bias\_2\_0\_1/histogram



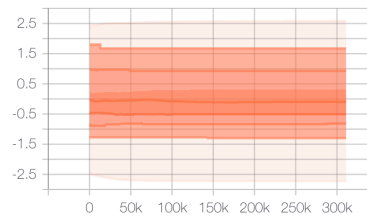
train

filter\_type\_0/bias\_2\_1\_1/histogram



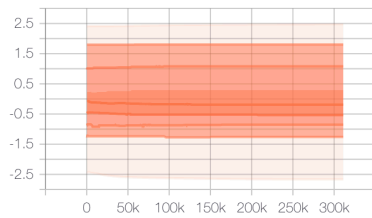
train

filter\_type\_0/bias\_3\_0\_1/histogram



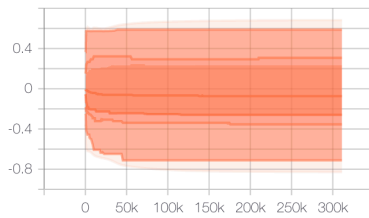
train

filter\_type\_0/bias\_3\_1\_1/histogram



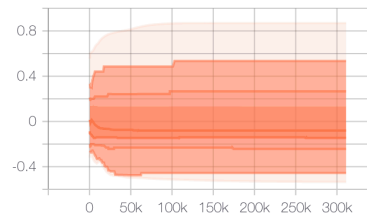
train

filter\_type\_0/matrix\_1\_0\_1/histogram



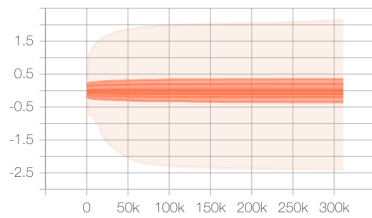
train

filter\_type\_0/matrix\_1\_1\_1/histogram



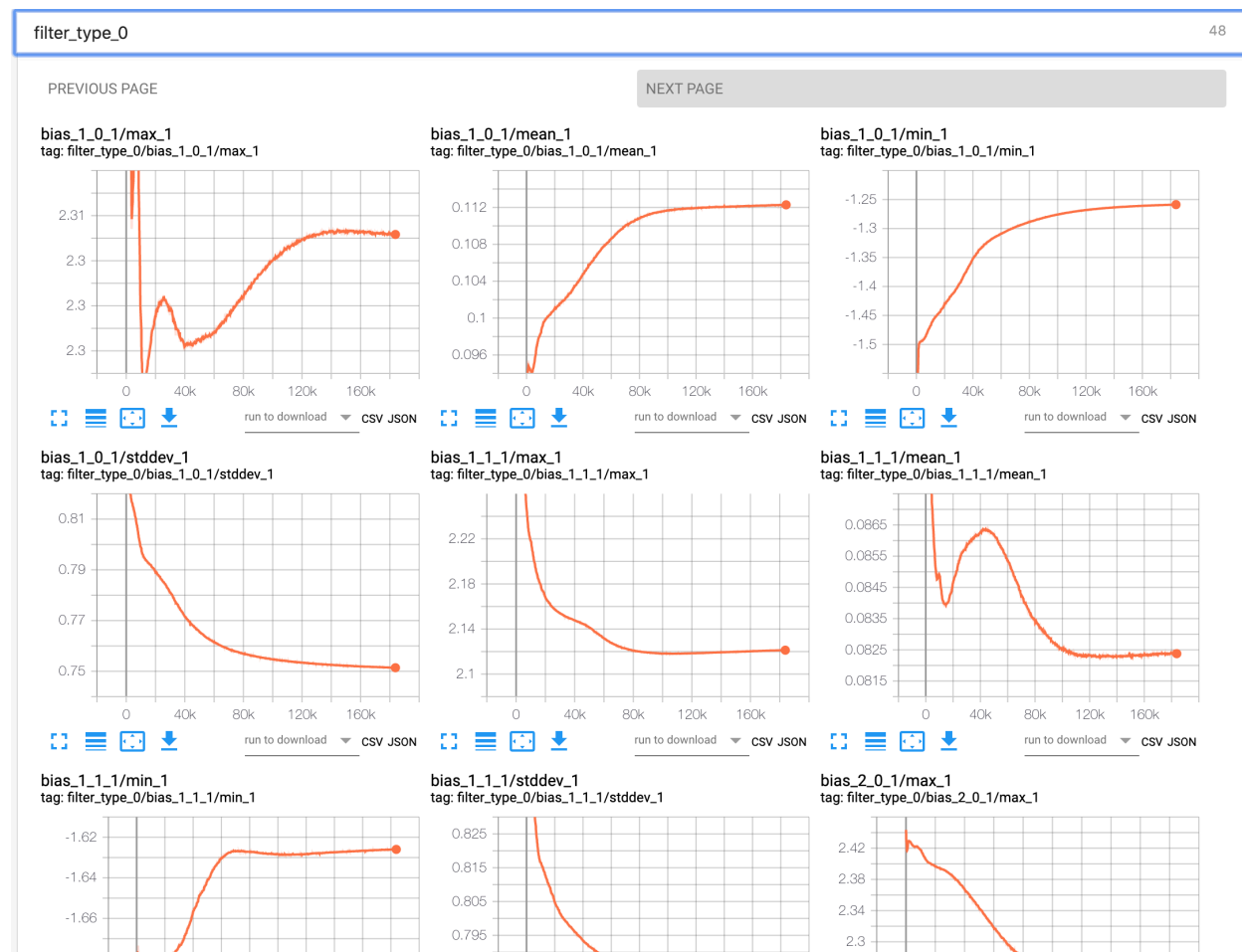
train

filter\_type\_0/matrix\_2\_0\_1/histogram



train

## Viewing summaries of trainable variables



### 5.5.4 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.

## 5.6 Known limitations of using GPUs

If you use deepmd-kit in a GPU environment, the acceptable value range of some variables are additionally restricted compared to the CPU environment due to the software's GPU implementations:

1. The number of atom type of a given system must be less than 128.
2. The maximum distance between an atom and it's 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 actually limited by the GPU memory size currently, usually within 1000,000 atoms even at 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 embedding net must be less than 1024 during the model compression process.

## FREEZE AND COMPRESS

### 6.1 Freeze a model

The trained neural network is extracted from a checkpoint and dumped into a database. 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 database is called `graph.pb`.

### 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:
  -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.
```

(continues on next page)

(continued from previous page)

```

        '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 including 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 it's uniform stride, while the second sub-table takes `10 * stride` as it's 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 overflow 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 the 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 original model generation should be 2.0.0-alpha.0 or above. If one has a frozen 1.2 or 1.3 model, one can upgrade it through the `dp convert-from` interface.(eg: `dp convert-from 1.2/1.3 -i old_frozen_model.pb -o new_frozen_model.pb`)

#### Acceptable descriptor type

Descriptors with `se_e2_a`, `se_e3`, `se_e2_r` type are supported by the model compression feature. Hybrid mixed with above descriptors is also supported.

Available activation functions for descriptor:

- tanh
- gelu
- relu
- relu6
- softplus
- sigmoid





## 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 file containing details of energy force and virial
                        accuracy
```

## 7.2 Calculate Model Deviation

One can also use a subcommand to calculate 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 with respect to 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.

## 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

```
from deepmd.infer import DeepPot
import numpy as np
dp = DeepPot('graph.pb')
coord = np.array([[1,0,0], [0,0,1.5], [1,0,3]]).reshape([1, -1])
cell = np.diag(10 * np.ones(3)).reshape([1, -1])
atype = [1,0,1]
e, f, v = dp.eval(coord, cell, atype)
```

where *e*, *f* and *v* are predicted energy, force and virial of the system, respectively.

Furthermore, one can use the python interface to calculate model deviation.

```
from deepmd.infer import calc_model_devi
from deepmd.infer import DeepPot as DP
import numpy as np

coord = np.array([[1,0,0], [0,0,1.5], [1,0,3]]).reshape([1, -1])
cell = np.diag(10 * np.ones(3)).reshape([1, -1])
atype = [1,0,1]
graphs = [DP("graph.000.pb"), DP("graph.001.pb")]
model_devi = calc_model_devi(coord, cell, atype, graphs)
```

### 8.2 C++ interface

The C++ interface of DeePMD-kit is also available for model interface, which is considered faster than 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.};
```

(continues on next page)

(continued from previous page)

```
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.

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

```
gcc infer_water.cpp -D HIGH_PREC -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
```

## COMMAND LINE INTERFACE

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

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

### 9.1 Named Arguments

<code>--version</code>	show program's version number and exit
------------------------	--

### 9.2 Valid subcommands

<code>command</code>	Possible choices: config, transfer, train, freeze, test, compress, doc-train-input, model-devi, convert-from, neighbor-stat, train-nvnmd
----------------------	--

### 9.3 Sub-commands:

#### 9.3.1 config

fast configuration of parameter file for smooth model

```
dp config [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
          [-o OUTPUT]
```

### Named Arguments

<code>-v, --log-level</code>	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"
<code>-l, --log-path</code>	set log file to log messages to disk, if not specified, the logs will only be output to console
<code>-o, --output</code>	the output json file Default: "input.json"

### 9.3.2 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

<code>-v, --log-level</code>	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"
<code>-l, --log-path</code>	set log file to log messages to disk, if not specified, the logs will only be output to console
<code>-r, --raw-model</code>	the model receiving parameters Default: "raw_frozen_model.pb"
<code>-O, --old-model</code>	the model providing parameters Default: "old_frozen_model.pb"
<code>-o, --output</code>	the model after passing parameters Default: "frozen_model.pb"

### 9.3.3 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]
         [-o OUTPUT] [-f INIT_FRZ_MODEL] [--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 'work-  
                              ers' 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.

-o, --output            The output file of the parameters used in training.  
                              Default: "out.json"

-f, --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

### 9.3.4 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]
```

## 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

### 9.3.5 test

test the model

```
dp test [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH] [-m MODEL]
        [-s SYSTEM] [-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: “.”
- 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 File where details of energy force and virial accuracy will be written
- a, --atomic Test the accuracy of atomic label, i.e. energy / tensor (dipole, polar)  
Default: False



### 9.3.6 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	<p>Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0</p> <p>set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG</p> <p>Default: "INFO"</p>
-l, --log-path	<p>set log file to log messages to disk, if not specified, the logs will only be output to console</p>
-m, --mpi-log	<p>Possible choices: master, collect, workers</p> <p>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</p> <p>Default: "master"</p>
-i, --input	<p>The original frozen model, which will be compressed by the code</p> <p>Default: "frozen_model.pb"</p>
-o, --output	<p>The compressed model</p> <p>Default: "frozen_model_compressed.pb"</p>
-s, --step	<p>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</p> <p>Default: 0.01</p>
-e, --extrapolate	<p>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]</p> <p>Default: 5</p>
-f, --frequency	<p>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</p> <p>Default: -1</p>

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

### 9.3.7 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 OUT_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
- out-type The output type  
Default: “rst”

### 9.3.8 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

### 9.3.9 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]
                {0.12,1.0,1.1,1.2,1.3,2.0}
```

#### Positional Arguments

FROM	Possible choices: 0.12, 1.0, 1.1, 1.2, 1.3, 2.0 The original model compatibility
------	---

#### 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”

### 9.3.10 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 ...]
```

### 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

### 9.3.11 train-nvnmd

train nvnmd model

```
dp train-nvnmd [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
                [-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
-s, --step	Possible choices: s1, s2 steps to train model of NVNMD: s1 (train CNN), s2 (train QNN) Default: "s1"

## 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., 1., 1.)],
               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      * *
```

where `graph.pb` is the file name of the frozen model. It should be noted that LAMMPS counts atom types starting from 1, therefore, all LAMMPS atom type will be firstly subtracted by 1, and then passed into the DeePMD-kit engine to compute the interactions.

## 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 `atomic` or `relative`

#### 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
```

#### 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](#).

The model deviation evaluate 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 will be output. 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 the force and  $l$  is provided as the parameter of the keyword `relative`.

### Restrictions

- The `deepmd` pair style is provided in the USER-DEEPMO 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 provide 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.

### Examples

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

The result of the compute can be dump 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-DEEPMO 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 `kpace_style`. To use it with DeePMD-kit, one writes

```
pair_style        deepmd graph.pb
pair_coeff
kpace_style       pppm 1.0e-5
kpace_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 `kpace_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:

$$d_{\text{vatom}} = \sum_m (\mathbf{r}_n - \mathbf{r}_m) \frac{de_m}{d\mathbf{r}_n}$$

Where  $\mathbf{r}_n$  is the atomic position of nth atom,  $\mathbf{v}_n$  velocity of atom and  $\frac{de_m}{d\mathbf{r}_n}$  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 component 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 per-atom stress tensor one can, for example, compute the heat flux defined as:

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

to compute the heat flux with LAMMPS:

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

#### Examples

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

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

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



## 10.4 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` (or `dp_ipi_low` for low precision) that computes the interactions (including energy, force and virial). The server and client communicates via the Unix domain socket or the Internet socket. Installation instructions of 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 is allow for computing, in parallel, the interactions of multiple replica 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 `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 descibed by a neural network potential (NNP) have to be turned off. In GROMACS, bonded interactions can be turned off by modifying [ bonds ], [ angles ], [ dihedrals ] and [ pairs ] sections. And LJ and Columb interactions must be turned off by [ exclusions ] section.

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

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

[ moleculetype ]
;name nrexcl
methane 3

[ atoms ]
; nr type resnr residue atom cgnr charge mass
1 c3 1 MOL C1 1 -0.1068 12.010
2 hc 1 MOL H1 2 0.0267 1.008
3 hc 1 MOL H2 3 0.0267 1.008
4 hc 1 MOL H3 4 0.0267 1.008
5 hc 1 MOL H4 5 0.0267 1.008

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

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

For comparsion, the original topology file genearted 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
```

(continues on next page)

(continued from previous page)

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 simulation, we need to tell GROMACS to use DeepPotential by setting 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 in consistent with 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

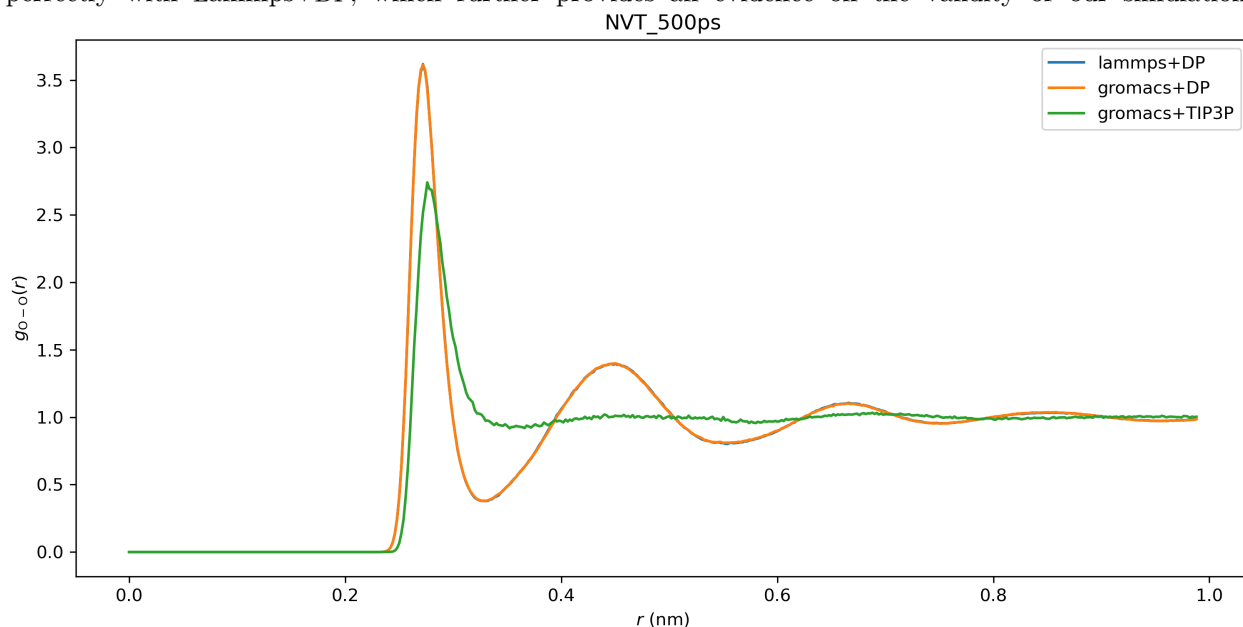
Finally, you can run GROMACS using `gmx mdrun` as usual.

### 10.5.2 All-atom DP Simulation

This part gives an example on how to run a simulation with 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 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 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 the Westlake University.

### 10.6.3 AMBER interface to DeePMD-kit

An `AMBER` interface to DeePMD-kit is written by the [York Lab](#) from the 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 the 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](#).



## 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](https://doi.org/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 the continuous neural network (CNN) training, but also the 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 a ML model that can decently reproduce the PES, 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 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 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 random 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 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 which is used to performs NVNMD in server [<http://nvnmd.picp.vip>]

## 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, the file containing details of energy, force and virial 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	<input style="width: 90%;" type="text"/>
Password	<input style="width: 90%;" type="password"/>

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	<input type="text" value="test"/>
Upload mode <sup>?</sup>	<span>Manual upload</span> <span>▼</span>
Input script	<span>Browse...</span> in.lmp
Model file	<span>Browse...</span> model.pb
Data files	<span>Browse...</span> 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 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	<a href="#">Cancel</a>			

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.Imp	Cancelled		0:01:20	<a href="#">Package</a> <a href="#">Separate files</a>	<a href="#">Delete</a>

### 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 <sup>②</sup>
output.zip	1.2 MB	<a href="#">Download</a>		<a href="#">Upload</a>

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	<a href="#">Download</a>		<a href="#">Upload</a>
in.lmp	3.1 KB	<a href="#">Download</a>		<a href="#">Upload</a>
lammps.xyz	2.1 MB	<a href="#">Download</a>		<a href="#">Upload</a>
log.lammps	14.0 KB	<a href="#">Download</a>		<a href="#">Upload</a>
model.pb	8.1 MB	<a href="#">Download</a>		<a href="#">Upload</a>
result.out	13.5 KB	<a href="#">Download</a>		<a href="#">Upload</a>

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.

## FAQS

In consequence of various differences of computers or systems, problems may occur. Some common circumstances are listed as follows. In addition, some frequently asked questions about parameters setting 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 Al<sub>2</sub>O<sub>3</sub>: (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 number of nodes used by a job ?

Set the number of CPU nodes used by DP algorithms with:

```
mpirun -np $num_nodes dp
```

Set the number of threads used by DP algorithms with:

```
export OMP_NUM_THREADS=$num_threads
```

Set the number of CPU nodes used by TF kernels with:

```
export TF_INTRA_OP_PARALLELISM_THREADS=$num_nodes
export TF_INTER_OP_PARALLELISM_THREADS=$num_nodes
```

## 12.3 Do we need to set $rcut < \text{half boxsize}$ ?

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

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

PS: The reason why some softwares require  $rcut < \text{half boxsize}$  is that they only consider the nearest mirrors from the center cell. Deepmd-kit is totally 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 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 plugin 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 early stages of MD

This is probably because your structure is too far from the equilibrium configuration.

Although, to make sure the potential model is truly accurate, we recommend to check 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-DEEPM to a LAMMPS source code folder and re-install LAMMPS.

To solve this problem, it suffices to first remove USER-DEEPM from LAMMPS source code by

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

and then install the new USER-DEEPM.

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 training 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 to v0.12.0, but is not compatible to v0.11.0 nor 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
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 v2.1.
- ☹: The model is incompatible with the DeePMD-kit package, and there is no way to convert models.

## CODING CONVENTIONS

### 13.1 Preface

The aim of these coding standards is to help create a codebase with defined and consistent coding style that every contributor can get easily familiar with. This will 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 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 over time 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.

### 13.2 Rules

The code must be compatible with the oldest supported version of python which is 3.6

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 formatting with `"%f" % x`. string format method `"{x:.2f}".format()` may be used sparsely where it is more convenient than f-strings.

## 13.3 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 a space in 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.

## 13.4 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.

## 13.5 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 doctring 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
```

## 13.6 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
```

## 13.7 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
```

## 13.8 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
```

## 13.9 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
```





## 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 module such as data processing, trainer, etc, you may want to read this section.

To incorporate your custom model you'll need to:

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

### 14.1 Design a new component

When creating a new component, take descriptor as the example, you should inherit `deepmd.descriptor.descriptor.Descriptor` class and override several methods. Abstract methods such as `deepmd.descriptor.descriptor.Descriptor.build` must be implemented and others are not. You should keep arguments of these methods unchanged.

After implementation, you need to register the component with a key:

```
from deepmd.descriptor import Descriptor

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

### 14.2 Register new arguments

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

```
from typing import List

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

(continues on next page)

(continued from previous page)

```
@descript_args_plugin.register("some_descript")
def descript_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_descript",
    "arg1": true,
    "arg2": 6.0
}
```

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

## 14.3 Package new codes

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

```
entry_points={
    'deepmd': [
        'some_descript=deepmd_some_descriptpt:SomeDescript',
    ],
},
```

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

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

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

## ATOM TYPE EMBEDDING

### 15.1 Overview

Here is an overview of the deepmd-kit algorithm. Given a specific centric atom, we can obtain the matrix describing its local environment, named as  $\mathcal{R}$ . It is consist of the distance between 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 embed 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 build **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.

### 15.2 Preliminary

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

$i$ : Type of centric atom

$j$ : Type of neighbor atom

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

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

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

$\text{Multi}(\cdot)$ : 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, output is type embedding vector of dim `nchan1`

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

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

Deepmd-kit applying atom type embedding:

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

or

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

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

## 15.3 How to use

A detailed introduction can be found at [se\\_e2\\_a\\_tebd](#). Looking for a fast start up, you can simply add a `type_embedding` section in the input json file as displayed in the following, and the algorithm will adopt atom type embedding algorithm automatically. An example of `type_embedding` is like

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

## 15.4 Code Modification

Atom type embedding can be applied to varied `embedding net` and `fitting net`, as a result we build a class `TypeEmbedNet` to support this free combination. In the following, we will go through the execution process of the code to explain our code modification.

### 15.4.1 trainer (train/trainer.py)

In `trainer.py`, it will parse the parameter from the input json file. If a `type_embedding` section is detected, it will build a `TypeEmbedNet`, which will be later input in the `model`. `model` will be built in the function `_build_network`.

### 15.4.2 model (model/ener.py)

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

### 15.4.3 embedding net (descriptor/se\*.py)

In `embedding net`, we shall take local environment  $\mathcal{R}$  as input and output matrix  $\mathcal{G}$ . Functions called in this process by 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(·)`, 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 using the atom type embedding vector of centric atom according to the value of `type_one_side` (if set True, then we only use the vector of the neighbor atom). The modified input will be put into the `fitting_net` to get  $\mathcal{G}$  for further matrix multiplication stage.

#### 15.4.4 fitting net (fit/ener.py)

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

P.S.: You can't apply compression method while using atom type embedding



## PYTHON API

### 16.1 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, deepmd.utils.batch_size.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.

#### Methods

<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	Sort atoms in the system according their types.

`load_prefix`: `str`

`make_natoms_vec(atom_types: numpy.ndarray) → numpy.ndarray`

Make the natom vector used by deepmd-kit.

#### Parameters

`atom_types` The type of atoms

Returns

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

**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: `numpy.ndarray`, imap: `List[int]`) → `numpy.ndarray`

Reverse mapping of a vector according to the index map

Parameters

vec Input vector. Be of shape [nframes, natoms, -1]

imap Index map. Be of shape [natoms]

Returns

**vec\_out** Reverse mapped vector.

**static sort\_input**(coord: `numpy.ndarray`, atom\_type: `numpy.ndarray`, sel\_atoms: `Optional[List[int]]` = None)

Sort atoms in the system according their types.

Parameters

coord The coordinates of atoms. Should be of shape [nframes, natoms, 3]

atom\_type The type of atoms Should be of shape [natoms]

sel\_atom The selected atoms by type

Returns

**coord\_out** The coordinates after sorting

**atom\_type\_out** The atom types after sorting

**idx\_map** The index mapping from the input to the output. For example coord\_out = coord[:,idx\_map,:]

**sel\_atom\_type** Only output if sel\_atoms is not None The sorted selected atom types

**sel\_idx\_map** Only output if sel\_atoms is not None The index mapping from the selected atoms to sorted selected atoms.

**deepmd.DeepPotential**(model\_file: `Union[str, pathlib.Path]`, load\_prefix: `str` = 'load', default\_tf\_graph: `bool` = False) → `Union[deepmd.infer.deep_dipole.DeepDipole, deepmd.infer.deep_polar.DeepGlobalPolar, deepmd.infer.deep_polar.DeepPolar, deepmd.infer.deep_pot.DeepPot, deepmd.infer.deep_wfc.DeepWFC]`

Factory function that will initialize 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 implemented potential

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

Bases: `deepmd.infer.deep_dipole.DeepDipole`

Parameters

model\_name The model file for the DeepDipole model

model\_charge\_map Gives the amount of charge for the wfcc

sys\_charge\_map Gives the amount of charge for the real atoms

ewald\_h Grid spacing of the reciprocal part of Ewald sum. Unit: Å

ewald\_beta Splitting parameter of the Ewald sum. Unit: Å<sup>-1</sup>

Attributes

**model\_type** Get type of model.

**model\_version** Get version of model.

## Methods

<code>build_fv_graph()</code>	Build the computational graph for the force and virial inference.
<code>eval(coord, box, atype[, eval_fv])</code>	Evaluate the modification
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>get_dim_apham()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>modify_data(data)</code>	Modify data.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	Sort atoms in the system according their types.

`build_fv_graph()` → `tensorflow.python.framework.ops.Tensor`

Build the computational graph for the force and virial inference.

`eval(coord: numpy.ndarray, box: numpy.ndarray, atype: numpy.ndarray, eval_fv: bool = True)` → `Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]`

Evaluate the modification

Parameters

`coord` The coordinates of atoms

`box` The simulation region. PBC is assumed

`atype` The atom types

`eval_fv` Evaluate force and virial

Returns

`tot_e` The energy modification

`tot_f` The force modification

`tot_v` The virial modification

`load_prefix: str`

`modify_data(data: dict)` → `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

### 16.1.1 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](https://github.com/deepsense-ai/tensorflow_on_slurm) ####

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

Get SLURM resources: nodename, nodelist, and gpus.

Returns

`Tuple[str, List[str], Optional[List[int]]]` nodename, nodelist, and gpus

Raises

`RuntimeError` if number of nodes could not be retrieved

`ValueError` list of nodes is not of the same length as a number of nodes

`ValueError` if current nodename is not found in node list

### deepmd.descriptor package

## Submodules

### deepmd.descriptor.descriptor module

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

Bases: `deepmd.utils.plugin.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.DescriptSeA'>
```

## Methods

<i>build</i> (coord_, atype_, natoms, box_, mesh, ...)	Build the computational graph for the descriptor.
<i>compute_input_stats</i> (data_coord, data_box, ...)	Compute the statistics (avg and std) of the training data.
<i>enable_compression</i> (min_nbor_dist[, ...])	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_dim_out</i> ()	Returns the output dimension of this descriptor.
<i>get_dim_rot_mat_1</i> ()	Returns the first dimension of the rotation matrix.
<i>get_feed_dict</i> (coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor
<i>get_nlist</i> ()	Returns neighbor information.
<i>get_ntypes</i> ()	Returns the number of atom types.
<i>get_rcut</i> ()	Returns the cut-off radius.
<i>get_tensor_names</i> ([suffix])	Get names of tensors.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the embedding net variables with the given dict
<i>pass_tensors_from_frz_model</i> (*tensors)	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz graph_def
<i>prod_force_virial</i> (atom_ener, natoms)	Compute force and virial.
<i>register</i> (key)	Register a descriptor plugin.

```
abstract build(coord_: tensorflow.python.framework.ops.Tensor, atype_:
    tensorflow.python.framework.ops.Tensor, natoms:
    tensorflow.python.framework.ops.Tensor, box_:
    tensorflow.python.framework.ops.Tensor, mesh:
    tensorflow.python.framework.ops.Tensor, input_dict: Dict[str, Any], reuse:
    Optional[bool] = None, suffix: str = "") → tensorflow.python.framework.ops.Tensor
```

Build the computational graph for the descriptor.

Parameters

coord\_ [[tf.Tensor](#)] The coordinate of atoms

atype\_ [[tf.Tensor](#)] The type of atoms

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

box [`tf.Tensor`] The box of frames

mesh [`tf.Tensor`] For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input\_dict [`dict[str, Any]`] Dictionary for additional inputs

reuse [`bool`, optional] The weights in the networks should be reused when get the variable.

suffix [`str`, optional] Name suffix to identify this descriptor

Returns

descriptor: `tf.Tensor` The output descriptor

### Notes

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

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

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

Parameters

data\_coord [`list[np.ndarray]`] The coordinates. Can be generated by `deepmd.model.model_stat.make_stat_input()`

data\_box [`list[np.ndarray]`] The box. Can be generated by `deepmd.model.model_stat.make_stat_input()`

data\_atype [`list[np.ndarray]`] The atom types. Can be generated by `deepmd.model.model_stat.make_stat_input()`

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

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

input\_dict [`dict[str, list[np.ndarray]]`] Dictionary for additional input

### Notes

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

```
enable_compression(min_nbor_dist: float, model_file: str = 'frozon_model.pb', table_extrapolate:
                    float = 5.0, table_stride_1: float = 0.01, table_stride_2: float = 0.1,
                    check_frequency: int = -1, suffix: str = '') → None
```

Reveive the statistics (distance, max\_nbor\_size and env\_mat\_range) of the training data.

Parameters

min\_nbor\_dist [`float`] The nearest distance between atoms

`model_file` [`str`, default: 'frozen\_model.pb'] The original frozen model, which will be compressed by the program  
`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\_: `tensorflow.python.framework.ops.Tensor`, atype\_: `tensorflow.python.framework.ops.Tensor`, natoms: `tensorflow.python.framework.ops.Tensor`, box: `tensorflow.python.framework.ops.Tensor`, mesh: `tensorflow.python.framework.ops.Tensor`) → `Dict[str, tensorflow.python.framework.ops.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]`:  $2 \leq i < Ntypes + 2$ , 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[tensorflow.python.framework.ops.Tensor, tensorflow.python.framework.ops.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: tensorflow.python.framework.ops.Graph, graph_def:
               tensorflow.core.framework.graph_pb2.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: tensorflow.python.framework.ops.Tensor) → None
```

Pass the descrtpt\_reshape tensor as well as descrtpt\_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: tensorflow.python.framework.ops.Tensor, natoms:
                           tensorflow.python.framework.ops.Tensor) →
                           Tuple[tensorflow.python.framework.ops.Tensor,
                                tensorflow.python.framework.ops.Tensor,
                                tensorflow.python.framework.ops.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) → deepmd.descriptor.descriptor.Descriptor
```

Register a descriptor plugin.

Parameters

key [[str](#)] the key of a descriptor

Returns



*Descriptor* the registered descriptor

### Examples

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

### deepmd.descriptor.hybrid module

`class deepmd.descriptor.hybrid.DescriptHybrid(*args, **kwargs)`

Bases: `deepmd.descriptor.descriptor.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

<i>build</i> (coord_, atype_, natoms, box_, mesh, ...)	Build the computational graph for the descriptor
<i>compute_input_stats</i> (data_coord, data_box, ...)	Compute the statisitcs (avg and std) of the training data.
<i>enable_compression</i> (min_nbor_dist[, ...])	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_dim_out</i> ()	Returns the output dimension of this descriptor
<i>get_dim_rot_mat_1</i> ()	Returns the first dimension of the rotation matrix.
<i>get_feed_dict</i> (coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor
<i>get_nlist</i> ()	Returns neighbor information.
<i>get_nlist_i</i> (ii)	Get the neighbor information of the ii-th descriptor
<i>get_ntypes</i> ()	Returns the number of atom types
<i>get_rcut</i> ()	Returns the cut-off radius
<i>get_tensor_names</i> ([suffix])	Get names of tensors.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the embedding net variables with the given dict
<i>pass_tensors_from_frz_model</i> (*tensors)	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def
<i>prod_force_virial</i> (atom_ener, natoms)	Compute force and virial
<i>register</i> (key)	Register a descriptor plugin.

**build**(coord\_: tensorflow.python.framework.ops.Tensor, atype\_: tensorflow.python.framework.ops.Tensor, natoms: tensorflow.python.framework.ops.Tensor, box\_: tensorflow.python.framework.ops.Tensor, mesh: tensorflow.python.framework.ops.Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → tensorflow.python.framework.ops.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]`:  $2 \leq i < Ntypes + 2$ , number of type  $i$  atoms

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

Parameters

`data_coord` The coordinates. Can be generated by `deepmd.model.make_stat_input`

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

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

`natoms_vec` The vector for the number of atoms of the system and different types of atoms. Can be generated by `deepmd.model.make_stat_input`

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

`input_dict` Dictionary for additional input

**enable\_compression**(`min_nbor_dist`: float, `model_file`: str = 'frozen\_model.pb', `table_extrapolate`: float = 5.0, `table_stride_1`: float = 0.01, `table_stride_2`: float = 0.1, `check_frequency`: int = -1, `suffix`: str = '') → None

Reveive the statistics (distance, max\_nbor\_size and env\_mat\_range) of the training data.

Parameters

`min_nbor_dist` [float] The nearest distance between atoms

`model_file` [str, default: 'frozen\_model.pb'] The original frozen model, which will be compressed by the program

`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**  
 Reveal 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\_i**(ii: **int**) → **Tuple**[tensorflow.python.framework.ops.Tensor, tensorflow.python.framework.ops.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: tensorflow.python.framework.ops.Graph, graph\_def: tensorflow.core.framework.graph\_pb2.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**(\*tensors: tensorflow.python.framework.ops.Tensor) → **None**  
 Pass the descrt\_reshape tensor as well as descrt\_deriv tensor from the frz graph\_def

Parameters

\*tensors [**tf.Tensor**] passed tensors

```
prod_force_virial(atom_ener: tensorflow.python.framework.ops.Tensor, natoms:
    tensorflow.python.framework.ops.Tensor) →
    Tuple[tensorflow.python.framework.ops.Tensor,
    tensorflow.python.framework.ops.Tensor,
    tensorflow.python.framework.ops.Tensor]
```

Compute force and virial

Parameters

**atom\_ener** The atomic energy

**natoms** The number of atoms. This tensor has the length of  $N_{\text{types}} + 2$ . `natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  $2 \leq i < N_{\text{types}} + 2$ , 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.DescriptLocFrame(*args, **kwargs)
```

Bases: *deepmd.descriptor.descriptor.Descriptor*

Defines a local frame at each atom, and the compute the descriptor as local coordinates under this frame.

Parameters

**rcut** The cut-off radius

**sel\_a** [*list*[*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 first 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]`: 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.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist[, ...])</code>	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_feed_dict(coord_, atype_, natoms, box, mesh)</code>	Generate the feed_dict for current descriptor
<code>get_nlist()</code>	Returns
<code>get_ntypes()</code>	Returns the number of atom types
<code>get_rcut()</code>	Returns the cut-off radius
<code>get_rot_mat()</code>	Get rotational matrix
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict
<code>pass_tensors_from_frz_model(*tensors)</code>	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz_graph_def
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial
<code>register(key)</code>	Register a descriptor plugin.

`build(coord_: tensorflow.python.framework.ops.Tensor, atype_: tensorflow.python.framework.ops.Tensor, natoms: tensorflow.python.framework.ops.Tensor, box_: tensorflow.python.framework.ops.Tensor, mesh: tensorflow.python.framework.ops.Tensor, input_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → tensorflow.python.framework.ops.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]`:  $2 \leq i < Ntypes+2$ , number of type *i* atoms

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

Parameters

data\_coord The coordinates. Can be generated by `deepmd.model.make_stat_input`

data\_box The box. Can be generated by `deepmd.model.make_stat_input`

data\_atype The atom types. Can be generated by `deepmd.model.make_stat_input`

natoms\_vec The vector for the number of atoms of the system and different types of atoms. Can be generated by `deepmd.model.make_stat_input`

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

input\_dict Dictionary for additional input

**get\_dim\_out**() → int

Returns the output dimension of this descriptor

**get\_nlist**() → Tuple[`tensorflow.python.framework.ops.Tensor`, `tensorflow.python.framework.ops.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**() → `tensorflow.python.framework.ops.Tensor`

Get rotational matrix

**init\_variables**(graph: `tensorflow.python.framework.ops.Graph`, graph\_def: `tensorflow.core.framework.graph_pb2.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: tensorflow.python.framework.ops.Tensor, natoms:
    tensorflow.python.framework.ops.Tensor) →
    Tuple[tensorflow.python.framework.ops.Tensor,
    tensorflow.python.framework.ops.Tensor,
    tensorflow.python.framework.ops.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.DescriptSe(*args, **kwargs)
```

Bases: *deepmd.descriptor.descriptor.Descriptor*

A base class for smooth version of descriptors.

## Notes

All of these descriptors have an environmental matrix and an embedding network (*deepmd.utils.network.embedding\_net()*), so they can share some similar methods without defining them twice.

Attributes

**embedding\_net\_variables** [*dict*] initial embedding network variables

**descript\_reshape** [*tf.Tensor*] the reshaped descriptor

**descript\_deriv** [*tf.Tensor*] the descriptor derivative

**rij** [*tf.Tensor*] distances between two atoms

**nlist** [*tf.Tensor*] the neighbor list

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist[, ...])</code>	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_feed_dict(coord_, atype_, natoms, box, mesh)</code>	Generate the feed_dict for current descriptor
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	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: `tensorflow.python.framework.ops.Graph`, graph\_def: `tensorflow.core.framework.graph_pb2.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: `tensorflow.python.framework.ops.Tensor`, descrpt\_deriv: `tensorflow.python.framework.ops.Tensor`, rij: `tensorflow.python.framework.ops.Tensor`, nlist: `tensorflow.python.framework.ops.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



descript\_deriv The passed descript\_deriv tensor  
 rij The passed rij tensor  
 nlist The passed nlist tensor

**property precision:** `tensorflow.python.framework.dtypes.DType`  
 Precision of filter network.

## deepmd.descriptor.se\_a module

**class** `deepmd.descriptor.se_a.DescriptSeA(*args, **kwargs)`

Bases: `deepmd.descriptor.se.DescriptSe`

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor  $\mathcal{D}^i \in \mathbb{R}^{M_1 \times M_2}$  is given by [1]

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}_{<}^i$$

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

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

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

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

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  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  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”.

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

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

## References

[1]

Attributes

**precision** Precision of filter network.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statisitcs (avg and std) of the training data.
<code>enable_compression(min_nbor_dist[, ...])</code>	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_feed_dict(coord_, atype_, natoms, box, mesh)</code>	Generate the feed_dict for current descriptor
<code>get_nlist()</code>	Returns
<code>get_ntypes()</code>	Returns the number of atom types
<code>get_rcut()</code>	Returns the cut-off radius
<code>get_rot_mat()</code>	Get rotational matrix
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial
<code>register(key)</code>	Register a descriptor plugin.

```

build(coord_: tensorflow.python.framework.ops.Tensor, atype_:
    tensorflow.python.framework.ops.Tensor, natoms: tensorflow.python.framework.ops.Tensor,
    box_: tensorflow.python.framework.ops.Tensor, mesh:
    tensorflow.python.framework.ops.Tensor, input_dict: dict, reuse: Optional[bool] = None,
    suffix: str = '') → tensorflow.python.framework.ops.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]:  $2 \leq i < \text{Ntypes} + 2$ , number of type i atoms

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, model_file: str = 'frozen_model.pb', 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

model\_file The original frozen model, which will be compressed by the program

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  
 Reveal 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[tensorflow.python.framework.ops.Tensor,  
 tensorflow.python.framework.ops.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**() → tensorflow.python.framework.ops.Tensor  
 Get rotational matrix  
**init\_variables**(graph: tensorflow.python.framework.ops.Graph, graph\_def:  
 tensorflow.core.framework.graph\_pb2.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: tensorflow.python.framework.ops.Tensor, natoms:  
 tensorflow.python.framework.ops.Tensor) →  
 Tuple[tensorflow.python.framework.ops.Tensor,  
 tensorflow.python.framework.ops.Tensor,  
 tensorflow.python.framework.ops.Tensor]  
 Compute force and virial  
 Parameters  
 atom\_ener The atomic energy

**natoms** The number of atoms. This tensor has the length of  $N_{\text{types}} + 2$  `natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  $2 \leq i < N_{\text{types}} + 2$ , 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.DescriptSeAEbd(*args, **kwargs)`

Bases: `deepmd.descriptor.se_a.DescriptSeA`

DeepPot-SE descriptor with type embedding approach.

Parameters

**rcut** The cut-off radius  
**rcut\_smth** From where the environment matrix should be smoothed  
**sel** [`list[str]`] `sel[i]` specifies the maximum number of type  $i$  atoms in the cut-off radius  
**neuron** [`list[int]`] Number of neurons in each hidden layers of the embedding net  
**axis\_neuron** Number of the axis neuron (number of columns of the sub-matrix of the embedding matrix)  
**resnet\_dt** Time-step  $dt$  in the resnet construction:  $y = x + dt * \phi(Wx + b)$   
**trainable** If the weights of embedding net are trainable.  
**seed** Random seed for initializing the network parameters.  
**type\_one\_side** Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  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

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist[, ...])</code>	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_feed_dict(coord_, atype_, natoms, box, mesh)</code>	Generate the feed_dict for current descriptor
<code>get_nlist()</code>	Returns
<code>get_ntypes()</code>	Returns the number of atom types
<code>get_rcut()</code>	Returns the cut-off radius
<code>get_rot_mat()</code>	Get rotational matrix
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial
<code>register(key)</code>	Register a descriptor plugin.

**build**(coord\_: tensorflow.python.framework.ops.Tensor, atype\_: tensorflow.python.framework.ops.Tensor, natoms: tensorflow.python.framework.ops.Tensor, box\_: tensorflow.python.framework.ops.Tensor, mesh: tensorflow.python.framework.ops.Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → tensorflow.python.framework.ops.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

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.DescriptSeAEf(*args, **kwargs)
```

Bases: *deepmd.descriptor.descriptor.Descriptor*

## Parameters

`rcut` The cut-off radius

`rcut_smth` From where the environment matrix should be smoothed

`sel` [`list[str]`] `sel[i]` specifies the maximum number of type `i` atoms in the cut-off radius

`neuron` [`list[int]`] Number of neurons in each hidden layers of the embedding net

`axis_neuron` Number of the axis neuron (number of columns of the sub-matrix of the embedding matrix)

`resnet_dt` Time-step `dt` in the resnet construction:  $y = x + dt * \phi(Wx + b)$

`trainable` If the weights of embedding net are trainable.

`seed` Random seed for initializing the network parameters.

`type_one_side` Try to build `N_types` embedding nets. Otherwise, building `N_types^2` embedding nets

`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”.

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

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

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist[, ...])</code>	Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Receive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_feed_dict(coord_, atype_, natoms, box, mesh)</code>	Generate the feed_dict for current descriptor
<code>get_nlist()</code>	Returns
<code>get_ntypes()</code>	Returns the number of atom types
<code>get_rcut()</code>	Returns the cut-off radius
<code>get_rot_mat()</code>	Get rotational matrix
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict
<code>pass_tensors_from_frz_model(*tensors)</code>	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz graph_def
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial
<code>register(key)</code>	Register a descriptor plugin.

**build**(coord\_: tensorflow.python.framework.ops.Tensor, atype\_: tensorflow.python.framework.ops.Tensor, natoms: tensorflow.python.framework.ops.Tensor, box\_: tensorflow.python.framework.ops.Tensor, mesh: tensorflow.python.framework.ops.Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → tensorflow.python.framework.ops.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

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

Parameters

data\_coord The coordinates. Can be generated by `deepmd.model.make_stat_input`

data\_box The box. Can be generated by `deepmd.model.make_stat_input`

data\_atype The atom types. Can be generated by `deepmd.model.make_stat_input`

natoms\_vec The vector for the number of atoms of the system and different types of atoms. Can be generated by `deepmd.model.make_stat_input`

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

input\_dict Dictionary for additional input

**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[`tensorflow.python.framework.ops.Tensor`, `tensorflow.python.framework.ops.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**() → `tensorflow.python.framework.ops.Tensor`

Get rotational matrix

**prod\_force\_virial**(atom\_ener: `tensorflow.python.framework.ops.Tensor`, natoms: `tensorflow.python.framework.ops.Tensor`) → Tuple[`tensorflow.python.framework.ops.Tensor`, `tensorflow.python.framework.ops.Tensor`, `tensorflow.python.framework.ops.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.DescriptSeAEfLower(*args, **kwargs)`

Bases: `deepmd.descriptor.se_a.DescriptSeA`

Helper class for implementing DescriptSeAEf

Attributes

**precision** Precision of filter network.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statisitcs (avg and std) of the training data.
<code>enable_compression(min_nbor_dist[, ...])</code>	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_feed_dict(coord_, atype_, natoms, box, mesh)</code>	Generate the feed_dict for current descriptor
<code>get_nlist()</code>	Returns
<code>get_ntypes()</code>	Returns the number of atom types
<code>get_rcut()</code>	Returns the cut-off radius
<code>get_rot_mat()</code>	Get rotational matrix
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial
<code>register(key)</code>	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

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\_r module

**class** `deepmd.descriptor.se_r.DescriptSeR(*args, **kwargs)`

Bases: `deepmd.descriptor.se.DescriptSe`

DeepPot-SE constructed from radial information of atomic configurations.

The embedding takes the distance between atoms as input.

Parameters

rcut The cut-off radius

rcut\_smth From where the environment matrix should be smoothed

sel [`list[str]`] sel[i] specifies the maximum number of type i atoms in the cut-off radius

neuron [`list[int]`] Number of neurons in each hidden layers of the embedding net

resnet\_dt Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

trainable If the weights of embedding net are trainable.

seed Random seed for initializing the network parameters.

type\_one\_side Try to build  $N_{types}$  embedding nets. Otherwise, building  $N_{types}^2$  embedding nets

exclude\_types [`List[List[int]]`] The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

**activation\_function** The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”.

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

**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

<i>build</i> (coord_, atype_, natoms, box_, mesh, ...)	Build the computational graph for the descriptor
<i>compute_input_stats</i> (data_coord, data_box, ...)	Compute the statisitcs (avg and std) of the training data.
<i>enable_compression</i> (min_nbor_dist[, ...])	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_dim_out</i> ()	Returns the output dimension of this descriptor
<i>get_dim_rot_mat_1</i> ()	Returns the first dimension of the rotation matrix.
<i>get_feed_dict</i> (coord_, atype_, natoms, box, mesh)	Generate the feed_dict for current descriptor
<i>get_nlist</i> ()	Returns
<i>get_ntypes</i> ()	Returns the number of atom types
<i>get_rcut</i> ()	Returns the cut-off radius
<i>get_tensor_names</i> ([suffix])	Get names of tensors.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the embedding net variables with the given dict
<i>pass_tensors_from_frz_model</i> (descript_reshape, ...)	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz_graph_def
<i>prod_force_virial</i> (atom_ener, natoms)	Compute force and virial
<i>register</i> (key)	Register a descriptor plugin.

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

**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, model\_file: str = 'frozen\_model.pb', 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

**model\_file** The original frozen model, which will be compressed by the program

**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

**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

**prod\_force\_virial**(atom\_ener: tensorflow.python.framework.ops.Tensor, natoms: tensorflow.python.framework.ops.Tensor) →  
**tuple**[tensorflow.python.framework.ops.Tensor,  
tensorflow.python.framework.ops.Tensor,  
tensorflow.python.framework.ops.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.DescriptSeT(\*args, \*\*kwargs)  
Bases: *deepmd.descriptor.se.DescriptSe*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes angles between two neighboring atoms as input.

Parameters

**rcut** The cut-off radius

**rcut\_smth** From where the environment matrix should be smoothed

**sel** [list[str]] sel[i] specifies the maximum number of type i atoms in the cut-off radius

**neuron** [list[int]] Number of neurons in each hidden layers of the embedding net

**resnet\_dt** Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

**trainable** If the weights of embedding net are trainable.

**seed** Random seed for initializing the network parameters.

**set\_davg\_zero** Set the shift of embedding net input to zero.

**activation\_function** The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”.

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

`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

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist[, ...])</code>	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_feed_dict(coord_, atype_, natoms, box, mesh)</code>	Generate the feed_dict for current descriptor
<code>get_nlist()</code>	Returns
<code>get_ntypes()</code>	Returns the number of atom types
<code>get_rcut()</code>	Returns the cut-off radius
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial
<code>register(key)</code>	Register a descriptor plugin.

`build(coord_: tensorflow.python.framework.ops.Tensor, atype_: tensorflow.python.framework.ops.Tensor, natoms: tensorflow.python.framework.ops.Tensor, box_: tensorflow.python.framework.ops.Tensor, mesh: tensorflow.python.framework.ops.Tensor, input_dict: dict, reuse: Optional[bool] = None, suffix: str = '')` → tensorflow.python.framework.ops.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]`:  $2 \leq i < \text{Ntypes} + 2$ , number of type  $i$  atoms

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

Parameters

data\_coord The coordinates. Can be generated by `deepmd.model.make_stat_input`

data\_box The box. Can be generated by `deepmd.model.make_stat_input`

data\_atype The atom types. Can be generated by `deepmd.model.make_stat_input`

natoms\_vec The vector for the number of atoms of the system and different types of atoms. Can be generated by `deepmd.model.make_stat_input`

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

input\_dict Dictionary for additional input

**enable\_compression**(min\_nbor\_dist: float, model\_file: str = 'frozen\_model.pb', table\_extrapolate: float = 5, table\_stride\_1: float = 0.01, table\_stride\_2: float = 0.1, check\_frequency: int = -1, suffix: str = '') → None

Reveive the statistics (distance, max\_nbor\_size and env\_mat\_range) of the training data.

Parameters

min\_nbor\_dist The nearest distance between atoms

model\_file The original frozen model, which will be compressed by the program

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[`tensorflow.python.framework.ops.Tensor`, `tensorflow.python.framework.ops.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

`prod_force_virial(atom_ener: tensorflow.python.framework.ops.Tensor, natoms: tensorflow.python.framework.ops.Tensor) → Tuple[tensorflow.python.framework.ops.Tensor, tensorflow.python.framework.ops.Tensor, tensorflow.python.framework.ops.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]`:  $2 \leq i < Ntypes+2$ , 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  $10 * \text{step}$  as its uniform step size. The range of the first table is automatically detected by the code, while the second table ranges from the first table's upper boundary(`upper`) to the `extrapolate(parameter) * upper`.

Parameters

`input` [`str`] frozen model file to compress

`output` [`str`] compressed model filename

`extrapolate` [`int`] scale of model extrapolation

`step` [`float`] uniform step size of the tabulation's first table

`frequency` [`str`] frequency of tabulation overflow check

`checkpoint_folder` [`str`] training checkpoint folder for freezing

`training_script` [`str`] training script of the input frozen model

`mpi_log` [`str`] mpi logging mode for training

`log_path` [`Optional[str]`] if specified log will be written to this file

log\_level [`int`] logging level

`deepmd.entrypoints.config(*, output: str, **kwargs)`

Auto config file generator.

Parameters

output: `str` file to write config file

Raises

`RuntimeError` if user does not input any systems

`ValueError` if output file is of wrong type

`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, **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

`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 / Lammmps). This paramter is used to determine the index in the output file.

`deepmd.entrypoints.neighbor_stat(*, system: str, rcut: float, type_map: List[str], **kwargs)`

Calculate neighbor statistics.

Parameters

system [`str`] system to stat

rcut [`float`] cutoff radius

type\_map [`list`[`str`]] type map

## 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, 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  
set\_prefix [str] string prefix of set  
numb\_test [int] number 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

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, **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

Raises

**RuntimeError** if distributed training job nem is wrong

```
deepmd.entrypoints.transfer(*, old_model: str, raw_model: str, output: str, **kwargs)
```

Transfer operation from old from graph to new prepared raw graph.

Parameters

old\_model [str] frozen old graph model

raw\_model [str] new model that will accept ops from old model

output [str] new model with transfered parameters will be saved to this location

## 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] training checkpoint folder for freezing

training\_script [str] training script of the input frozen model

mpi\_log [str] mpi logging mode for training

log\_path [Optional[str]] if specified log will be written to this file

log\_level [int] logging level

**deepmd.entrypoints.config module**

Quickly create a configuration file for smooth model.

`deepmd.entrypoints.config.config(*, output: str, **kwargs)`

Auto config file generator.

Parameters

output: str file to write config file

Raises

`RuntimeError` if user does not input any systems

`ValueError` if output file is of wrong type

**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 training input arguments to console.

**deepmd.entrypoints.freeze module**

Script for freezing TF trained graph so it can be used with LAMMPS and i-PI.

**References**

<https://blog.metaflow.fr/tensorflow-how-to-freeze-a-model-and-serve-it-with-a-python-api-d4f3596b3adc>

`deepmd.entrypoints.freeze.freeze(*, checkpoint_folder: str, output: str, node_names: Optional[str] = None, nvnm_weight: Optional[str] = None, **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

**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()`

DeePMD-Kit entry point.

Raises

`RuntimeError` if no command was input

`deepmd.entrypoints.main.main_parser() → argparse.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) → argparse.Namespace`

Parse arguments and convert argument strings to objects.

Parameters

`args: List[str]` list of command line arguments, main purpose is testing default option  
None takes arguments from `sys.argv`

Returns

`argparse.Namespace` the populated namespace

**deepmd.entrypoints.neighbor\_stat module**

`deepmd.entrypoints.neighbor_stat.neighbor_stat(*, system: str, rcut: float, type_map: List[str],  
**kwargs)`

Calculate neighbor statistics.

Parameters

`system [str]` system to stat

`rcut [float]` cutoff radius

`type_map [list[str]]` type map

## 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, 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  
set\_prefix [str] string prefix of set  
numb\_test [int] number 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

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, **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

Raises

`RuntimeError` if distributed training job nem is wrong

## deepmd.entrypoints.transfer module

Module used for transferring parameters between models.

`deepmd.entrypoints.transfer.transfer`(\*, `old_model`: `str`, `raw_model`: `str`, `output`: `str`, \*\*kwargs)

Transfer operation from old from graph to new prepared raw graph.

Parameters

`old_model` [`str`] frozen old graph model

`raw_model` [`str`] new model that will accept ops from old model

`output` [`str`] new model with transfered parameters will be saved to this location

## deepmd.fit package

### Submodules

#### deepmd.fit.dipole module

```
class deepmd.fit.dipole.DipoleFittingSeA(descrpt: tensorflow.python.framework.ops.Tensor,
                                         neuron: List[int] = [120, 120, 120], resnet_dt: bool = True,
                                         sel_type: Optional[List[int]] = None, seed: Optional[int] =
                                         None, activation_function: str = 'tanh', precision: str =
                                         'default', uniform_seed: bool = False)
```

Bases: `deepmd.fit.fitting.Fitting`

Fit the atomic dipole with descriptor `se_a`

Parameters

`descrpt` [`tf.Tensor`] The descriptor

`neuron` [`List[int]`] Number of neurons in each hidden layer of the fitting net

`resnet_dt` [`bool`] Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

`sel_type` [`List[int]`] The atom types selected to have an atomic dipole prediction. If is None, all atoms are selected.

`seed` [`int`] Random seed for initializing the network parameters.

`activation_function` [`str`] The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”.



**precision** [`str`] The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”.

**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

<code>build(input_d, rot_mat, natoms[, reuse, suffix])</code>	Build the computational graph for fitting net
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_out_size()</code>	Get the output size.
<code>get_sel_type()</code>	Get selected type
<code>init_variables(graph, graph_def[, suffix])</code>	Init the fitting net variables with the given dict

**build**(input\_d: tensorflow.python.framework.ops.Tensor, rot\_mat: tensorflow.python.framework.ops.Tensor, natoms: tensorflow.python.framework.ops.Tensor, reuse: `bool` = None, suffix: `str` = '') → tensorflow.python.framework.ops.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

**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: tensorflow.python.framework.ops.Graph, graph\_def: tensorflow.core.framework.graph\_pb2.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(descrpt: tensorflow.python.framework.ops.Tensor, neuron:
    List[int] = [120, 120, 120], resnet_dt: bool = True, numb_fparam:
    int = 0, numb_aparam: int = 0, rcond: float = 0.001,
    tot_ener_zero: bool = False, trainable: Optional[List[bool]] =
    None, seed: Optional[int] = None, atom_ener: List[float] = [],
    activation_function: str = 'tanh', precision: str = 'default',
    uniform_seed: bool = False)
```

Bases: `deepmd.fit.fitting.Fitting`

Fitting the energy of the system. The force and the virial can also be trained.

The potential energy  $E$  is a fitting network function of the descriptor  $\mathcal{D}$ :

$$E(\mathcal{D}) = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \dots \circ \mathcal{L}^{(1)} \circ \mathcal{L}^{(0)}$$

The first  $n$  hidden layers  $\mathcal{L}^{(0)}, \dots, \mathcal{L}^{(n-1)}$  are given by

$$y = \mathcal{L}(x; w, b) = \phi(x^T w + b)$$

where  $x \in \mathbb{R}^{N_1}$  is the input vector and  $y \in \mathbb{R}^{N_2}$  is the output vector.  $w \in \mathbb{R}^{N_1 \times N_2}$  and  $b \in \mathbb{R}^{N_2}$  are weights and biases, respectively, both of which are trainable if trainable[i] is True.  $\phi$  is the activation function.

The output layer  $\mathcal{L}^{(n)}$  is given by

$$y = \mathcal{L}^{(n)}(x; w, b) = x^T w + b$$

where  $x \in \mathbb{R}^{N_{n-1}}$  is the input vector and  $y \in \mathbb{R}$  is the output scalar.  $w \in \mathbb{R}^{N_{n-1}}$  and  $b \in \mathbb{R}$  are weights and bias, respectively, both of which are trainable if trainable[n] is True.

#### Parameters

descrpt The descriptor  $\mathcal{D}$   
 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 descriptor should be set.

**activation\_function** The activation function  $\phi$  in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”.

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

**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

<i>build</i> (inputs, natoms[, input_dict, reuse, ...])	Build the computational graph for fitting net
<i>compute_input_stats</i> (all_stat[, protection])	Compute the input statistics
<i>compute_output_stats</i> (all_stat)	Compute the output statistics
<i>enable_compression</i> (model_file[, suffix])	Set the fitting net attributes from the frozen model_file when fparam or aparam is not zero
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_numb_aparam</i> ()	Get the number of atomic parameters
<i>get_numb_fparam</i> ()	Get the number of frame parameters
<i>init_variables</i> (graph, graph_def[, suffix])	Init the fitting net variables with the given dict

**build**(inputs: tensorflow.python.framework.ops.Tensor, natoms: tensorflow.python.framework.ops.Tensor, input\_dict: dict = None, reuse: bool = None, suffix: str = '') → tensorflow.python.framework.ops.Tensor

Build the computational graph for fitting net

#### Parameters

**inputs** The input descriptor

**input\_dict** Additional dict for inputs. if num\_b\_fparam > 0, should have input\_dict['fparam'] if num\_b\_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 num\_b\_fparam > 0 must have all\_stat['fparam'] if num\_b\_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) → None

Compute the output statistics

Parameters

all\_stat must have the following components: all\_stat['energy'] of shape n\_sys x n\_batch x n\_frame can be prepared by model.make\_stat\_input

**enable\_compression**(model\_file: str, suffix: str = '') → None

Set the fitting net attributes from the frozen model\_file when fparam or aparam is not zero

Parameters

model\_file [str] The input frozen model file

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\_numb\_aparam**() → int

Get the number of atomic parameters

**get\_numb\_fparam**() → int

Get the number of frame parameters

**init\_variables**(graph: tensorflow.python.framework.ops.Graph, graph\_def: tensorflow.core.framework.graph\_pb2.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

Bases: object

Attributes

*precision* Precision of fitting network.

## Methods

---

<code>init_variables(graph, graph_def[, suffix])</code>	Init the fitting net variables with the given dict
---	--

---

**init\_variables**(graph: tensorflow.python.framework.ops.Graph, graph\_def: tensorflow.core.framework.graph\_pb2.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:** tensorflow.python.framework.dtypes.DType

Precision of fitting network.

## deepmd.fit.polar module

**class** deepmd.fit.polar.GlobalPolarFittingSeA(descrpt: tensorflow.python.framework.ops.Tensor, neuron: List[int] = [120, 120, 120], resnet\_dt: bool = True, sel\_type: Optional[List[int]] = None, fit\_diag: bool = True, scale: Optional[List[float]] = None, diag\_shift: Optional[List[float]] = None, seed: Optional[int] = None, activation\_function: str = 'tanh', precision: str = 'default')

Bases: object

Fit the system polarizability with descriptor se\_a

Parameters

descrpt [tf.Tensor] The descriptor  
neuron [List[int]] Number of neurons in each hidden layer of the fitting net  
resnet\_dt [bool] Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$   
sel\_type [List[int]] The atom types selected to have an atomic polarizability prediction  
fit\_diag [bool] Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.  
scale [List[float]] The output of the fitting net (polarizability matrix) for type i atom will be scaled by scale[i]  
diag\_shift [List[float]] The diagonal part of the polarizability matrix of type i will be shifted by diag\_shift[i]. The shift operation is carried out after scale.  
seed [int] Random seed for initializing the network parameters.

`activation_function` [`str`] The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”.

`precision` [`str`] The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”.

## Methods

<code>build(input_d, rot_mat, natoms[, reuse, suffix])</code>	Build the computational graph for fitting net
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_out_size()</code>	Get the output size.
<code>get_sel_type()</code>	Get selected atom types
<code>init_variables(graph, graph_def[, suffix])</code>	Init the fitting net variables with the given dict

`build(input_d, rot_mat, natoms, reuse=None, suffix='')` → `tensorflow.python.framework.ops.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]`:  $2 \leq i < Ntypes + 2$ , 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

**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: tensorflow.python.framework.ops.Graph, graph_def: tensorflow.core.framework.graph_pb2.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.PolarFittingLocFrame(jdata, descrpt)
```

Bases: `object`

Fitting polarizability with local frame descriptor.

Deprecated since version 2.0.0: This class is not supported any more.

## Methods

<code>build</code>	
<code>get_out_size</code>	
<code>get_sel_type</code>	

```
build(input_d, rot_mat, natoms, reuse=None, suffix="")
```

```
get_out_size()
```

```
get_sel_type()
```

```
class deepmd.fit.polar.PolarFittingSeA(descrpt: tensorflow.python.framework.ops.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, shift_diag: bool = True, seed:
    Optional[int] = None, activation_function: str = 'tanh',
    precision: str = 'default', uniform_seed: bool = False)
```

Bases: `deepmd.fit.fitting.Fitting`

Fit the atomic polarizability with descriptor `se_a`

### Parameters

`descrpt` [`tf.Tensor`] The descriptor

`neuron` [`List[int]`] Number of neurons in each hidden layer of the fitting net

`resnet_dt` [`bool`] Time-step  $dt$  in the resnet construction:  $y = x + dt * \phi(Wx + b)$

`sel_type` [`List[int]`] The atom types selected to have an atomic polarizability prediction. If is None, all atoms are selected.

`fit_diag` [`bool`] Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.

`scale` [`List[float]`] The output of the fitting net (polarizability matrix) for type  $i$  atom will be scaled by `scale[i]`

`diag_shift` [`List[float]`] The diagonal part of the polarizability matrix of type  $i$  will be shifted by `diag_shift[i]`. The shift operation is carried out after scale.

`seed` [`int`] Random seed for initializing the network parameters.

`activation_function` [`str`] The activation function in the embedding net. Supported options are "relu", "relu6", "softplus", "sigmoid", "tanh", "gelu", "gelu\_tf".

`precision` [`str`] The precision of the embedding net parameters. Supported options are "default", "float16", "float32", "float64".

`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

<i>build</i> (input_d, rot_mat, natoms[, reuse, suffix])	Build the computational graph for fitting net
<i>compute_input_stats</i> (all_stat[, protection])	Compute the input statistics
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_out_size</i> ()	Get the output size.
<i>get_sel_type</i> ()	Get selected atom types
<i>init_variables</i> (graph, graph_def[, suffix])	Init the fitting net variables with the given dict

**build**(input\_d: tensorflow.python.framework.ops.Tensor, rot\_mat: tensorflow.python.framework.ops.Tensor, natoms: tensorflow.python.framework.ops.Tensor, reuse: 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

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: tensorflow.python.framework.ops.Graph, graph_def:
               tensorflow.core.framework.graph_pb2.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.wfc module

```
class deepmd.fit.wfc.WCFFitting(jdata, descrpt)
```

Bases: `object`

Fitting Wannier function centers (WFCs) with local frame descriptor.

Deprecated since version 2.0.0: This class is not supported any more.

## Methods

build	
get_out_size	
get_sel_type	
get_wfc_numb	

```
build(input_d, rot_mat, natoms, reuse=None, suffix='')
```

```
get_out_size()
```

```
get_sel_type()
```

```
get_wfc_numb()
```

## deepmd.infer package

Submodule containing all the implemented potentials.

```
class deepmd.infer.DeepDipole(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool =
                               False)
```

Bases: `deepmd.infer.deep_tensor.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 change the order!

#### Attributes

**model\_type** Get type of model.  
**model\_version** Get version of model.

#### Methods

<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	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,
    deepmd.utils.batch_size.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.

## Methods

<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	Sort atoms in the system according their types.

**load\_prefix:** `str`

**make\_natoms\_vec**(atom\_types: `numpy.ndarray`) → `numpy.ndarray`

Make the natom vector used by deepmd-kit.

Parameters

atom\_types The type of atoms

Returns

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

**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: `numpy.ndarray`, imap: `List[int]`) → `numpy.ndarray`

Reverse mapping of a vector according to the index map

Parameters

vec Input vector. Be of shape [nframes, natoms, -1]

imap Index map. Be of shape [natoms]

Returns

**vec\_out** Reverse mapped vector.

**static sort\_input**(coord: `numpy.ndarray`, atom\_type: `numpy.ndarray`, sel\_atoms: `Optional[List[int]]` = None)

Sort atoms in the system according their types.

Parameters

coord The coordinates of atoms. Should be of shape [nframes, natoms, 3]

atom\_type The type of atoms Should be of shape [natoms]

sel\_atom The selected atoms by type

Returns

**coord\_out** The coordinates after sorting

**atom\_type\_out** The atom types after sorting

**idx\_map** The index mapping from the input to the output. For example `coord_out = coord[:,idx_map,:]`

**sel\_atom\_type** Only output if `sel_atoms` is not `None` The sorted selected atom types

**sel\_idx\_map** Only output if `sel_atoms` is not `None` The index mapping from the selected atoms to sorted selected atoms.

```
class deepmd.infer.DeepGlobalPolar(model_file: str, load_prefix: str = 'load', default_tf_graph: bool = False)
```

Bases: `deepmd.infer.deep_tensor.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.

## Methods

<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	Sort atoms in the system according their types.

**eval**(`coords`: `numpy.ndarray`, `cells`: `numpy.ndarray`, `atom_types`: `List[int]`, `atomic`: `bool` = `False`, `fparam`: `Optional[numpy.ndarray]` = `None`, `aparam`: `Optional[numpy.ndarray]` = `None`, `efield`: `Optional[numpy.ndarray]` = `None`) → `numpy.ndarray`

Evaluate the model.

Parameters

`coords` The coordinates of atoms. The array should be of size `nframes x natoms x 3`

**cells** The cell of the region. If None then non-PBC is assumed, otherwise using PBC.  
The array should be of size nframes x 9

**atom\_types** The atom types The list should contain natoms ints

**atomic** 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: *deepmd.infer.deep\_tensor.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 change the order!

Attributes

**model\_type** Get type of model.

**model\_version** Get version of model.

## Methods

<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	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, deepmd.utils.batch_size.AutoBatchSize]
                           = True)
```

Bases: `deepmd.infer.deep_eval.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 change 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.

## Methods

<i>eval</i> (coords, cells, atom_types[, atomic, ...])	Evaluate the energy, force and virial by using this DP.
<i>eval_descriptor</i> (coords, cells, atom_types[, ...])	Evaluate descriptors by using this DP.
<i>get_dim_aparam</i> ()	Get the number (dimension) of atomic parameters of this DP.
<i>get_dim_fparam</i> ()	Get the number (dimension) of frame parameters of this DP.
<i>get_ntypes</i> ()	Get the number of atom types of this model.
<i>get_rcut</i> ()	Get the cut-off radius of this model.
<i>get_sel_type</i> ()	Unsupported in this model.
<i>get_type_map</i> ()	Get the type map (element name of the atom types) of this model.
<i>make_natoms_vec</i> (atom_types)	Make the natom vector used by deepmd-kit.
<i>reverse_map</i> (vec, imap)	Reverse mapping of a vector according to the index map
<i>sort_input</i> (coord, atom_type[, sel_atoms])	Sort atoms in the system according their types.

**eval**(coords: `numpy.ndarray`, cells: `numpy.ndarray`, atom\_types: `List[int]`, atomic: `bool` = False, fparam: `Optional[numpy.ndarray]` = None, aparam: `Optional[numpy.ndarray]` = None, efield: `Optional[numpy.ndarray]` = None) → `Tuple[numpy.ndarray, ...]`

Evaluate the energy, force and virial by using this DP.

### Parameters

- coords The coordinates of atoms. The array should be of size nframes x natoms x 3
- cells The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9
- atom\_types The atom types The list should contain natoms ints
- atomic Calculate the atomic energy and virial
- fparam The frame parameter. The array can be of size : - nframes x dim\_fparam. - dim\_fparam. Then all frames are assumed to be provided with the same fparam.

**aparam** The atomic parameter The array can be of size : - nframes x natoms x dim\_aparam. - natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam. - dim\_aparam. Then all frames and atoms are provided with the same aparam.

**efield** The external field on atoms. The array should be of size nframes x natoms x 3

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: `numpy.ndarray`, cells: `numpy.ndarray`, atom\_types: `List[int]`, fparam: `Optional[numpy.ndarray]` = None, aparam: `Optional[numpy.ndarray]` = None, efield: `Optional[numpy.ndarray]` = None) → `numpy.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

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\_rcut()** → `float`

Get the cut-off radius of this model.

**get\_sel\_type()** → `List[int]`

Unsupported in this model.



`get_type_map()` → `List[int]`

Get the type map (element name of the atom types) of this model.

`load_prefix`: `str`

`deepmd.infer.DeepPotential`(`model_file`: `Union[str, pathlib.Path]`, `load_prefix`: `str` = 'load',  
`default_tf_graph`: `bool` = False) →  
`Union[deepmd.infer.deep_dipole.DeepDipole,`  
`deepmd.infer.deep_polar.DeepGlobalPolar,`  
`deepmd.infer.deep_polar.DeepPolar, deepmd.infer.deep_pot.DeepPot,`  
`deepmd.infer.deep_wfc.DeepWFC]`

Factory function that will initialize 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 implemented potential

`class deepmd.infer.DeepWFC`(`model_file`: `Path`, `load_prefix`: `str` = 'load', `default_tf_graph`: `bool` = False)

Bases: `deepmd.infer.deep_tensor.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 change the order!

Attributes

`model_type` Get type of model.

`model_version` Get version of model.

## Methods

<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	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: `deepmd.infer.deep_dipole.DeepDipole`

## Parameters

`model_name` The model file for the DeepDipole model

`model_charge_map` Gives the amount of charge for the wfcc

`sys_charge_map` Gives the amount of charge for the real atoms

`ewald_h` Grid spacing of the reciprocal part of Ewald sum. Unit: Å

`ewald_beta` Splitting parameter of the Ewald sum. Unit: Å<sup>-1</sup>

## Attributes

`model_type` Get type of model.

`model_version` Get version of model.

## Methods

<code>build_fv_graph()</code>	Build the computational graph for the force and virial inference.
<code>eval(coord, box, atype[, eval_fv])</code>	Evaluate the modification
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>get_dim_apham()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>modify_data(data)</code>	Modify data.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	Sort atoms in the system according their types.

`build_fv_graph()` → tensorflow.python.framework.ops.Tensor

Build the computational graph for the force and virial inference.

`eval(coord: numpy.ndarray, box: numpy.ndarray, atype: numpy.ndarray, eval_fv: bool = True) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]`

Evaluate the modification

Parameters

`coord` The coordinates of atoms  
`box` The simulation region. PBC is assumed  
`atype` The atom types  
`eval_fv` Evaluate force and virial

Returns

`tot_e` The energy modification  
`tot_f` The force modification  
`tot_v` The virial modification

`load_prefix: str`

`modify_data(data: dict) → 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

```
class deepmd.infer.EwaldRecp(hh, beta)
```

Bases: `object`

Evaluate the reciprocal part of the Ewald sum

## Methods

<code>eval(coord, charge, box)</code>	Evaluate
---------------------------------------	----------

```
eval(coord: numpy.ndarray, charge: numpy.ndarray, box: numpy.ndarray) →  
    Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]
```

Evaluate

Parameters

`coord` The coordinates of atoms

`charge` The atomic charge

`box` The simulation region. PBC is assumed

Returns

**e** The energy

**f** The force

**v** The virial

```
deepmd.infer.calc_model_devi(coord, box, atype, models, fname=None, frequency=1)
```

Python interface to calculate model deviation

Parameters

`coord` [`numpy.ndarray`, `n_frames` x `n_atoms` x 3] Coordinates of system to calculate

`box` [`numpy.ndarray` or `None`, `n_frames` x 3 x 3] Box to specify periodic boundary condition. If `None`, no pbc will be used

`atype` [`numpy.ndarray`, `n_atoms` x 1] Atom types

`models` [list of *DeepPot* models] Models used to evaluate deviation

`fname` [`str` or `None`] File to dump results, default `None`

`frequency` [`int`] Steps between frames (if the system is given by molecular dynamics engine), default 1

Returns

`model_devi` [`numpy.ndarray`, `n_frames` x 7] Model deviation results. The first column is index of steps, the other 6 columns are `max_devi_v`, `min_devi_v`, `avg_devi_v`, `max_devi_f`, `min_devi_f`, `avg_devi_f`.

## 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: *deepmd.infer.deep\_dipole.DeepDipole*

#### Parameters

**model\_name** The model file for the DeepDipole model

**model\_charge\_map** Gives the amount of charge for the wfcc

**sys\_charge\_map** Gives the amount of charge for the real atoms

**ewald\_h** Grid spacing of the reciprocal part of Ewald sum. Unit: Å

**ewald\_beta** Splitting parameter of the Ewald sum. Unit: Å<sup>-1</sup>

#### Attributes

**model\_type** Get type of model.

**model\_version** Get version of model.

## Methods

<code>build_fv_graph()</code>	Build the computational graph for the force and virial inference.
<code>eval(coord, box, atype[, eval_fv])</code>	Evaluate the modification
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>get_dim_apham()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>modify_data(data)</code>	Modify data.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	Sort atoms in the system according their types.

`build_fv_graph()` → tensorflow.python.framework.ops.Tensor

Build the computational graph for the force and virial inference.

`eval(coord: numpy.ndarray, box: numpy.ndarray, atype: numpy.ndarray, eval_fv: bool = True)` → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]

Evaluate the modification

Parameters

`coord` The coordinates of atoms  
`box` The simulation region. PBC is assumed  
`atype` The atom types  
`eval_fv` Evaluate force and virial

Returns

`tot_e` The energy modification  
`tot_f` The force modification  
`tot_v` The virial modification

`load_prefix:` str

`modify_data(data: dict)` → 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

**deepmd.infer.deep\_dipole module**

```
class deepmd.infer.deep_dipole.DeepDipole(model_file: Path, load_prefix: str = 'load',
                                          default_tf_graph: bool = False)
```

Bases: *deepmd.infer.deep\_tensor.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 change the order!

Attributes

`model_type` Get type of model.

`model_version` Get version of model.

**Methods**

<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	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\_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,
    deepmd.utils.batch_size.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.

**Methods**

<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	Sort atoms in the system according their types.

`load_prefix`: `str`

`make_natoms_vec(atom_types: numpy.ndarray) → numpy.ndarray`

Make the natom vector used by deepmd-kit.

**Parameters**

`atom_types` The type of atoms

**Returns**

**natoms** The number of atoms. This tensor has the length of `Ntypes + 2` `natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  $2 \leq i < Ntypes+2$ , 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`: `numpy.ndarray`, `imap`: `List[int]`)  $\rightarrow$  `numpy.ndarray`

Reverse mapping of a vector according to the index map

Parameters

`vec` Input vector. Be of shape `[nframes, natoms, -1]`

`imap` Index map. Be of shape `[natoms]`

Returns

**vec\_out** Reverse mapped vector.

**static** `sort_input`(`coord`: `numpy.ndarray`, `atom_type`: `numpy.ndarray`, `sel_atoms`: `Optional[List[int]] = None`)

Sort atoms in the system according their types.

Parameters

`coord` The coordinates of atoms. Should be of shape `[nframes, natoms, 3]`

`atom_type` The type of atoms Should be of shape `[natoms]`

`sel_atom` The selected atoms by type

Returns

**coord\_out** The coordinates after sorting

**atom\_type\_out** The atom types after sorting

**idx\_map** The index mapping from the input to the output. For example `coord_out = coord[:,idx_map,:]`

**sel\_atom\_type** Only output if `sel_atoms` is not `None` The sorted selected atom types

**sel\_idx\_map** Only output if `sel_atoms` is not `None` The index mapping from the selected atoms to sorted selected atoms.

## deepmd.infer.deep\_polar module

**class** `deepmd.infer.deep_polar.DeepGlobalPolar`(`model_file`: `str`, `load_prefix`: `str = 'load'`, `default_tf_graph`: `bool = False`)

Bases: `deepmd.infer.deep_tensor.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.

## Methods

<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	Sort atoms in the system according their types.

`eval`(coords: `numpy.ndarray`, cells: `numpy.ndarray`, atom\_types: `List[int]`, atomic: `bool` = False, fparam: `Optional[numpy.ndarray]` = None, aparam: `Optional[numpy.ndarray]` = None, efield: `Optional[numpy.ndarray]` = None) → `numpy.ndarray`

Evaluate the model.

Parameters

coords The coordinates of atoms. The array should be of size nframes x natoms x 3  
cells The cell of the region. If None then non-PBC is assumed, otherwise using PBC.  
The array should be of size nframes x 9  
atom\_types The atom types The list should contain natoms ints  
atomic 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: `deepmd.infer.deep_tensor.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 change the order!

#### Attributes

`model_type` Get type of model.

`model_version` Get version of model.

#### Methods

<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	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.DeepPot(model_file: Path, load_prefix: str = 'load', default_tf_graph:
    bool = False, auto_batch_size: Union[bool, int,
    deepmd.utils.batch_size.AutoBatchSize] = True)
```

Bases: `deepmd.infer.deep_eval.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 change 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.

## Methods

<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the energy, force and virial by using this DP.
<code>eval_descriptor(coords, cells, atom_types[, ...])</code>	Evaluate descriptors by using this DP.
<code>get_dim_aparam()</code>	Get the number (dimension) of atomic parameters of this DP.
<code>get_dim_fparam()</code>	Get the number (dimension) of frame parameters of this DP.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Unsupported in this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	Sort atoms in the system according their types.

`eval` (coords: `numpy.ndarray`, cells: `numpy.ndarray`, atom\_types: `List[int]`, atomic: `bool` = False, fparam: `Optional[numpy.ndarray]` = None, aparam: `Optional[numpy.ndarray]` = None, efield: `Optional[numpy.ndarray]` = None) → `Tuple[numpy.ndarray, ...]`

Evaluate the energy, force and virial by using this DP.

#### Parameters

**coords** The coordinates of atoms. The array should be of size  $nframes \times natoms \times 3$

**cells** The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size  $nframes \times 9$

**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 : -  $nframes \times dim\_fparam$ . -  $dim\_fparam$ . Then all frames are assumed to be provided with the same fparam.

**aparam** The atomic parameter The array can be of size : -  $nframes \times natoms \times dim\_aparam$ . -  $natoms \times dim\_aparam$ . 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  $nframes \times natoms \times 3$

#### 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: `numpy.ndarray`, cells: `numpy.ndarray`, atom\_types: `List[int]`, fparam: `Optional[numpy.ndarray]` = None, aparam: `Optional[numpy.ndarray]` = None, efield: `Optional[numpy.ndarray]` = None)  $\rightarrow$  `numpy.array`

Evaluate descriptors by using this DP.

#### Parameters

**coords** The coordinates of atoms. The array should be of size  $nframes \times natoms \times 3$

**cells** The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size  $nframes \times 9$

**atom\_types** The atom types The list should contain  $natoms$  ints

**fparam** The frame parameter. The array can be of size : -  $nframes \times dim\_fparam$ . -  $dim\_fparam$ . Then all frames are assumed to be provided with the same fparam.

**aparam** The atomic parameter The array can be of size : -  $nframes \times natoms \times dim\_aparam$ . -  $natoms \times dim\_aparam$ . 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  $nframes \times natoms \times 3$

#### 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_rcut() → float`

Get the cut-off radius of this model.

`get_sel_type() → List[int]`

Unsupported in this model.

`get_type_map() → List[int]`

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: *deepmd.infer.deep\_eval.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.

## Methods

<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>get_dim_aparam()</code>	Get the number (dimension) of atomic parameters of this DP.
<code>get_dim_fparam()</code>	Get the number (dimension) of frame parameters of this DP.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	Sort atoms in the system according their types.

**eval**(coords: `numpy.ndarray`, cells: `numpy.ndarray`, atom\_types: `List[int]`, atomic: `bool` = True, fparam: `Optional[numpy.ndarray]` = None, aparam: `Optional[numpy.ndarray]` = None, efield: `Optional[numpy.ndarray]` = None) → `numpy.ndarray`

Evaluate the model.

Parameters

**coords** The coordinates of atoms. The array should be of size nframes x natoms x 3  
**cells** The cell of the region. If None then non-PBC is assumed, otherwise using PBC.  
The array should be of size nframes x 9  
**atom\_types** The atom types The list should contain natoms ints  
**atomic** If True (default), return the atomic tensor Otherwise return the global tensor  
**fparam** Not used in this model  
**aparam** Not used in this model  
**efield** Not used in this model

Returns

**tensor** The returned tensor If atomic == False then of size nframes x output\_dim  
else of size nframes x natoms x output\_dim

**eval\_full**(coords: `numpy.ndarray`, cells: `numpy.ndarray`, atom\_types: `List[int]`, atomic: `bool` = False, fparam: `Optional[numpy.array]` = None, aparam: `Optional[numpy.array]` = None, efield: `Optional[numpy.array]` = None) → `Tuple[numpy.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

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[int]`

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': 'descript_attr/ntypes:0', 't_output_dim':
'model_attr/output_dim:0', 't_rcut': 'descript_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: `deepmd.infer.deep_tensor.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 change the order!

#### Attributes

`model_type` Get type of model.

`model_version` Get version of model.

#### Methods

<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types)</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map
<code>sort_input(coord, atom_type[, sel_atoms])</code>	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\_recip module

`class deepmd.infer.ewald_recip.EwaldRecp(hh, beta)`  
Bases: `object`  
Evaluate the reciprocal part of the Ewald sum

## Methods

<code>eval(coord, charge, box)</code>	Evaluate
---------------------------------------	----------

`eval(coord: numpy.ndarray, charge: numpy.ndarray, box: numpy.ndarray) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]`

Evaluate

Parameters

`coord` The coordinates of atoms  
`charge` The atomic charge  
`box` The simulation region. PBC is assumed

Returns

`e` The energy  
`f` The force  
`v` The virial

## deepmd.infer.model\_devi module

`deepmd.infer.model_devi.calc_model_devi(coord, box, atype, models, fname=None, frequency=1)`

Python interface to calculate model deviation

Parameters

`coord` [`numpy.ndarray`, `n_frames` x `n_atoms` x 3] Coordinates of system to calculate  
`box` [`numpy.ndarray` or `None`, `n_frames` x 3 x 3] Box to specify periodic boundary condition. If `None`, no pbc will be used  
`atype` [`numpy.ndarray`, `n_atoms` x 1] Atom types  
`models` [list of DeepPot models] Models used to evaluate deviation  
`fname` [`str` or `None`] File to dump results, default `None`  
`frequency` [`int`] Steps between frames (if the system is given by molecular dynamics engine), default 1

Returns

`model_devi` [`numpy.ndarray`, `n_frames` x 7] Model deviation results. The first column is index of steps, the other 6 columns are `max_devi_v`, `min_devi_v`, `avg_devi_v`, `max_devi_f`, `min_devi_f`, `avg_devi_f`.

## 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: `numpy.ndarray`)

Parameters

es [`numpy.ndarray`] size of `n_models x n_frames x n_atoms`

`deepmd.infer.model_devi.calc_model_devi_f`(fs: `numpy.ndarray`)

Parameters

fs [`numpy.ndarray`] size of `n_models x n_frames x n_atoms x 3`

`deepmd.infer.model_devi.calc_model_devi_v`(vs: `numpy.ndarray`)

Parameters

vs [`numpy.ndarray`] size of `n_models x n_frames x 9`

`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 / Lammmps). This paramter is used to determine the index in the output file.

`deepmd.infer.model_devi.write_model_devi_out`(devi: `numpy.ndarray`, fname: `str`, header: `str` = "")

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>  
[avoid-tensorflow-print-on-standard-error](https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U)      <https://stackoverflow.com/questions/56085015/suppress-openmp-debug-messages-when-running-tensorflow-on-cpu>  
[suppress-openmp-debug-messages-when-running-tensorflow-on-cpu](https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U)

## 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

### Submodules

#### 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: `deepmd.loss.loss.Loss`

## Methods

<code>build</code> (learning_rate, natoms, model_dict, ...)	Build the loss function graph.
<code>eval</code> (sess, feed_dict, natoms)	Eval the loss function.

<code>print_header</code>	
<code>print_on_training</code>	

**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

**static 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: `deepmd.loss.loss.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

<code>build</code> (learning_rate, natoms, model_dict, ...)	Build the loss function graph.
<code>eval</code> (sess, feed_dict, natoms)	Eval the loss function.

<code>print_header</code>	
<code>print_on_training</code>	

**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)

## deepmd.loss.loss module

**class** deepmd.loss.loss.Loss

Bases: `object`

The abstract class for the loss function.

## Methods

<code>build</code> (learning_rate, natoms, model_dict, ...)	Build the loss function graph.
<code>eval</code> (sess, feed_dict, natoms)	Eval the loss function.

```
abstract build(learning_rate: tensorflow.python.framework.ops.Tensor, natoms:
    tensorflow.python.framework.ops.Tensor, model_dict: Dict[str,
    tensorflow.python.framework.ops.Tensor], label_dict: Dict[str,
    tensorflow.python.framework.ops.Tensor], suffix: str) →
    Tuple[tensorflow.python.framework.ops.Tensor, Dict[str,
    tensorflow.python.framework.ops.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: tensorflow.python.client.session.Session, feed_dict:
    Dict[tensorflow.python.ops.array_ops.placeholder,
    tensorflow.python.framework.ops.Tensor], natoms:
    tensorflow.python.framework.ops.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: *deepmd.loss.loss.Loss*

Loss function for tensorial properties.

**Methods**

<i>build</i> (learning_rate, natoms, model_dict, ...)	Build the loss function graph.
<i>eval</i> (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)
```

## deepmd.model package

## Submodules

## 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)
```

Bases: `deepmd.model.model.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  $(N_t + 1) * N_t / 2 + 1$  columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

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

sw\_rmin The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when `use_srtab` is provided.

sw\_rmin The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when `use_srtab` is provided.

## Methods

---

<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model
--	--

---

build	
data_stat	
get_ntypes	
get_rcut	
get_type_map	

```

build(coord_, atype_, natoms, box, mesh, input_dict, frz_model=None, suffix="", reuse=None)

data_stat(data)

get_ntypes()

get_rcut()

get_type_map()

init_variables(graph: tensorflow.python.framework.ops.Graph, graph_def:
               tensorflow.core.framework.graph_pb2.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'

```

## deepmd.model.model module

```
class deepmd.model.model.Model
```

Bases: `object`

### Methods

<i>init_variables</i> (graph, graph_def[, ...])	Init the embedding net variables with the given frozen model
---	--

```

init_variables(graph: tensorflow.python.framework.ops.Graph, graph_def:
               tensorflow.core.framework.graph_pb2.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

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.tensor module**

```
class deepmd.model.tensor.DipoleModel(descrpt, fitting, type_map: Optional[List[str]] = None,
                                     data_stat_nbatch: int = 10, data_stat_protect: float = 0.01)
```

Bases: *deepmd.model.tensor.TensorModel*

**Methods**


---

<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model
--	--

---

build	
data_stat	
get_ntypes	
get_out_size	
get_rcut	
get_sel_type	
get_type_map	

```
class deepmd.model.tensor.GlobalPolarModel(descrpt, fitting, type_map: Optional[List[str]] = None,
                                           data_stat_nbatch: int = 10, data_stat_protect: float = 0.01)
```

Bases: *deepmd.model.tensor.TensorModel*

## Methods

---

<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model
--	--

---

build	
data_stat	
get_ntypes	
get_out_size	
get_rcut	
get_sel_type	
get_type_map	

```
class deepmd.model.tensor.PolarModel(descrpt, fitting, type_map: Optional[List[str]] = None,
                                     data_stat_nbatch: int = 10, data_stat_protect: float = 0.01)
```

Bases: *deepmd.model.tensor.TensorModel*

## Methods

---

<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model
--	--

---

build	
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, type_map:
                                     Optional[List[str]] = None, data_stat_nbatch: int = 10,
                                     data_stat_protect: float = 0.01)
```

Bases: *deepmd.model.model.Model*

Tensor model.

Parameters

tensor\_name Name of the tensor.

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

---

<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model
--	--

---

<code>build</code>	
<code>data_stat</code>	
<code>get_ntypes</code>	
<code>get_out_size</code>	
<code>get_rcut</code>	
<code>get_sel_type</code>	
<code>get_type_map</code>	

`build(coord_, atype_, natoms, box, mesh, input_dict, frz_model=None, suffix='', reuse=None)`

`data_stat(data)`

`get_ntypes()`

`get_out_size()`

`get_rcut()`

`get_sel_type()`

`get_type_map()`

`init_variables(graph: tensorflow.python.framework.ops.Graph, graph_def: tensorflow.core.framework.graph_pb2.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(descrpt, fitting, type_map: Optional[List[str]] = None, data_stat_nbatch: int = 10, data_stat_protect: float = 0.01)`

Bases: `deepmd.model.tensor.TensorModel`

## Methods

---

<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model
--	--

---

build	
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.descript2r4(inputs, natoms)`

Replace  $r_{ji} \rightarrow r'_{ji}$ , where  $r_{ji} = (x_{ji}, y_{ji}, z_{ji})$  and  $r'_{ji} = (s_{ji}, \frac{s_{ji}x_{ji}}{r_{ji}}, \frac{s_{ji}y_{ji}}{r_{ji}}, \frac{s_{ji}z_{ji}}{r_{ji}})$

`deepmd.nvnmd.descriptor.se_a.filter_GR2D(xyz_scatter_1)`

Replace `se_a.py/_filter`

`deepmd.nvnmd.descriptor.se_a.filter_lower_R42GR(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 relationship of atomic distance, cutoff function, and embedding matrix.

three mapping table will be built:

$$\begin{aligned}r_{ji}^2 &\rightarrow s_{ji} \\ r_{ji}^2 &\rightarrow sr_{ji} \\ r_{ji}^2 &\rightarrow \mathcal{G}_{ji}\end{aligned}$$



where  $s_{ji}$  is cut-off function,  $sr_{ji} = \frac{s(r_{ji})}{r_{ji}}$ , and  $\mathcal{G}_{ji}$  is embedding matrix.

The mapping function can be defined 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 .npz file containing the configuration information of NVNMD model

weight\_file input file name an .npz file containing the weights of NVNMD model

map\_file output file name an .npz file containing the mapping tables of NVNMD model

## References

DOI: 10.1038/s41524-022-00773-z

## Methods

build_dG_ds	
build_ds_dr	
build_map	
build_r2s	
build_r2s_r2ds	
build_s2G	
build_s2G_s2dG	
qqq	
run_s2G	
run_u2s	

`build_dG_ds(G, s)`

`build_ds_dr(r2, s, sr)`

`build_map()`

`build_r2s(r2)`

`build_r2s_r2ds()`

`build_s2G(s)`

`build_s2G_s2dG()`

```
qqq(dat, NBIT_FEA_FL, NBIT_FEA_X, is_set_zero=False)
```

```
run_s2G(dat)
```

```
run_u2s()
```

```
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 lammmps 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

<code>wrap_dscp()</code>	Wrap the configuration of descriptor
<code>wrap_fitn()</code>	Wrap the weights of fitting net
<code>wrap_map()</code>	Wrap the mapping table of embedding network

<code>wrap</code>	
<code>wrap_bias</code>	
<code>wrap_head</code>	
<code>wrap_weight</code>	

```
wrap()
```

```
wrap_bias(bias, NBIT_SUM, NBIT_DATA_FL)
```

```
wrap_dscp()
```

Wrap the configuration of descriptor

```
wrap_fitn()
```

Wrap the weights of fitting net

```
wrap_head(NCFG, NNET, NFEA)
```

```
wrap_map()
```

Wrap the mapping table of embedding network

```
wrap_weight(weight, NBIT_WEIGHT, NBIT_WEIGHT_FL)
```

```
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

```
descript_attr/t_avg:0
descript_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 relationship of atomic distance, cutoff function, and embedding matrix.

three mapping table will be built:

$$\begin{aligned} r_{ji}^2 &\rightarrow s_{ji} \\ r_{ji}^2 &\rightarrow sr_{ji} \\ r_{ji}^2 &\rightarrow \mathcal{G}_{ji} \end{aligned}$$

where  $s_{ji}$  is cut-off function,  $sr_{ji} = \frac{s(r_{ji})}{r_{ji}}$ , and  $\mathcal{G}_{ji}$  is embedding matrix.

The mapping function 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 .npz file containing the configuration information of NVNMD model

`weight_file` input file name an .npz file containing the weights of NVNMD model

`map_file` output file name an .npz file containing the mapping tables of NVNMD model

#### References

DOI: 10.1038/s41524-022-00773-z

#### Methods

<code>build_dG_ds</code>	
<code>build_ds_dr</code>	
<code>build_map</code>	
<code>build_r2s</code>	
<code>build_r2s_r2ds</code>	
<code>build_s2G</code>	
<code>build_s2G_s2dG</code>	
<code>qqq</code>	
<code>run_s2G</code>	
<code>run_u2s</code>	

`build_dG_ds(G, s)`

`build_ds_dr(r2, s, sr)`

`build_map()`

`build_r2s(r2)`

`build_r2s_r2ds()`

`build_s2G(s)`

`build_s2G_s2dG()`

`qqq(dat, NBIT_FEA_FL, NBIT_FEA_X, is_set_zero=False)`

`run_s2G(dat)`

`run_u2s()`

```
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.add_path(p, p2)

deepmd.nvnmd.entrypoints.train.normalized_input(fn, PATH_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.replace_path(p, p2)

deepmd.nvnmd.entrypoints.train.train_nvnmd(*, INPUT: 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 lammmps 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

<i>wrap_dscp()</i>	Wrap the configuration of descriptor
<i>wrap_fitn()</i>	Wrap the weights of fitting net
<i>wrap_map()</i>	Wrap the mapping table of embedding network

wrap	
wrap_bias	
wrap_head	
wrap_weight	

**wrap()**

**wrap\_bias**(bias, NBIT\_SUM, NBIT\_DATA\_FL)

**wrap\_dscp()**

Wrap the configuration of descriptor

**wrap\_fitn()**

Wrap the weights of fitting net

**wrap\_head**(NCFG, NNET, NFEA)

**wrap\_map()**

Wrap the mapping table of embedding network

**wrap\_weight**(weight, NBIT\_WEIGHT, NBIT\_WEIGHT\_FL)

```
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.utils package

```
class deepmd.nvnmd.utils.Encode
```

Bases: `object`

Encoding value as hex, bin, and dec format

## Methods

<code>bin2hex(data)</code>	Convert binary string list to hex string list
<code>bin2hex_str(sbin)</code>	Convert binary string to hex string
<code>check_dec(idec, nbit[, signed, name])</code>	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
<code>dec2bin(idec[, nbit, signed, name])</code>	Convert dec array to binary string list
<code>extend_bin(slbin, nfull)</code>	Extend the element of list (slbin) to the length (nfull)
<code>extend_hex(slhex, nfull)</code>	Extend the element of list (slhex) to the length (nfull)
<code>extend_list(slbin, nfull)</code>	Extend the list (slbin) to the length (nfull) the attached element of list is 0
<code>hex2bin(data)</code>	Convert hex string list to binary string list
<code>hex2bin_str(shex)</code>	Convert hex string to binary string
<code>merge_bin(slbin, nmerge)</code>	Merge binary string list per nmerge value
<code>qc(v[, nbit])</code>	Quantize value using ceil
<code>qf(v[, nbit])</code>	Quantize value using floor
<code>qr(v[, nbit])</code>	Quantize value using round
<code>reverse_bin(slbin, nreverse)</code>	Reverse binary string list per nreverse value
<code>split_bin(sbin, nbit)</code>	Split sbins into many segment with the length nbit

**bin2hex**(data)

Convert binary string list to hex string list

**bin2hex\_str**(sbin)

Convert binary string to hex string

**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 attached element of list is 0
    such as, when

    slbin = ['10010','10100'],
    nfull = 4

    extent it to
    ['10010','10100','00000','00000']
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: 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
class deepmd.nvnmd.utils.FioBin
    Bases: object
    Input and output for binary file
```

### Methods

---

<code>load</code> ([file_name, default_value])	Load binary file into bytes value
<code>save</code> ([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: 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

<code>update(jdata, jdata_o)</code>	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()

```
class deepmd.nvnmd.utils.FioTxt
```

Bases: `object`

Input and output for .txt file with string

### Methods

<code>load([file_name, default_value])</code>	Load .txt file into string list
<code>save([file_name, data])</code>	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

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

`spe_i(int)` special order of central atom  $i$   $0 \sim n_{\text{type}} - 1$

`layer_l(int)` layer order in embedding network  $0 \sim n_{\text{layer}} - 1$

`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

<i>disp_message()</i>	Display the log of NVNMD
<i>get_deepmd_jdata()</i>	Generate input script with member element one by one
<i>get_dscp_jdata()</i>	Generate model/descriptor in input script
<i>get_fitn_jdata()</i>	Generate model/fitting_net in input script
<i>get_learning_rate_jdata()</i>	Generate learning_rate in input script
<i>get_loss_jdata()</i>	Generate loss in input script
<i>get_model_jdata()</i>	Generate model in input script
<i>get_nvnmmd_jdata()</i>	Generate nvnmmd in input script
<i>get_training_jdata()</i>	Generate training in input script
<i>init_ctrl(jdata[, jdata_parent])</i>	Initial members about control signal
<i>init_dscp(jdata[, jdata_parent])</i>	Initial members about descriptor
<i>init_fitn(jdata[, jdata_parent])</i>	Initial members about fitting network
<i>init_from_config(jdata)</i>	Initial member element one by one
<i>init_from_deepmd_input(jdata)</i>	Initial members with input script of deepmd
<i>init_from_jdata([jdata])</i>	Initial this class with jdata loaded from input script
<i>init_nbit(jdata[, jdata_parent])</i>	Initial members about quantification precision
<i>init_net_size()</i>	Initial net_size
<i>init_size(jdata[, jdata_parent])</i>	Initial members about ram capacity
<i>init_train_mode([mod])</i>	Configure for taining cnn or qnn
<i>init_value()</i>	Initial member with dict
<i>save([file_name])</i>	Save all configuration to file

**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\_nvnmmd\_jdata()**

Generate nvnmmd in input script

**get\_training\_jdata()**

Generate training in input script

`init_ctrl(jdata: dict, jdata_parent: dict = {}) → dict`  
Initial members about control signal

`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 training cnn or qnn

`init_value()`  
Initial member with dict

`save(file_name=None)`  
Save all configuration to file

#### **deepmd.nvnmd.utils.encode module**

`class deepmd.nvnmd.utils.encode.Encode`  
Bases: `object`  
Encoding value as hex, bin, and dec format

## Methods

<code>bin2hex(data)</code>	Convert binary string list to hex string list
<code>bin2hex_str(sbin)</code>	Convert binary string to hex string
<code>check_dec(idec, nbit[, signed, name])</code>	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
<code>dec2bin(idec[, nbit, signed, name])</code>	Convert dec array to binary string list
<code>extend_bin(slbin, nfull)</code>	Extend the element of list (slbin) to the length (nfull)
<code>extend_hex(slhex, nfull)</code>	Extend the element of list (slhex) to the length (nfull)
<code>extend_list(slbin, nfull)</code>	Extend the list (slbin) to the length (nfull) the attached element of list is 0
<code>hex2bin(data)</code>	Convert hex string list to binary string list
<code>hex2bin_str(shex)</code>	Convert hex string to binary string
<code>merge_bin(slbin, nmerge)</code>	Merge binary string list per nmerge value
<code>qc(v[, nbit])</code>	Quantize value using ceil
<code>qf(v[, nbit])</code>	Quantize value using floor
<code>qr(v[, nbit])</code>	Quantize value using round
<code>reverse_bin(slbin, nreverse)</code>	Reverse binary string list per nreverse value
<code>split_bin(sbin, nbit)</code>	Split sbins into many segments with the length nbit

**bin2hex**(data)

Convert binary string list to hex string list

**bin2hex\_str**(sbin)

Convert binary string to hex string

**check\_dec**(idec, nbit, signed=False, name='')

Check whether the data (idec) is in the range range is  $[0, 2^{nbit} - 1]$  for unsigned range is  $[-2^{nbit-1}, 2^{nbit-1} - 1]$  for signed

**dec2bin**(idec, nbit=10, signed=False, name='')

Convert dec array to binary string list

**extend\_bin**(slbin, nfull)

Extend the element of list (slbin) to the length (nfull)

such as, when

```
slbin = ['10010', '10100'],
nfull = 6
```

extend to

```
['010010', '010100']
```

**extend\_hex**(slhex, nfull)

Extend the element of list (slhex) to the length (nfull)

**extend\_list**(slbin, nfull)

Extend the list (slbin) to the length (nfull) the attached element of list is 0  
such as, when

```
slbin = ['10010', '10100'],  
nfull = 4
```

extent it to

```
['10010', '10100', '00000', '00000']
```

**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: `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

## 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

<code>load([file_name, default_value])</code>	Load binary file into bytes value
<code>save([file_name, data])</code>	Save hex string into binary file

```
load(file_name='', default_value='')

```

Load binary file into bytes value

```
save(file_name: str = '', data: 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 .npz file containing a dictionary

### Methods

<code>update(jdata, jdata_o)</code>	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()

```
class deepmd.nvnmd.utils.fio.FioJsonDic
```

Bases: `object`

Input and output for .json file containing dictionary

## Methods

<code>load([file_name, default_value])</code>	Load .json file into dict
<code>save([file_name, dic])</code>	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 .npz 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

<code>load([file_name, default_value])</code>	Load .txt file into string list
<code>save([file_name, data])</code>	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.tanh2(x, nbit=-1, nbit2=-1)`

User-defined activation function tanh2

`deepmd.nvnmd.utils.network.tanh4(x, nbit=-1, nbit2=-1)`

User-defined activation function tanh4

### **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.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

`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

`spe_i(int)` special order of central atom i 0~ntype-1

`layer_l(int)` layer order in embedding network 0~nlayer-1

`deepmd.nvnmd.utils.weight.get_normalize(weights: dict)`

Get normalize parameter (avg and std) of  $s_{ji}$

`deepmd.nvnmd.utils.weight.get_rng_s(weights: dict)`

Guess the range 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

Initially this subdir is unpopulated. CMake will install all the op module python files and shared libs.

## deepmd.train package

### Submodules

#### deepmd.train.run\_options module

Module taking care of important package constants.

```
class deepmd.train.run_options.RunOptions(init_model: Optional[str] = None, init_frz_model:
Optional[str] = None, restart: Optional[str] = None,
log_path: Optional[str] = None, log_level: int = 0,
mpi_log: str = 'master')
```

Bases: `object`

Class with info on how to run training (cluster, MPI and GPU config).

Attributes

`gpus`: `Optional[List[int]]` list of GPUs if any are present else None

`is_chief`: bool in distributed training it is true for the main MPI process in serial it is always true

`world_size`: int total worker count

`my_rank`: int index of the MPI task

`nodename`: str name of the node

`node_list`: `[List[str]]` the list of nodes of the current mpirun

`my_device`: str device type - gpu or cpu

## Methods

---

<code>print_resource_summary()</code>	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`  
`odelist: 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

---

<code>save_compressed()</code>	Save the compressed graph
--------------------------------	---------------------------

---

build	
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)`  
`get_evaluation_results(batch_list)`  
`get_feed_dict(batch, is_training)`  
`get_global_step()`

```
static print_header(fp, train_results, valid_results)

static print_on_training(fp, train_results, valid_results, cur_batch, cur_lr)

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)
```

## deepmd.utils package

### Submodules

#### deepmd.utils.argcheck module

```
class deepmd.utils.argcheck.ArgsPlugin
    Bases: object
```

#### Methods

---

<code>get_all_argument([exclude_hybrid])</code>	Get all arguments.
<code>register(name[, alias])</code>	Register a descriptor argument plugin.

---

**get\_all\_argument** (exclude\_hybrid: `bool` = False) → `List[dargs.dargs.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[dargs.dargs.Argument]]`

Register a descriptor argument plugin.

Parameters

name [`str`] the name of a descriptor

alias [`List[str]`, `optional`] the list of aliases of this descriptor

Returns

`Callable[[], List[Argument]]` the registered descriptor argument method

## Examples

```
>>> some_plugin = ArgsPlugin()
>>> @some_plugin.register("some_descrpt")
    def descrpt_some_descrpt_args():
        return []
```

```
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_tpe_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) →
    dargs.dargs.Variant
deepmd.utils.argcheck.fitting_dipole()
deepmd.utils.argcheck.fitting_ener()
deepmd.utils.argcheck.fitting_polar()
deepmd.utils.argcheck.fitting_variant_type_args()
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_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_ener()
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_hybrid_list(hy_list)
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)  
`factor` [`float`, default: 2.] increased factor

#### Notes

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

<code>execute(callable, start_index, natoms)</code>	Excuate a method with given batch size.
<code>execute_all(callable, total_size, natoms, ...)</code>	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[numpy.ndarray]`

Excuate a method with all given data.

Parameters

callable [`Callable`] The method should accept \*args and \*\*kwargs as input and return the similiar array.

total\_size [`int`] Total size

natoms [`int`] The number of atoms

\*\*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, pathlib.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, pathlib.Path]]` = None) → `Dict[str, Any]`

```
deepmd.utils.compat.deprecate_numb_test(jdata: Dict[str, Any], warning: bool = True, dump:
                                         Optional[Union[str, pathlib.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, pathlib.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 1.0 graph text to 1.1 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

`file` [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.data module**

```
class deepmd.utils.data.DataSets(sys_path, set_prefix, seed=None, shuffle_test=True)
```

Bases: `object`

Outdated class for one data system.

Deprecated since version 2.0.0: This class is not maintained any more.

**Methods**

<code>get_batch(batch_size)</code>	returned property prefector [4] in order: energy, force, virial, atom_ener
<code>get_test()</code>	returned property prefector [4] in order: energy, force, virial, atom_ener
<code>load_energy(set_name, nframes, nvalues, ...)</code>	return : coeff_ener, ener, coeff_atom_ener, atom_ener

check_batch_size	
check_test_size	
get_ener	
get_natoms	
get_natoms_2	
get_natoms_vec	
get_numb_set	
get_set	
get_sys_numb_batch	
get_type_map	
load_batch_set	
load_data	
load_set	
load_test_set	
load_type	
load_type_map	
numb_aparam	
numb_fparam	
reset_iter	
set_numb_batch	
stats_energy	

```
check_batch_size(batch_size)
```

```
check_test_size(test_size)
```

```
get_batch(batch_size)
```

returned property prefector [4] in order: energy, force, virial, atom\_ener

```
get_ener()
```

```
get_natoms()
```

```

get_natoms_2(ntypes)
get_natoms_vec(ntypes)
get_numb_set()
get_set(data, idx=None)
get_sys_numb_batch(batch_size)
get_test()
    returned property prefector [4] in order: energy, force, virial, atom_ener
get_type_map()
load_batch_set(set_name)
load_data(set_name, data_name, shape, is_necessary=True)
load_energy(set_name, nframes, nvalues, energy_file, atom_energy_file)
    return : coeff_ener, ener, coeff_atom_ener, atom_ener
load_set(set_name, shuffle=True)
load_test_set(set_name, shuffle_test)
load_type(sys_path)
load_type_map(sys_path)
numb_aparam()
numb_fparam()
reset_iter()
set_numb_batch(batch_size)
stats_energy()

```

```

class deepmd.utils.data.DeepmdData(sys_path: str, set_prefix: str = 'set', shuffle_test: bool = True,
                                   type_map: Optional[List[str]] = None, 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
- `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

<code>add(key, ndof[, atomic, must, high_prec, ...])</code>	Add a data item that to be loaded
<code>avg(key)</code>	Return the average value of an item.
<code>check_batch_size(batch_size)</code>	Check if the system can get a batch of data with <code>batch_size</code> frames.
<code>check_test_size(test_size)</code>	Check if the system can get a test dataset with <code>test_size</code> frames.
<code>get_atom_type()</code>	Get atom types
<code>get_batch(batch_size)</code>	Get a batch of data with <code>batch_size</code> frames.
<code>get_data_dict()</code>	Get the <code>data_dict</code>
<code>get_natoms()</code>	Get number of atoms
<code>get_natoms_vec(ntypes)</code>	Get number of atoms and number of atoms in different types
<code>get_ntypes()</code>	Number of atom types in the system
<code>get_num_batch(batch_size, set_idx)</code>	Get the number of batches in a set.
<code>get_num_set()</code>	Get number of training sets
<code>get_sys_num_batch(batch_size)</code>	Get the number of batches in the data system.
<code>get_test([ntests])</code>	Get the test data with <code>ntests</code> frames.
<code>get_type_map()</code>	Get the type map
<code>reduce(key_out, key_in)</code>	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)

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

**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.DataSystem(systems, set_prefix, batch_size, test_size, rcut,
                                          run_opt=None)
```

Bases: `object`

Outdated class for the data systems.

Deprecated since version 2.0.0: This class is not maintained any more.

### Methods

check_type_map_consistency	
compute_energy_shift	
format_name_length	
get_batch	
get_batch_size	
get_nbatches	
get_nsystems	
get_ntypes	
get_sys	
get_test	
get_type_map	
numb_fparam	
print_summary	
process_sys_weights	

```
check_type_map_consistency(type_map_list)
```

```
compute_energy_shift()
```

```
format_name_length(name, width)
```

```
get_batch(sys_idx=None, sys_weights=None, style='prob_sys_size')
```

```
get_batch_size()
```

```
get_nbatches()
```

```
get_nsystems()
```

```
get_ntypes()
```

```
get_sys(sys_idx)
```

```
get_test(sys_idx=None)
```

```
get_type_map()
```

```
numb_fparam()
```

```

print_summary()

process_sys_weights(sys_weights)

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, 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

<code>add(key, ndof[, atomic, must, high_prec, ...])</code>	Add a data item that to be loaded
<code>add_dict(adict)</code>	Add items to the data system by a dict.
<code>get_batch([sys_idx])</code>	Get a batch of data from the data systems
<code>get_batch_size()</code>	Get the batch size
<code>get_nbatchses()</code>	Get the total number of batches
<code>get_nsystems()</code>	Get the number of data systems
<code>get_ntypes()</code>	Get the number of types
<code>get_sys(idx)</code>	Get a certain data system
<code>get_sys_ntest([sys_idx])</code>	Get number of tests for the currently selected system,
<code>get_test([sys_idx, n_test])</code>	Get test data from the the data systems.
<code>get_type_map()</code>	Get the type map
<code>reduce(key_out, key_in)</code>	Generate a new item from the reduction of another atom

<code>compute_energy_shift</code>	
<code>get_data_dict</code>	
<code>print_summary</code>	
<code>set_sys_probs</code>	

`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 explanation of the keys see add

**compute\_energy\_shift**(rcond=0.001, key='energy')

**get\_batch**(sys\_idx: Optional[int] = None)

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.

**get\_batch\_size**() → int

Get the batch size

**get\_data\_dict**(ii: int = 0) → dict

**get\_nbatch**() → 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) → deepmd.utils.data.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.graph module

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: tensorflow.core.framework.graph\_pb2.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: `tensorflow.core.framework.graph_pb2.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: `tensorflow.core.framework.graph_pb2.GraphDef`)  
→ `Dict`

Get the fitting net nodes with the given `tf.GraphDef` object

Parameters

graph\_def The input `tf.GraphDef` object

Returns

`Dict` The fitting net nodes within the given `tf.GraphDef` object

`deepmd.utils.graph.get_fitting_net_variables`(model\_file: `str`) → `Dict`

Get the fitting net variables with the given frozen model(model\_file)

Parameters

model\_file The input frozen model path

Returns

`Dict` The fitting net variables within the given frozen model

`deepmd.utils.graph.get_fitting_net_variables_from_graph_def`(graph\_def: `tensorflow.core.framework.graph_pb2.GraphDef`)  
→ `Dict`

Get the fitting net variables with the given `tf.GraphDef` object

Parameters

graph\_def The input `tf.GraphDef` object

Returns

**Dict** The fitting net variables within the given `tf.GraphDef` object

`deepmd.utils.graph.get_pattern_nodes_from_graph_def`(`graph_def`: `tensorflow.core.framework.graph_pb2.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`) →  
`tensorflow.python.framework.ops.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`:  
`tensorflow.python.framework.ops.Graph`,  
`tensor_name`: `str`) →  
`tensorflow.python.framework.ops.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`: `numpy.dtype`) →  
`tensorflow.python.framework.ops.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: `tensorflow.core.framework.graph_pb2.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: `tensorflow.core.framework.graph_pb2.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[tensorflow.python.framework.ops.Graph, tensorflow.core.framework.graph_pb2.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  $\alpha$  is the learning rate,  $\alpha_0$  is the starting learning rate,  $\lambda$  is the decay rate, and  $\tau$  is the decay steps.

## Parameters

`start_lr` Starting learning rate  $\alpha_0$

`stop_lr` Stop learning rate  $\alpha_1$

`decay_steps` Learning rate decay every this number of steps  $\tau$

`decay_rate` The decay rate  $\lambda$ . If `stop_step` is provided in `build`, then it will be determined automatically and overwritten.

## Methods

<code>build(global_step[, stop_step])</code>	Build the learning rate
<code>start_lr()</code>	Get the start lr
<code>value(step)</code>	Get the lr at a certain step

**build**(global\_step: tensorflow.python.framework.ops.Tensor, stop\_step: Optional[int] = None) → tensorflow.python.framework.ops.Tensor

Build the learning rate

## Parameters

`global_step` The tf Tensor providing the global training step

`stop_step` The stop step. If provided, the `decay_rate` will be determined automatically and overwritten.

## Returns

**learning\_rate** The learning rate

**start\_lr()** → float

Get the start lr

**value**(step: int) → float

Get the lr at a certain step

**deepmd.utils.neighbor\_stat module**

**class** deepmd.utils.neighbor\_stat.NeighborStat(ntypes: int, rcut: float)

Bases: `object`

Class for getting training data information.

It loads data from DeepmdData object, and measures the data info, including nearest nbor distance between atoms, max nbor size of atoms and the output data range of the environment matrix.

## Parameters

`ntypes` The num of atom types

`rcut` The cut-off radius

## Methods

---

<code>get_stat(data)</code>	get the data statistics of the training data, including nearest nbor distance between atoms, max nbor size of atoms
-----------------------------	---

---

`get_stat(data: deepmd.utils.data_system.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  $\mathcal{L}^{(i)}$ :

$$\mathcal{N} = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \dots \circ \mathcal{L}^{(1)}$$

A layer  $\mathcal{L}$  is given by one of the following forms, depending on the number of nodes: [1]

$$y = \mathcal{L}(x; w, b) = \begin{cases} \phi(x^T w + b) + x, & N_2 = N_1 \\ \phi(x^T w + b) + (x, x), & N_2 = 2N_1 \\ \phi(x^T w + b), & \text{otherwise} \end{cases}$$

where  $x \in \mathbb{R}^{N_1}$  is the input vector and  $y \in \mathbb{R}^{N_2}$  is the output vector.  $w \in \mathbb{R}^{N_1 \times N_2}$  and  $b \in \mathbb{R}^{N_2}$  are weights and biases, respectively, both of which are trainable if trainable is True.  $\phi$  is the activation function.

Parameters

`xx` [Tensor] Input tensor x of shape [-1,1]

`network_size`: list of int Size of the embedding network. For example [16,32,64]

`precision`: Precision of network weights. For example, tf.float64

`activation_fn`: Activation function  $\phi$

`resnet_dt`: boolean 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 initial bias  
 seed: int Random seed for initializing network parameters  
 trainable: boolean 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',
                                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: tensorflow.python.ops.variables.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`

Parameters

filename File name for the short-range tabulated potential. The table is a text data file with  $(N_t + 1) * N_t / 2 + 1$  columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

## Methods

<code>get()</code>	Get the serialized table.
<code>reinit(filename)</code>	Initialize the tabulated interaction

`get()` → `Tuple[numpy.array, numpy.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  $(N_t + 1) * N_t / 2 + 1$  columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

## deepmd.utils.parallel\_op module

```
class deepmd.utils.parallel_op.ParallelOp(builder: Callable[[...], Tuple[Dict[str,
    tensorflow.python.framework.ops.Tensor],
    Tuple[tensorflow.python.framework.ops.Tensor]]],
    nthreads: Optional[int] = None, config: Optional[tensorflow.core.protobuf.config_pb2.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

<code>generate(sess, feed)</code>	Returns a generator.
-----------------------------------	----------------------

---

**generate**(sess: tensorflow.python.client.session.Session, feed: Generator[Dict[str, Any], None, None])  
 → Generator[Tuple, None, None]

Returns a generator.

Parameters

    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: *deepmd.utils.path.DPPath*

The path class to data system (DeepmdData) for HDF5 files.

Parameters

    path [str] path

## Notes

OS - HDF5 relationship: directory - Group file - Dataset

## Methods

<code>glob(pattern)</code>	Search path using the glob pattern.
<code>is_dir()</code>	Check if self is directory.
<code>is_file()</code>	Check if self is file.
<code>load_numpy()</code>	Load NumPy array.
<code>load_txt([dtype])</code>	Load NumPy array from text.
<code>rglob(pattern)</code>	This is like calling <i>DPPath.glob()</i> with <code>**/</code> added in front of the given relative pattern.

**glob**(pattern: str) → List[deepmd.utils.path.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()` → `numpy.ndarray`

Load NumPy array.

Returns

`np.ndarray` loaded NumPy array

`load_txt(dtype: Optional[numpy.dtype] = None, **kwargs)` → `numpy.ndarray`

Load NumPy array from text.

Returns

`np.ndarray` loaded NumPy array

`rglob(pattern: str)` → `List[deepmd.utils.path.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: `deepmd.utils.path.DPPath`

The OS path class to data system (DeepmdData) for real directories.

Parameters

path [`str`] path

## Methods

<code>glob(pattern)</code>	Search path using the glob pattern.
<code>is_dir()</code>	Check if self is directory.
<code>is_file()</code>	Check if self is file.
<code>load_numpy()</code>	Load NumPy array.
<code>load_txt(**kwargs)</code>	Load NumPy array from text.
<code>rglob(pattern)</code>	This is like calling <code>DPPath.glob()</code> with <code>**/</code> added in front of the given relative pattern.

`glob(pattern: str)` → `List[deepmd.utils.path.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()` → `numpy.ndarray`

Load NumPy array.

Returns

`np.ndarray` loaded NumPy array

`load_txt(**kwargs)` → `numpy.ndarray`

Load NumPy array from text.

Returns

`np.ndarray` loaded NumPy array

`rglob(pattern: str)` → `List[deepmd.utils.path.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.ABC`

The path class to data system (DeepmdData).

Parameters

path [`str`] path

## Methods

<code>glob(pattern)</code>	Search path using the glob pattern.
<code>is_dir()</code>	Check if self is directory.
<code>is_file()</code>	Check if self is file.
<code>load_numpy()</code>	Load NumPy array.
<code>load_txt(**kwargs)</code>	Load NumPy array from text.
<code>rglob(pattern)</code>	This is like calling <code>DPPath.glob()</code> with <code>**/</code> added in front of the given relative pattern.

`abstract glob(pattern: str)` → `List[deepmd.utils.path.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()`  $\rightarrow$  `bool`

Check if self is file.

**abstract** `load_numpy()`  $\rightarrow$  `numpy.ndarray`

Load NumPy array.

Returns

`np.ndarray` loaded NumPy array

**abstract** `load_txt(**kwargs)`  $\rightarrow$  `numpy.ndarray`

Load NumPy array from text.

Returns

`np.ndarray` loaded NumPy array

**abstract** `rglob(pattern: str)`  $\rightarrow$  `List[deepmd.utils.path.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

<code>get_plugin(key)</code>	Visit a plugin by key.
<code>register(key)</code>	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: `deepmd.utils.plugin.VariantMeta`, `abc.ABCMeta`

## Methods

<code>__call__(*args, **kwargs)</code>	Remove type and keys that starts with underline.
<code>mro()</code>	Return a type's method resolution order.
<code>register(subclass)</code>	Register a virtual subclass of an ABC.

`class deepmd.utils.plugin.VariantMeta`

Bases: `object`

## Methods

<code>__call__(*args, **kwargs)</code>	Remove type and keys that starts with underline.
--	--

**deepmd.utils.random module**

`deepmd.utils.random.choice(a: numpy.ndarray, p: Optional\[numpy.ndarray\] = 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: numpy.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: tensorflow.python.client.session.Session, *args, **kwargs)`

Run session with errors caught.

Parameters

sess: `tf.Session` TensorFlow Session

Returns

the result of `sess.run()`

**deepmd.utils.tabulate module**

```
class deepmd.utils.tabulate.DPTabulate(descrpt: deepmd.descriptor.descriptor.Descriptor, neuron:
    typing.List[int], model_file: str, 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 its uniform stride. The range of the first table is automatically detected by deepmd-kit, while the second table ranges from the first table's upper boundary(`upper`) to the `extrapolate(parameter) * upper`.

Parameters

`descrpt` Descriptor of the original model

`neuron` Number of neurons in each hidden layers of the embedding net  $\mathcal{N}$

`model_file` The frozen 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**


---

<i>build</i> ( <code>min_nbor_dist</code> , <code>extrapolate</code> , <code>stride0</code> , ...)	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

`neuron` Number of neurons in each hidden layers of the embedding net  $\mathcal{N}$

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: str = 'tanh', precision: str =
                                           'default', trainable: bool = True, seed: Optional[int] =
                                           None, uniform_seed: bool = False)
```

Bases: `object`

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”.

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

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

## Methods

<code>build(ntypes[, reuse, suffix])</code>	Build the computational graph for the descriptor
<code>init_variables(graph, graph_def[, suffix])</code>	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: tensorflow.python.framework.ops.Graph`, `graph_def: tensorflow.core.framework.graph_pb2.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:
                                         tensorflow.python.framework.ops.Tensor,
                                         type_embedding:
                                         tensorflow.python.framework.ops.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]`:  $2 \leq i < Ntypes+2$ , 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 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 Tuple: (error, weight)

Returns

`Dict` weighted averages

## 16.1.2 Submodules

### 16.1.3 deepmd.calculator module

ASE calculator interface module.

```
class deepmd.calculator.DP(model: Union[str, pathlib.Path], label: str = 'DP', type_dict:
                             Optional[Dict[str, int]] = None, **kwargs)
```

Bases: `ase.calculators.calculator.Calculator`

Implementation of ASE deepmd calculator.

Implemented propertie are energy, forces and stress

Parameters

`model` [`Union[str, Path]`] path to the model

`label` [`str`, optional] calculator label, by default “DP”

`type_dict [Dict[str, int], optional]` mapping of element types and their numbers, best left None and the calculator will infer this information from model, by default None

## 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

<code>band_structure()</code>	Create band-structure object for plotting.
<code>calculate([atoms, properties, system_changes])</code>	Run calculation with deepmd model.
<code>calculate_numerical_forces(atoms[, d])</code>	Calculate numerical forces using finite difference.
<code>calculate_numerical_stress(atoms[, d, voigt])</code>	Calculate numerical stress using finite difference.
<code>calculate_properties(atoms, properties)</code>	This method is experimental; currently for internal use.
<code>check_state(atoms[, tol])</code>	Check for any system changes since last calculation.
<code>get_magnetic_moments([atoms])</code>	Calculate magnetic moments projected onto atoms.
<code>get_property(name[, atoms, allow_calculation])</code>	Get the named property.
<code>get_stresses([atoms])</code>	the calculator should return intensive stresses, i.e., such that <code>stresses.sum(axis=0) == stress</code>
<code>read(label)</code>	Read atoms, parameters and calculated properties from output file.
<code>reset()</code>	Clear all information from old calculation.
<code>set(**kwargs)</code>	Set parameters like <code>set(key1=value1, key2=value2, ...)</code> .
<code>set_label(label)</code>	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 compat-
            ibility, by default all_changes

    implemented_properties: List[str] = ['energy', 'free_energy', 'forces', 'virial',
    'stress']

    Properties calculator can handle (energy, forces, ...)

    name = 'DP'

```

### 16.1.4 deepmd.common module

Collection of functions and classes used throughout the whole package.

**class** `deepmd.common.ClassArg`

Bases: `object`

Class that take care of input json/yaml parsing.

The rules for parsing are defined by the add method, than parse is called to process the supplied dict

Attributes

`arg_dict`: Dict[str, Any] dictionary containing parsing rules

`alias_map`: Dict[str, Any] dictionary with keyword aliases

#### Methods

<code>add(key, types_[, alias, default, must])</code>	Add key to be parsed.
<code>get_dict()</code>	Get dictionary built from rules defined by add method.
<code>parse(jdata)</code>	Parse input dictionary, use the rules defined by add method.

**add**(key: str, types\_: Union[type, List[type]], alias: Optional[Union[str, List[str]]] = None, default: Optional[Any] = None, must: bool = False) → `deepmd.common.ClassArg`

Add key to be parsed.

Parameters

`key` [str] key name

`types_` [Union[type, List[type]]] list of allowed key types

`alias` [Optional[Union[str, List[str]]], optional] alias for the key, by default None

`default` [Any, optional] default value for the key, by default None

`must` [bool, optional] if the key is mandatory, by default False

Returns

`ClassArg` instance with added key

**get\_dict**() → Dict[str, Any]

Get dictionary built from rules defined by add method.

Returns

`Dict[str, Any]` settings dictionary with default values

`parse(jdata: Dict[str, Any]) → Dict[str, Any]`

Parse input dictionary, use the rules defined by add method.

Parameters

`jdata` [`Dict[str, Any]`] loaded json/yaml data

Returns

`Dict[str, Any]` parsed dictionary

`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)`

Specify data requirements for training.

Parameters

`key` [`str`] type of data stored in corresponding \*.npz file e.g. forces or energy

`ndof` [`int`] number of the degrees of freedom, this is tied to atomic parameter e.g. forces have `atomic=True` and `ndof=3`

`atomic` [`bool`, `optional`] specifies whether the `ndof` keyword applies to per atom quantity or not, by default `False`

`must` [`bool`, `optional`] specify if the \*.npz 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 repeat data repeat times, by default 1

`default` [`float`, `optional`, `default=0.`] default value of data

`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.expand_sys_str`(root\_dir: `Union[str, pathlib.Path]`)  $\rightarrow$  `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: `tensorflow.python.framework.ops.Tensor`)  $\rightarrow$   
`tensorflow.python.framework.ops.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: `tensorflow.python.framework.ops.Tensor`)  $\rightarrow$   
`tensorflow.python.framework.ops.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: _ACTIVATION) → Callable[[tensorflow.python.framework.ops.Tensor], tensorflow.python.framework.ops.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) → numpy.dtype`

Get numpy precision constant from string.

Parameters

`precision` [\_PRECISION] string name of numpy constant or default

Returns

`np.dtype` numpy presicion constant

Raises

`RuntimeError` if string is invalid

`deepmd.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 acoresponding TF constant

`deepmd.common.j_loader(filename: Union[str, pathlib.Path]) → Dict[str, Any]`

Load yaml or json settings file.

Parameters

`filename` [Union[str, Path]] path to file

Returns

`Dict[str, Any]` loaded dictionary

Raises

`TypeError` if the supplied file is of unsupported type

`deepmd.common.j_must_have(jdata: Dict[str, _DICT_VAL], key: str, deprecated_key: List[str] = []) → _DICT_VAL`

Assert that supplied dictionary contains specified key.

Returns

`_DICT_VAL` value that was stored under supplied key

Raises

`RuntimeError` if the key is not present

`deepmd.common.make_default_mesh(test_box: numpy.ndarray, cell_size: float = 3.0) → numpy.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: tensorflow.python.framework.ops.Tensor, from_precision: tensorflow.python.framework.dtypes.DType, to_precision: tensorflow.python.framework.dtypes.DType) → tensorflow.python.framework.ops.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

`precision` [`tf.DType`] Tensor data type that casts to

Returns

`tf.Tensor` casted Tensor

`deepmd.common.select_idx_map(atom_types: numpy.ndarray, select_types: numpy.ndarray) → numpy.ndarray`

Build map of indices for element supplied element types from all atoms list.

Parameters

`atom_types` [`np.ndarray`] array specifying type for each atom 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`



### 16.1.5 deepmd.env module

Module that sets tensorflow working environment and exports important constants.

`deepmd.env.GLOBAL_ENER_FLOAT_PRECISION`

alias of `numpy.float64`

`deepmd.env.GLOBAL_NP_FLOAT_PRECISION`

alias of `numpy.float64`

`deepmd.env.global_cvt_2_ener_float(xx: tensorflow.python.framework.ops.Tensor) → tensorflow.python.framework.ops.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: tensorflow.python.framework.ops.Tensor) → tensorflow.python.framework.ops.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 Class Hierarchy

- Namespace deepmd
  - Struct deepmd\_exception
  - Struct InputNlist
  - Struct NeighborListData
  - Struct tf\_exception
  - Template Class AtomMap
  - Class DeepPot
  - Class DeepPotModelDevi
  - Class DeepTensor
  - Class DipoleChargeModifier

## 17.2 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

## 17.3 Full API

### 17.3.1 Namespaces

#### Namespace `deepmd`

##### Contents

- [Classes](#)
- [Functions](#)
- [Typedefs](#)

#### Classes

- [Struct `deepmd\_exception`](#)
- [Struct `InputNlist`](#)
- [Struct `NeighborListData`](#)
- [Struct `tf\_exception`](#)
- [Template 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\_compatible`](#)
- [Function `deepmd::name\_prefix`](#)
- [Function `deepmd::read\_file\_to\_string`](#)
- [Function `deepmd::select\_by\_type`](#)
- [Template Function `deepmd::select\_map\(std::vector<VT>&, const std::vector<VT>&, const std::vector<int>&, const int&\)`](#)
- [Template Function `deepmd::select\_map\(typename std::vector<VT>::iterator, const typename std::vector<VT>::const\_iterator, const std::vector<int>&, const int&\)`](#)

- 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&)`
- Function `deepmd::select_real_atoms`
- Template Function `deepmd::session_get_scalar`
- Template Function `deepmd::session_get_vector`
- 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 VALUETYPE&, const std::vector<VALUETYPE>&, const std::vector<VALUETYPE>&, const deepmd::AtomMap<VALUETYPE>&, const std::string)`
- 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<VALUETYPE>&, const int, const int, const std::string)`

## Typedefs

- Typedef `deepmd::ENERGYTYPE`
- Typedef `deepmd::STRINGTYPE`
- Typedef `deepmd::VALUETYPE`

## Namespace tensorflow

### 17.3.2 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 InputNlist

- Defined in file `_source_api_cc_include_common.h`

## Struct Documentation

struct **InputNlist**

## 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<VALUETYPE> &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)
```

### Template Class `AtomMap`

- Defined in `file_source_api_cc_include_AtomMap.h`

### Class Documentation

```
template<typename VALUETYPE>
```

```
class AtomMap
```

#### Public Functions

```
AtomMap()
```

```
AtomMap(const std::vector<int>::const_iterator in_begin, const std::vector<int>::const_iterator  
in_end)
```

```
void forward(typename std::vector<VALUETYPE>::iterator out, const typename  
std::vector<VALUETYPE>::const_iterator in, const int stride = 1) const
```

```
void backward(typename std::vector<VALUETYPE>::iterator out, const typename
              std::vector<VALUETYPE>::const_iterator in, const int stride = 1) 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.



```
void compute(ENERGYTYPE &ener, std::vector<VALUETYPE> &force,
             std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const
             std::vector<int> &atype, const std::vector<VALUETYPE> &box, const
             std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const
             std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())
```

Evaluate the energy, force and virial by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam. dim\_aparam. Then all frames and atoms are provided with the same aparam.

```
void compute(ENERGYTYPE &ener, std::vector<VALUETYPE> &force,
             std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const
             std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost,
             const InputNlist &inlist, const int &ago, const std::vector<VALUETYPE> &fparam =
             std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam =
             std::vector<VALUETYPE>())
```

Evaluate the energy, force and virial by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.

- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam. dim\_aparam. Then all frames and atoms are provided with the same aparam.

```
void compute(ENERGYTYPE &ener, std::vector<VALUETYPE> &force,
             std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy,
             std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord,
             const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const
             std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const
             std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())
```

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_energy** – [out] The atomic energy.
- **atom\_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam. dim\_aparam. Then all frames and atoms are provided with the same aparam.

```
void compute(ENERGYTYPE &ener, std::vector<VALUETYPE> &force,
             std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy,
             std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord,
             const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int
             nghost, const InputNlist &imp_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. dim\_aparam. Then all frames and atoms are provided with the same aparam.

inline **VALUETYPE 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 **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 contructor.

Parameters

- **models** – [in] The names of the frozen model files.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **file\_contents** – [in] The contents of the model files. If it is not empty, DP will read from the strings instead of the files.

`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.

```
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 VALUETYPE 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 **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 **compute\_avg**(**ENERGYTYPE** &dener, const std::vector<**ENERGYTYPE**> &all\_energy)

Compute the average energy.

Parameters

- **dener** – [out] The average energy.
- **all\_energy** – [in] The energies of all models.

void **compute\_avg**(**VALUETYPE** &dener, const std::vector<**VALUETYPE**> &all\_energy)

Compute the average energy.

Parameters

- **dener** – [out] The average energy.
- **all\_energy** – [in] The energies of all models.

void **compute\_avg**(std::vector<**VALUETYPE**> &avg, const  
std::vector<std::vector<**VALUETYPE**>> &xx)

Compute the average of vectors.

Parameters

- **avg** – [out] The average of vectors.
- **xx** – [in] The vectors of all models.

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.

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.

```
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.

```
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.

```
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(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.

`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.

`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.



```
void compute(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force,
            std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const
            std::vector<int> &atype, const std::vector<VALUETYPE> &box)
```

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global\_tensor** – [out] The global tensor to evaluate.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.

```
void compute(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force,
            std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const
            std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost,
            const InputNlist &inlist)
```

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global\_tensor** – [out] The global tensor to evaluate.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.

```
void compute(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force,
            std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_tensor,
            std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord,
            const std::vector<int> &atype, const std::vector<VALUETYPE> &box)
```

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global\_tensor** – [out] The global tensor to evaluate.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **atom\_tensor** – [out] The atomic tensor value of the model, size natoms x odim.
- **atom\_virial** – [out] The component-wise atomic virial of the global tensor, size odim x natoms x 9.

- **coord** – [in] The coordinates of atoms. The array should be of size `natoms x 3`.
- **atype** – [in] The atom types. The list should contain `natoms` ints.
- **box** – [in] The cell of the region. The array should be of size 9.

```
void compute(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force,  
            std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_tensor,  
            std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord,  
            const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int  
            nghost, const InputNlist &inlist)
```

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global\_tensor** – [out] The global tensor to evaluate.
- **force** – [out] The component-wise force of the global tensor, size `odim x natoms x 3`.
- **virial** – [out] The component-wise virial of the global tensor, size `odim x 9`.
- **atom\_tensor** – [out] The atomic tensor value of the model, size `natoms x odim`.
- **atom\_virial** – [out] The component-wise atomic virial of the global tensor, size `odim x natoms x 9`.
- **coord** – [in] The coordinates of atoms. The array should be of size `natoms x 3`.
- **atype** – [in] The atom types. The list should contain `natoms` ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.

```
inline VALUETYPE 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
```

## Class DipoleChargeModifier

- Defined in file `_source_api_cc_include_DataModifier.h`

## Class Documentation

class `DipoleChargeModifier`

### Public Functions

`DipoleChargeModifier()`

`DipoleChargeModifier(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")`

`inline ~DipoleChargeModifier()`

`void init(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")`

`void print_summary(const std::string &pre) const`

`void compute(std::vector<VALUETYPE> &dfcorr_, std::vector<VALUETYPE> &dvcorr_, const std::vector<VALUETYPE> &dcoord_, const std::vector<int> &dtype_, const std::vector<VALUETYPE> &dbox, const std::vector<std::pair<int, int>> &pairs, const std::vector<VALUETYPE> &delef_, const int nghost, const InputNlist &imp_list)`

`inline VALUETYPE cutoff() const`

`inline int numb_types() const`

`inline std::vector<int> sel_types() const`

## 17.3.3 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`** – [out] 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\_compatible**

- Defined in file\_source\_api\_cc\_include\_common.h

**Function Documentation**

bool deepmd::model\_compatible(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::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.

**Function deepmd::select\_by\_type**

- Defined in file\_source\_api\_cc\_include\_common.h

**Function Documentation**

```
void deepmd::select_by_type(std::vector<int> &fwd_map, std::vector<int> &bkw_map, int
                           &nghost_real, const std::vector<VALUE_TYPE> &dcoord_, const
                           std::vector<int> &dtype_, const int &nghost, const std::vector<int>
                           &sel_type_)
```

**Template Function** `deepmd::select_map(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(std::vector<VT> &out, const std::vector<VT> &in, const std::vector<int>
                        &fwd_map, const int &stride)
```

**Template Function** `deepmd::select_map(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(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_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)
```

## Function deepmd::select\_real\_atoms

- Defined in file\_source\_api\_cc\_include\_common.h

## Function Documentation

```
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> &dtype_, const int &nghost, const int &ntypes)
```

## 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 = "")
```

## 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 = "")
```

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 VALUETYPE&`, `const std::vector<VALUETYPE>&`, `const std::vector<VALUETYPE>&`, `const deepmd::AtomMap<VALUETYPE>&`, `const std::string`)

- Defined in `file_source_api_cc_include_common.h`

### Function Documentation

```
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> &dtype_, const
    std::vector<VALUETYPE> &dbox, const VALUETYPE &cell_size,
    const std::vector<VALUETYPE> &fparam_, const
    std::vector<VALUETYPE> &aparam_, const
    deepmd::AtomMap<VALUETYPE> &atommap, const std::string
    scope = "")
```

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<VALUETYPE>&`, `const int`, `const int`, `const std::string`)

- Defined in `file_source_api_cc_include_common.h`

### Function Documentation

```
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> &dtype_, const
    std::vector<VALUETYPE> &dbox, InputNlist &dlist, const
    std::vector<VALUETYPE> &fparam_, const
    std::vector<VALUETYPE> &aparam_, const
    deepmd::AtomMap<VALUETYPE> &atommap, const int nghost,
    const int ago, const std::string scope = "")
```

## 17.3.4 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`

### Typedef deepmd::VALUETYPE

- Defined in `file_source_api_cc_include_common.h`

### Typedef Documentation

`typedef float deepmd::VALUETYPE`



---

## CHAPTER EIGHTEEN

---

### LICENSE

The project DeePMD-kit is licensed under [GNU LGPLv3.0](#).



## AUTHORS AND CREDITS

### 19.1 Package Contributors

- AnguseZhang
- baohan
- bwang-ecnu
- denghuilu
- frankhan91
- GeiduanLiu
- gzb942560379
- Han Wang
- haidi-ustc
- hlyang1992
- hsulab
- iProzd
- Jiequn Han
- JiabinYang
- jxxiaoshaoye
- Linfeng Zhang
- marian-code
- njzjz
- Nick Lin
- pkulzy
- Shaochen Shi
- tuoping
- wsyxbcl
- Xia, Yu
- Ye Ding

- Yingze Wang
- Yixiao Chen
- YWolfeee
- Zhanlue Yang
- zhouwei25
- ZiyaoLi

## 19.2 Other Credits

- Zhang ZiXuan for designing the Deepmodeling logo.
- Everyone on the Deepmodeling mailing list for contributing to many discussions and decisions!

(If you have contributed to the `deepmd-kit` core package and your name is missing, please send an email to the contributors, or open a pull request in the [deepmd-kit repository](#))

- genindex
- modindex
- search

## BIBLIOGRAPHY

- [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. Identity mappings in deep residual networks. In *Computer Vision – ECCV 2016*, pages 630–645. Springer International Publishing, 2016.





## PYTHON MODULE INDEX

### d

deepmd, 127  
deepmd.calculator, 265  
deepmd.cluster, 130  
deepmd.cluster.local, 131  
deepmd.cluster.slurm, 131  
deepmd.common, 268  
deepmd.descriptor, 131  
deepmd.descriptor.descriptor, 131  
deepmd.descriptor.hybrid, 137  
deepmd.descriptor.loc\_frame, 140  
deepmd.descriptor.se, 143  
deepmd.descriptor.se\_a, 145  
deepmd.descriptor.se\_a\_ebd, 149  
deepmd.descriptor.se\_a\_ef, 151  
deepmd.descriptor.se\_r, 155  
deepmd.descriptor.se\_t, 158  
deepmd.entrypoints, 161  
deepmd.entrypoints.compress, 164  
deepmd.entrypoints.config, 165  
deepmd.entrypoints.convert, 165  
deepmd.entrypoints.doc, 165  
deepmd.entrypoints.freeze, 165  
deepmd.entrypoints.main, 166  
deepmd.entrypoints.neighbor\_stat, 166  
deepmd.entrypoints.test, 167  
deepmd.entrypoints.train, 167  
deepmd.entrypoints.transfer, 168  
deepmd.env, 273  
deepmd.fit, 168  
deepmd.fit.dipole, 168  
deepmd.fit.ener, 170  
deepmd.fit.fitting, 172  
deepmd.fit.polar, 173  
deepmd.fit.wfc, 177  
deepmd.infer, 177  
deepmd.infer.data\_modifier, 189  
deepmd.infer.deep\_dipole, 191  
deepmd.infer.deep\_eval, 192  
deepmd.infer.deep\_polar, 193  
deepmd.infer.deep\_pot, 195  
deepmd.infer.deep\_tensor, 198  
deepmd.infer.deep\_wfc, 200  
deepmd.infer.ewald\_recip, 201  
deepmd.infer.model\_devi, 202  
deepmd.loggers, 204  
deepmd.loggers.loggers, 204  
deepmd.loss, 205  
deepmd.loss.ener, 205  
deepmd.loss.loss, 207  
deepmd.loss.tensor, 209  
deepmd.model, 210  
deepmd.model.ener, 210  
deepmd.model.model, 211  
deepmd.model.model\_stat, 212  
deepmd.model.tensor, 212  
deepmd.nvnmd, 215  
deepmd.nvnmd.data, 215  
deepmd.nvnmd.data.data, 216  
deepmd.nvnmd.descriptor, 216  
deepmd.nvnmd.descriptor.se\_a, 216  
deepmd.nvnmd.entrypoints, 216  
deepmd.nvnmd.entrypoints.freeze, 219  
deepmd.nvnmd.entrypoints.mapt, 219  
deepmd.nvnmd.entrypoints.train, 221  
deepmd.nvnmd.entrypoints.wrap, 221  
deepmd.nvnmd.fit, 222  
deepmd.nvnmd.fit.ener, 222  
deepmd.nvnmd.utils, 222  
deepmd.nvnmd.utils.argcheck, 226  
deepmd.nvnmd.utils.config, 226  
deepmd.nvnmd.utils.encode, 228  
deepmd.nvnmd.utils.fio, 230  
deepmd.nvnmd.utils.network, 232  
deepmd.nvnmd.utils.op, 233  
deepmd.nvnmd.utils.weight, 233  
deepmd.op, 234  
deepmd.train, 234  
deepmd.train.run\_options, 234  
deepmd.train.trainer, 235  
deepmd.utils, 236  
deepmd.utils.argcheck, 236  
deepmd.utils.batch\_size, 238  
deepmd.utils.compat, 239

`deepmd.utils.convert`, 240  
`deepmd.utils.data`, 242  
`deepmd.utils.data_system`, 246  
`deepmd.utils.errors`, 249  
`deepmd.utils.graph`, 249  
`deepmd.utils.learning_rate`, 252  
`deepmd.utils.neighbor_stat`, 253  
`deepmd.utils.network`, 254  
`deepmd.utils.pair_tab`, 255  
`deepmd.utils.parallel_op`, 256  
`deepmd.utils.path`, 257  
`deepmd.utils.plugin`, 260  
`deepmd.utils.random`, 262  
`deepmd.utils.sess`, 262  
`deepmd.utils.tabulate`, 263  
`deepmd.utils.type_embed`, 264  
`deepmd.utils.weight_avg`, 265

## INDEX

### A

[add\(\)](#) (deepmd.common.ClassArg method), 268  
[add\(\)](#) (deepmd.utils.data.DeepmdData method), 244  
[add\(\)](#) (deepmd.utils.data\_system.DeepmdDataSystem method), 247  
[add\\_data\\_requirement\(\)](#) (in module deepmd.common), 269  
[add\\_dict\(\)](#) (deepmd.utils.data\_system.DeepmdDataSystem method), 248  
[add\\_path\(\)](#) (in module deepmd.nvnmd.entrypoints.train), 221  
[ArgsPlugin](#) (class in deepmd.utils.argcheck), 236  
[AutoBatchSize](#) (class in deepmd.utils.batch\_size), 238  
[avg\(\)](#) (deepmd.utils.data.DeepmdData method), 244

### B

[bin2hex\(\)](#) (deepmd.nvnmd.utils.Encode method), 223  
[bin2hex\(\)](#) (deepmd.nvnmd.utils.encode.Encode method), 229  
[bin2hex\\_str\(\)](#) (deepmd.nvnmd.utils.Encode method), 223  
[bin2hex\\_str\(\)](#) (deepmd.nvnmd.utils.encode.Encode method), 229  
[build\(\)](#) (deepmd.descriptor.descriptor.Descriptor method), 132  
[build\(\)](#) (deepmd.descriptor.hybrid.DescriptHybrid method), 137  
[build\(\)](#) (deepmd.descriptor.loc\_frame.DescriptLocFrame method), 141  
[build\(\)](#) (deepmd.descriptor.se\_a.DescriptSeA method), 146  
[build\(\)](#) (deepmd.descriptor.se\_a\_ebd.DescriptSeAEbd method), 150  
[build\(\)](#) (deepmd.descriptor.se\_a\_ef.DescriptSeAEf method), 152  
[build\(\)](#) (deepmd.descriptor.se\_a\_ef.DescriptSeAEfLower method), 154  
[build\(\)](#) (deepmd.descriptor.se\_r.DescriptSeR method), 156

[build\(\)](#) (deepmd.descriptor.se\_t.DescriptSeT method), 159  
[build\(\)](#) (deepmd.fit.dipole.DipoleFittingSeA method), 169  
[build\(\)](#) (deepmd.fit.ener.EnerFitting method), 171  
[build\(\)](#) (deepmd.fit.polar.GlobalPolarFittingSeA method), 174  
[build\(\)](#) (deepmd.fit.polar.PolarFittingLocFrame method), 175  
[build\(\)](#) (deepmd.fit.polar.PolarFittingSeA method), 176  
[build\(\)](#) (deepmd.fit.wfc.WFCFitting method), 177  
[build\(\)](#) (deepmd.loss.ener.EnerDipoleLoss method), 206  
[build\(\)](#) (deepmd.loss.ener.EnerStdLoss method), 207  
[build\(\)](#) (deepmd.loss.loss.Loss method), 208  
[build\(\)](#) (deepmd.loss.tensor.TensorLoss method), 209  
[build\(\)](#) (deepmd.model.ener.EnerModel method), 210  
[build\(\)](#) (deepmd.model.tensor.TensorModel method), 214  
[build\(\)](#) (deepmd.train.trainer.DPTrainer method), 235  
[build\(\)](#) (deepmd.utils.learning\_rate.LearningRateExp method), 253  
[build\(\)](#) (deepmd.utils.tabulate.DPTabulate method), 263  
[build\(\)](#) (deepmd.utils.type\_embed.TypeEmbedNet method), 264  
[build\\_davg\\_dstd\(\)](#) (in module deepmd.nvnmd.descriptor.se\_a), 216  
[build\\_dG\\_ds\(\)](#) (deepmd.nvnmd.entrypoints.maptable.MapTable method), 220  
[build\\_dG\\_ds\(\)](#) (deepmd.nvnmd.entrypoints.MapTable method), 217  
[build\\_ds\\_dr\(\)](#) (deepmd.nvnmd.entrypoints.maptable.MapTable method), 220  
[build\\_ds\\_dr\(\)](#) (deepmd.nvnmd.entrypoints.MapTable method), 217  
[build\\_fv\\_graph\(\)](#) (deepmd.DipoleChargeModifier

method), 129

build\_fv\_graph() (deepmd.infer.data\_modifier.DipoleChargeModifier method), 190

build\_fv\_graph() (deepmd.infer.DipoleChargeModifier method), 187

build\_map() (deepmd.nvnmd.entrypoints.map.MapTable method), 220

build\_map() (deepmd.nvnmd.entrypoints.MapTable method), 217

build\_op\_descriptor() (in module deepmd.nvnmd.descriptor.se\_a), 216

build\_r2s() (deepmd.nvnmd.entrypoints.map.MapTable method), 220

build\_r2s() (deepmd.nvnmd.entrypoints.MapTable method), 217

build\_r2s\_r2ds() (deepmd.nvnmd.entrypoints.map.MapTable method), 220

build\_r2s\_r2ds() (deepmd.nvnmd.entrypoints.MapTable method), 217

build\_s2G() (deepmd.nvnmd.entrypoints.map.MapTable method), 220

build\_s2G() (deepmd.nvnmd.entrypoints.MapTable method), 217

build\_s2G\_s2dG() (deepmd.nvnmd.entrypoints.map.MapTable method), 220

build\_s2G\_s2dG() (deepmd.nvnmd.entrypoints.MapTable method), 217

## C

calc\_model\_devi() (in module deepmd.infer), 188

calc\_model\_devi() (in module deepmd.infer.model\_devi), 202

calc\_model\_devi\_e() (in module deepmd.infer.model\_devi), 203

calc\_model\_devi\_f() (in module deepmd.infer.model\_devi), 203

calc\_model\_devi\_v() (in module deepmd.infer.model\_devi), 203

calculate() (deepmd.calculator.DP method), 267

cast\_precision() (in module deepmd.common), 269

check\_batch\_size() (deepmd.utils.data.DataSets method), 242

check\_batch\_size() (deepmd.utils.data.DeepmdData method), 244

check\_dec() (deepmd.nvnmd.utils.Encode method), 223

check\_dec() (deepmd.nvnmd.utils.encode.Encode method), 229

check\_test\_size() (deepmd.utils.data.DataSets method), 242

check\_test\_size() (deepmd.utils.data.DeepmdData method), 244

check\_type\_map\_consistency() (in module deepmd.common), 268

choice() (in module deepmd.utils.random), 262

ClassArg (class in deepmd.common), 268

compress() (in module deepmd.entrypoints), 161

compress() (in module deepmd.entrypoints.compress), 164

compute\_energy\_shift() (deepmd.utils.data\_system.DataSystem method), 246

compute\_energy\_shift() (deepmd.utils.data\_system.DeepmdDataSystem method), 248

compute\_input\_stats() (deepmd.descriptor.descriptor.Descriptor method), 133

compute\_input\_stats() (deepmd.descriptor.hybrid.DescriptHybrid method), 138

compute\_input\_stats() (deepmd.descriptor.loc\_frame.DescriptLocFrame method), 142

compute\_input\_stats() (deepmd.descriptor.se\_a.DescriptSeA method), 147

compute\_input\_stats() (deepmd.descriptor.se\_a\_ef.DescriptSeAEf method), 152

compute\_input\_stats() (deepmd.descriptor.se\_a\_ef.DescriptSeAEfLower method), 155

compute\_input\_stats() (deepmd.descriptor.se\_r.DescriptSeR method), 157

compute\_input\_stats() (deepmd.descriptor.se\_t.DescriptSeT method), 160

compute\_input\_stats() (deepmd.fit.ener.EnerFitting method), 171

compute\_input\_stats() (deepmd.fit.polar.PolarFittingSeA method), 176

compute\_output\_stats() (deepmd.fit.ener.EnerFitting method), 171

config() (in module deepmd.entrypoints), 162

config() (in module deepmd.entrypoints.config), 165

convert() (in module deepmd.entrypoints), 162

convert() (in module deepmd.entrypoints.convert), 165

convert\_012\_to\_21() (in module

deepmd.utils.convert), 240  
 convert\_10\_to\_21() (in  
 deepmd.utils.convert), 240  
 convert\_12\_to\_21() (in  
 deepmd.utils.convert), 240  
 convert\_13\_to\_21() (in  
 deepmd.utils.convert), 240  
 convert\_20\_to\_21() (in  
 deepmd.utils.convert), 241  
 convert\_dp012\_to\_dp10() (in  
 deepmd.utils.convert), 241  
 convert\_dp10\_to\_dp11() (in  
 deepmd.utils.convert), 241  
 convert\_dp12\_to\_dp13() (in  
 deepmd.utils.convert), 241  
 convert\_dp13\_to\_dp20() (in  
 deepmd.utils.convert), 241  
 convert\_dp20\_to\_dp21() (in  
 deepmd.utils.convert), 241  
 convert\_input\_v0\_v1() (in  
 deepmd.utils.compat), 239  
 convert\_input\_v1\_v2() (in  
 deepmd.utils.compat), 239  
 convert\_pb\_to\_pbtxt() (in  
 deepmd.utils.convert), 241  
 convert\_pbtxt\_to\_pb() (in  
 deepmd.utils.convert), 241  
 create\_file\_path() (deepmd.nvnmd.utils.fio.Fio  
 method), 230

## D

data\_stat() (deepmd.model.ener.EnerModel  
 method), 211  
 data\_stat() (deepmd.model.tensor.TensorModel  
 method), 214  
 DataSets (class in deepmd.utils.data), 242  
 DataSystem (class in deepmd.utils.data\_system), 246  
 dec2bin() (deepmd.nvnmd.utils.Encode method),  
 223  
 dec2bin() (deepmd.nvnmd.utils.encode.Encode  
 method), 229  
 DeepDipole (class in deepmd.infer), 177  
 DeepDipole (class in deepmd.infer.deep\_dipole), 191  
 DeepEval (class in deepmd), 127  
 DeepEval (class in deepmd.infer), 178  
 DeepEval (class in deepmd.infer.deep\_eval), 192  
 DeepGlobalPolar (class in deepmd.infer), 180  
 DeepGlobalPolar (class in  
 deepmd.infer.deep\_polar), 193  
 deepmd  
 module, 127  
 deepmd.calculator  
 module, 265  
 deepmd.cluster  
 module, 130  
 deepmd.cluster.local  
 module, 131  
 deepmd.cluster.slurm  
 module, 131  
 deepmd.common  
 module, 268  
 deepmd.descriptor  
 module, 131  
 deepmd.descriptor.descriptor  
 module, 131  
 deepmd.descriptor.hybrid  
 module, 137  
 deepmd.descriptor.loc\_frame  
 module, 140  
 deepmd.descriptor.se  
 module, 143  
 deepmd.descriptor.se\_a  
 module, 145  
 deepmd.descriptor.se\_a\_ebd  
 module, 149  
 deepmd.descriptor.se\_a\_ef  
 module, 151  
 deepmd.descriptor.se\_r  
 module, 155  
 deepmd.descriptor.se\_t  
 module, 158  
 deepmd.entrypoints  
 module, 161  
 deepmd.entrypoints.compress  
 module, 164  
 deepmd.entrypoints.config  
 module, 165  
 deepmd.entrypoints.convert  
 module, 165  
 deepmd.entrypoints.doc  
 module, 165  
 deepmd.entrypoints.freeze  
 module, 165  
 deepmd.entrypoints.main  
 module, 166  
 deepmd.entrypoints.neighbor\_stat  
 module, 166  
 deepmd.entrypoints.test  
 module, 167  
 deepmd.entrypoints.train  
 module, 167  
 deepmd.entrypoints.transfer  
 module, 168  
 deepmd.env  
 module, 273  
 deepmd.fit  
 module, 168  
 deepmd.fit.dipole

- module, 168
- deepmd.fit.ener
  - module, 170
- deepmd.fit.fitting
  - module, 172
- deepmd.fit.polar
  - module, 173
- deepmd.fit.wfc
  - module, 177
- deepmd.infer
  - module, 177
- deepmd.infer.data\_modifier
  - module, 189
- deepmd.infer.deep\_dipole
  - module, 191
- deepmd.infer.deep\_eval
  - module, 192
- deepmd.infer.deep\_polar
  - module, 193
- deepmd.infer.deep\_pot
  - module, 195
- deepmd.infer.deep\_tensor
  - module, 198
- deepmd.infer.deep\_wfc
  - module, 200
- deepmd.infer.ewald\_recip
  - module, 201
- deepmd.infer.model\_devi
  - module, 202
- deepmd.loggers
  - module, 204
- deepmd.loggers.loggers
  - module, 204
- deepmd.loss
  - module, 205
- deepmd.loss.ener
  - module, 205
- deepmd.loss.loss
  - module, 207
- deepmd.loss.tensor
  - module, 209
- deepmd.model
  - module, 210
- deepmd.model.ener
  - module, 210
- deepmd.model.model
  - module, 211
- deepmd.model.model\_stat
  - module, 212
- deepmd.model.tensor
  - module, 212
- deepmd.nvnmd
  - module, 215
- deepmd.nvnmd.data

- module, 215
- deepmd.nvnmd.data.data
  - module, 216
- deepmd.nvnmd.descriptor
  - module, 216
- deepmd.nvnmd.descriptor.se\_a
  - module, 216
- deepmd.nvnmd.entrypoints
  - module, 216
- deepmd.nvnmd.entrypoints.freeze
  - module, 219
- deepmd.nvnmd.entrypoints.mapt
  - module, 219
- deepmd.nvnmd.entrypoints.train
  - module, 221
- deepmd.nvnmd.entrypoints.wrap
  - module, 221
- deepmd.nvnmd.fit
  - module, 222
- deepmd.nvnmd.fit.ener
  - module, 222
- deepmd.nvnmd.utils
  - module, 222
- deepmd.nvnmd.utils.argcheck
  - module, 226
- deepmd.nvnmd.utils.config
  - module, 226
- deepmd.nvnmd.utils.encode
  - module, 228
- deepmd.nvnmd.utils.fio
  - module, 230
- deepmd.nvnmd.utils.network
  - module, 232
- deepmd.nvnmd.utils.op
  - module, 233
- deepmd.nvnmd.utils.weight
  - module, 233
- deepmd.op
  - module, 234
- deepmd.train
  - module, 234
- deepmd.train.run\_options
  - module, 234
- deepmd.train.trainer
  - module, 235
- deepmd.utils
  - module, 236
- deepmd.utils.argcheck
  - module, 236
- deepmd.utils.batch\_size
  - module, 238
- deepmd.utils.compat
  - module, 239
- deepmd.utils.convert

---

```

    module, 240
deepmd.utils.data
    module, 242
deepmd.utils.data_system
    module, 246
deepmd.utils.errors
    module, 249
deepmd.utils.graph
    module, 249
deepmd.utils.learning_rate
    module, 252
deepmd.utils.neighbor_stat
    module, 253
deepmd.utils.network
    module, 254
deepmd.utils.pair_tab
    module, 255
deepmd.utils.parallel_op
    module, 256
deepmd.utils.path
    module, 257
deepmd.utils.plugin
    module, 260
deepmd.utils.random
    module, 262
deepmd.utils.sess
    module, 262
deepmd.utils.tabulate
    module, 263
deepmd.utils.type_embed
    module, 264
deepmd.utils.weight_avg
    module, 265
deepmd::AtomMap (C++ class), 279
deepmd::AtomMap::AtomMap (C++ function), 279
deepmd::AtomMap::backward (C++ function), 279
deepmd::AtomMap::forward (C++ function), 279
deepmd::AtomMap::get_bkw_map (C++ function),
    280
deepmd::AtomMap::get_fwd_map (C++ function),
    280
deepmd::AtomMap::get_type (C++ function), 280
deepmd::check_status (C++ function), 291
deepmd::convert_pbtxt_to_pb (C++ function),
    292
deepmd::deepmd_exception (C++ struct), 278
deepmd::DeepPot (C++ class), 280
deepmd::DeepPot::~DeepPot (C++ function), 280
deepmd::DeepPot::compute (C++ function), 280--
    282
deepmd::DeepPot::cutoff (C++ function), 283
deepmd::DeepPot::DeepPot (C++ function), 280
deepmd::DeepPot::dim_aparam (C++ function),
    283
deepmd::DeepPot::dim_fparam (C++ function),
    283
deepmd::DeepPot::get_type_map (C++ function),
    283
deepmd::DeepPot::init (C++ function), 280
deepmd::DeepPot::numb_types (C++ function),
    283
deepmd::DeepPot::print_summary (C++ function),
    280
deepmd::DeepPotModelDevi (C++ class), 284
deepmd::DeepPotModelDevi::~DeepPotModelDevi
    (C++ function), 284
deepmd::DeepPotModelDevi::compute (C++ func-
    tion), 284, 285
deepmd::DeepPotModelDevi::compute_avg (C++
    function), 286
deepmd::DeepPotModelDevi::compute_relative_std
    (C++ function), 286
deepmd::DeepPotModelDevi::compute_relative_std_f
    (C++ function), 287
deepmd::DeepPotModelDevi::compute_std (C++
    function), 286
deepmd::DeepPotModelDevi::compute_std_e
    (C++ function), 287
deepmd::DeepPotModelDevi::compute_std_f
    (C++ function), 287
deepmd::DeepPotModelDevi::cutoff (C++ func-
    tion), 285
deepmd::DeepPotModelDevi::DeepPotModelDevi
    (C++ function), 284
deepmd::DeepPotModelDevi::dim_aparam (C++
    function), 286
deepmd::DeepPotModelDevi::dim_fparam (C++
    function), 286
deepmd::DeepPotModelDevi::init (C++ function),
    284
deepmd::DeepPotModelDevi::numb_types (C++
    function), 285
deepmd::DeepTensor (C++ class), 287
deepmd::DeepTensor::compute (C++ function),
    288--290
deepmd::DeepTensor::cutoff (C++ function), 290
deepmd::DeepTensor::DeepTensor (C++ function),
    288
deepmd::DeepTensor::init (C++ function), 288
deepmd::DeepTensor::numb_types (C++ function),
    290
deepmd::DeepTensor::output_dim (C++ function),
    290
deepmd::DeepTensor::print_summary (C++ func-
    tion), 288
deepmd::DeepTensor::sel_types (C++ function),
    290
deepmd::DipoleChargeModifier (C++ class), 291

```



deepmd::DipoleChargeModifier::~~DipoleChargeModifier (C++ function), 291  
 deepmd::DipoleChargeModifier::compute (C++ function), 291  
 deepmd::DipoleChargeModifier::cutoff (C++ function), 291  
 deepmd::DipoleChargeModifier::DipoleChargeModifier (C++ function), 291  
 deepmd::DipoleChargeModifier::init (C++ function), 291  
 deepmd::DipoleChargeModifier::numb\_types (C++ function), 291  
 deepmd::DipoleChargeModifier::print\_summary (C++ function), 291  
 deepmd::DipoleChargeModifier::sel\_types (C++ function), 291  
 deepmd::ENERGYTYPE (C++ type), 297  
 deepmd::get\_env\_nthreads (C++ function), 292  
 deepmd::InputNlist (C++ struct), 278  
 deepmd::load\_op\_library (C++ function), 292  
 deepmd::model\_compatible (C++ function), 293  
 deepmd::name\_prefix (C++ function), 293  
 deepmd::NeighborListData (C++ struct), 278  
 deepmd::NeighborListData::copy\_from\_nlist (C++ function), 278  
 deepmd::NeighborListData::firstneigh (C++ member), 278  
 deepmd::NeighborListData::ilist (C++ member), 278  
 deepmd::NeighborListData::jlist (C++ member), 278  
 deepmd::NeighborListData::make\_inlist (C++ function), 278  
 deepmd::NeighborListData::numneigh (C++ member), 278  
 deepmd::NeighborListData::shuffle (C++ function), 278  
 deepmd::NeighborListData::shuffle\_exclude\_empty (C++ function), 278  
 deepmd::read\_file\_to\_string (C++ function), 293  
 deepmd::select\_by\_type (C++ function), 293  
 deepmd::select\_map (C++ function), 294  
 deepmd::select\_map\_inv (C++ function), 294, 295  
 deepmd::select\_real\_atoms (C++ function), 295  
 deepmd::session\_get\_scalar (C++ function), 295  
 deepmd::session\_get\_vector (C++ function), 295  
 deepmd::session\_input\_tensors (C++ function), 296  
 deepmd::STRINGTYPE (C++ type), 297  
 deepmd::tf\_exception (C++ struct), 279  
 deepmd::tf\_exception::tf\_exception (C++ function), 279  
 deepmd::VALUETYPE (C++ type), 297  
 DeepPMDData (class in deepmd.utils.data), 243  
 DeepPMDDataSystem (class in deepmd.utils.data\_system), 247  
 DeepPolar (class in deepmd.infer), 181  
 DeepPolar (class in deepmd.infer.deep\_polar), 194  
 DeepPot (class in deepmd.infer), 182  
 DeepPot (class in deepmd.infer.deep\_pot), 195  
 DeepPotential() (in module deepmd), 128  
 DeepPotential() (in module deepmd.infer), 185  
 DeepTensor (class in deepmd.infer.deep\_tensor), 198  
 DeepWFC (class in deepmd.infer), 185  
 DeepWFC (class in deepmd.infer.deep\_wfc), 200  
 deprecate\_numb\_test() (in module deepmd.utils.compat), 239  
 Descriptor (class in deepmd.descriptor.descriptor), 131  
 descrpt2r4() (in module deepmd.nvnmd.descriptor.se\_a), 216  
 descrpt\_hybrid\_args() (in module deepmd.utils.argcheck), 237  
 descrpt\_local\_frame\_args() (in module deepmd.utils.argcheck), 237  
 descrpt\_se\_a\_args() (in module deepmd.utils.argcheck), 237  
 descrpt\_se\_a\_tpe\_args() (in module deepmd.utils.argcheck), 237  
 descrpt\_se\_r\_args() (in module deepmd.utils.argcheck), 237  
 descrpt\_se\_t\_args() (in module deepmd.utils.argcheck), 237  
 descrpt\_variant\_type\_args() (in module deepmd.utils.argcheck), 237  
 DescrptHybrid (class in deepmd.descriptor.hybrid), 137  
 DescrptLocFrame (class in deepmd.descriptor.loc\_frame), 140  
 DescrptSe (class in deepmd.descriptor.se), 143  
 DescrptSeA (class in deepmd.descriptor.se\_a), 145  
 DescrptSeAEbd (class in deepmd.descriptor.se\_a\_ebd), 149  
 DescrptSeAEf (class in deepmd.descriptor.se\_a\_ef), 151  
 DescrptSeAEfLower (class in deepmd.descriptor.se\_a\_ef), 154  
 DescrptSeR (class in deepmd.descriptor.se\_r), 155  
 DescrptSeT (class in deepmd.descriptor.se\_t), 158  
 DipoleChargeModifier (class in deepmd), 129  
 DipoleChargeModifier (class in deepmd.infer), 186  
 DipoleChargeModifier (class in deepmd.infer.data\_modifier), 189  
 DipoleFittingSeA (class in deepmd.fit.dipole), 168  
 DipoleModel (class in deepmd.model.tensor), 212  
 disp\_message() (deepmd.nvnmd.utils.config.NvnmdConfig method), 227



`doc_train_input()` (in module `deepmd.entypoints`), 162  
`doc_train_input()` (in module `deepmd.entypoints.doc`), 165  
`DP` (class in `deepmd.calculator`), 265  
`DPH5Path` (class in `deepmd.utils.path`), 257  
`DPOSPath` (class in `deepmd.utils.path`), 258  
`DPPPath` (class in `deepmd.utils.path`), 259  
`DPTabulate` (class in `deepmd.utils.tabulate`), 263  
`DPTrainer` (class in `deepmd.train.trainer`), 235  
**E**  
`embed_atom_type()` (in module `deepmd.utils.type_embed`), 265  
`embedding_net()` (in module `deepmd.utils.network`), 254  
`embedding_net_rand_seed_shift()` (in module `deepmd.utils.network`), 255  
`enable_compression()` (`deepmd.descriptor.descriptor.Descriptor` method), 133  
`enable_compression()` (`deepmd.descriptor.hybrid.DescriptHybrid` method), 138  
`enable_compression()` (`deepmd.descriptor.se_a.DescriptSeA` method), 147  
`enable_compression()` (`deepmd.descriptor.se_r.DescriptSeR` method), 157  
`enable_compression()` (`deepmd.descriptor.se_t.DescriptSeT` method), 160  
`enable_compression()` (`deepmd.fit.ener.EnerFitting` method), 172  
`enable_mixed_precision()` (`deepmd.descriptor.descriptor.Descriptor` method), 134  
`enable_mixed_precision()` (`deepmd.descriptor.hybrid.DescriptHybrid` method), 139  
`enable_mixed_precision()` (`deepmd.descriptor.se_a.DescriptSeA` method), 148  
`enable_mixed_precision()` (`deepmd.fit.dipole.DipoleFittingSeA` method), 169  
`enable_mixed_precision()` (`deepmd.fit.ener.EnerFitting` method), 172  
`enable_mixed_precision()` (`deepmd.fit.polar.GlobalPolarFittingSeA` method), 174  
`enable_mixed_precision()` (`deepmd.fit.polar.PolarFittingSeA` method), 176  
`Encode` (class in `deepmd.nvnmd.utils`), 222  
`Encode` (class in `deepmd.nvnmd.utils.encode`), 228  
`EnerDipoleLoss` (class in `deepmd.loss.ener`), 205  
`EnerFitting` (class in `deepmd.fit.ener`), 170  
`EnerModel` (class in `deepmd.model.ener`), 210  
`EnerStdLoss` (class in `deepmd.loss.ener`), 206  
`eval()` (`deepmd.DipoleChargeModifier` method), 130  
`eval()` (`deepmd.infer.data_modifier.DipoleChargeModifier` method), 190  
`eval()` (`deepmd.infer.deep_polar.DeepGlobalPolar` method), 194  
`eval()` (`deepmd.infer.deep_pot.DeepPot` method), 196  
`eval()` (`deepmd.infer.deep_tensor.DeepTensor` method), 199  
`eval()` (`deepmd.infer.DeepGlobalPolar` method), 180  
`eval()` (`deepmd.infer.DeepPot` method), 183  
`eval()` (`deepmd.infer.DipoleChargeModifier` method), 187  
`eval()` (`deepmd.infer.ewald_recip.EwaldRecp` method), 202  
`eval()` (`deepmd.infer.EwaldRecp` method), 188  
`eval()` (`deepmd.loss.ener.EnerDipoleLoss` method), 206  
`eval()` (`deepmd.loss.ener.EnerStdLoss` method), 207  
`eval()` (`deepmd.loss.loss.Loss` method), 208  
`eval()` (`deepmd.loss.tensor.TensorLoss` method), 209  
`eval_descriptor()` (`deepmd.infer.deep_pot.DeepPot` method), 197  
`eval_descriptor()` (`deepmd.infer.DeepPot` method), 184  
`eval_full()` (`deepmd.infer.deep_tensor.DeepTensor` method), 199  
`EwaldRecp` (class in `deepmd.infer`), 187  
`EwaldRecp` (class in `deepmd.infer.ewald_recip`), 201  
`execute()` (`deepmd.utils.batch_size.AutoBatchSize` method), 238  
`execute_all()` (`deepmd.utils.batch_size.AutoBatchSize` method), 239  
`exits()` (`deepmd.nvnmd.utils.fio.Fio` method), 231  
`expand_sys_str()` (in module `deepmd.common`), 270  
`extend_bin()` (`deepmd.nvnmd.utils.Encode` method), 223  
`extend_bin()` (`deepmd.nvnmd.utils.encode.Encode` method), 229  
`extend_hex()` (`deepmd.nvnmd.utils.Encode` method), 224  
`extend_hex()` (`deepmd.nvnmd.utils.encode.Encode` method), 229  
`extend_list()` (`deepmd.nvnmd.utils.Encode`

method), 224  
 extend\_list() (deepmd.nvnmd.utils.encode.Encode  
 method), 229

## F

filter\_GR2D() (in module  
 deepmd.nvnmd.descriptor.se\_a), 216  
 filter\_lower\_R42GR() (in module  
 deepmd.nvnmd.descriptor.se\_a), 216  
 filter\_tensorVariableList() (in module  
 deepmd.nvnmd.entrypoints.freeze), 219  
 Fio (class in deepmd.nvnmd.utils.fio), 230  
 FioBin (class in deepmd.nvnmd.utils), 224  
 FioBin (class in deepmd.nvnmd.utils.fio), 231  
 FioDic (class in deepmd.nvnmd.utils), 225  
 FioDic (class in deepmd.nvnmd.utils.fio), 231  
 FioJsonDic (class in deepmd.nvnmd.utils.fio), 231  
 FioNpyDic (class in deepmd.nvnmd.utils.fio), 232  
 FioTxt (class in deepmd.nvnmd.utils), 225  
 FioTxt (class in deepmd.nvnmd.utils.fio), 232  
 Fitting (class in deepmd.fit.fitting), 172  
 fitting\_dipole() (in module  
 deepmd.utils.argcheck), 237  
 fitting\_ener() (in module deepmd.utils.argcheck),  
 237  
 fitting\_polar() (in module  
 deepmd.utils.argcheck), 237  
 fitting\_variant\_type\_args() (in module  
 deepmd.utils.argcheck), 237  
 format\_name\_length()  
 (deepmd.utils.data\_system.DataSystem  
 method), 246  
 freeze() (in module deepmd.entrypoints), 162  
 freeze() (in module deepmd.entrypoints.freeze),  
 165

## G

gelu() (in module deepmd.common), 270  
 gelu\_tf() (in module deepmd.common), 270  
 gen\_doc() (in module deepmd.utils.argcheck), 237  
 gen\_json() (in module deepmd.utils.argcheck), 237  
 generate() (deepmd.utils.parallel\_op.ParallelOp  
 method), 257  
 get() (deepmd.nvnmd.utils.fio.FioDic method), 231  
 get() (deepmd.nvnmd.utils.FioDic method), 225  
 get() (deepmd.utils.pair\_tab.PairTab method), 256  
 get\_activation\_func() (in module  
 deepmd.common), 271  
 get\_all\_argument() (deepmd.utils.argcheck.ArgsPlug  
 method), 236  
 get\_atom\_type() (deepmd.utils.data.DeepmdData  
 method), 245  
 get\_batch() (deepmd.utils.data.DataSets method),  
 242

get\_batch() (deepmd.utils.data.DeepmdData  
 method), 245  
 get\_batch() (deepmd.utils.data\_system.DataSystem  
 method), 246  
 get\_batch() (deepmd.utils.data\_system.DeepmdDataSystem  
 method), 248  
 get\_batch\_size() (deepmd.utils.data\_system.DataSystem  
 method), 246  
 get\_batch\_size() (deepmd.utils.data\_system.DeepmdDataSystem  
 method), 248  
 get\_constant\_initializer() (in module  
 deepmd.nvnmd.utils.weight), 233  
 get\_data\_dict() (deepmd.utils.data.DeepmdData  
 method), 245  
 get\_data\_dict() (deepmd.utils.data\_system.DeepmdDataSystem  
 method), 248  
 get\_deepmd\_jdata() (deepmd.nvnmd.utils.config.NvnmdConfig  
 method), 227  
 get\_dict() (deepmd.common.ClassArg method),  
 268  
 get\_dim\_aparam() (deepmd.infer.deep\_dipole.DeepDipole  
 method), 191  
 get\_dim\_aparam() (deepmd.infer.deep\_polar.DeepGlobalPolar  
 method), 194  
 get\_dim\_aparam() (deepmd.infer.deep\_polar.DeepPolar  
 method), 195  
 get\_dim\_aparam() (deepmd.infer.deep\_pot.DeepPot  
 method), 197  
 get\_dim\_aparam() (deepmd.infer.deep\_tensor.DeepTensor  
 method), 200  
 get\_dim\_aparam() (deepmd.infer.deep\_wfc.DeepWFC  
 method), 201  
 get\_dim\_aparam() (deepmd.infer.DeepDipole  
 method), 178  
 get\_dim\_aparam() (deepmd.infer.DeepGlobalPolar  
 method), 181  
 get\_dim\_aparam() (deepmd.infer.DeepPolar  
 method), 182  
 get\_dim\_aparam() (deepmd.infer.DeepPot method),  
 184  
 get\_dim\_aparam() (deepmd.infer.DeepWFC  
 method), 186  
 get\_dim\_fparam() (deepmd.infer.deep\_dipole.DeepDipole  
 method), 191  
 get\_dim\_fparam() (deepmd.infer.deep\_polar.DeepGlobalPolar  
 method), 194  
 get\_dim\_fparam() (deepmd.infer.deep\_polar.DeepPolar  
 method), 195  
 get\_dim\_fparam() (deepmd.infer.deep\_pot.DeepPot  
 method), 198  
 get\_dim\_fparam() (deepmd.infer.deep\_tensor.DeepTensor  
 method), 200  
 get\_dim\_fparam() (deepmd.infer.deep\_wfc.DeepWFC  
 method), 201

<code>get_dim_fparam()</code> (deepmd.infer.DeepDipole method), 178	<code>get_filter_weight()</code> (in module deepmd.nvnmd.utils), 225
<code>get_dim_fparam()</code> (deepmd.infer.DeepGlobalPolar method), 181	<code>get_filter_weight()</code> (in module deepmd.nvnmd.utils.weight), 233
<code>get_dim_fparam()</code> (deepmd.infer.DeepPolar method), 182	<code>get_fitn_jdata()</code> (deepmd.nvnmd.utils.config.NvnmdConfig method), 227
<code>get_dim_fparam()</code> (deepmd.infer.DeepPot method), 184	<code>get_fitnet_weight()</code> (in module deepmd.nvnmd.utils), 225
<code>get_dim_fparam()</code> (deepmd.infer.DeepWFC method), 186	<code>get_fitnet_weight()</code> (in module deepmd.nvnmd.utils.weight), 233
<code>get_dim_out()</code> (deepmd.descriptor.descriptor.Descriptor method), 134	<code>get_fitting_net_nodes()</code> (in module deepmd.utils.graph), 250
<code>get_dim_out()</code> (deepmd.descriptor.hybrid.DescriptorHybrid method), 139	<code>get_fitting_net_nodes_from_graph_def()</code> (in module deepmd.utils.graph), 250
<code>get_dim_out()</code> (deepmd.descriptor.loc_frame.DescriptorLocFrame method), 142	<code>get_fitting_net_variables()</code> (in module deepmd.utils.graph), 250
<code>get_dim_out()</code> (deepmd.descriptor.se_a.DescriptorSeA method), 148	<code>get_fitting_net_variables_from_graph_def()</code> (in module deepmd.utils.graph), 250
<code>get_dim_out()</code> (deepmd.descriptor.se_a_ef.DescriptorSeAEf method), 153	<code>get_global_step()</code> (deepmd.train.trainer.DPTrainer method), 235
<code>get_dim_out()</code> (deepmd.descriptor.se_r.DescriptorSeR method), 157	<code>get_gpus()</code> (in module deepmd.cluster.local), 131
<code>get_dim_out()</code> (deepmd.descriptor.se_t.DescriptorSeT method), 160	<code>get_learning_rate_jdata()</code> (deepmd.nvnmd.utils.config.NvnmdConfig method), 227
<code>get_dim_rot_mat_1()</code> (deepmd.descriptor.descriptor.Descriptor method), 134	<code>get_ll()</code> (in module deepmd.entrypoints.main), 166
<code>get_dim_rot_mat_1()</code> (deepmd.descriptor.se_a.DescriptorSeA method), 148	<code>get_loss_jdata()</code> (deepmd.nvnmd.utils.config.NvnmdConfig method), 227
<code>get_dim_rot_mat_1()</code> (deepmd.descriptor.se_a_ef.DescriptorSeAEf method), 153	<code>get_model_jdata()</code> (deepmd.nvnmd.utils.config.NvnmdConfig method), 227
<code>get_dscp_jdata()</code> (deepmd.nvnmd.utils.config.NvnmdConfig method), 227	<code>get_natoms()</code> (deepmd.utils.data.DataSets method), 242
<code>get_embedding_net_nodes()</code> (in module deepmd.utils.graph), 249	<code>get_natoms()</code> (deepmd.utils.data.DeepmdData method), 245
<code>get_embedding_net_nodes_from_graph_def()</code> (in module deepmd.utils.graph), 249	<code>get_natoms_2()</code> (deepmd.utils.data.DataSets method), 242
<code>get_embedding_net_variables()</code> (in module deepmd.utils.graph), 249	<code>get_natoms_vec()</code> (deepmd.utils.data.DataSets method), 243
<code>get_embedding_net_variables_from_graph_def()</code> (in module deepmd.utils.graph), 250	<code>get_natoms_vec()</code> (deepmd.utils.data.DeepmdData method), 245
<code>get_ener()</code> (deepmd.utils.data.DataSets method), 242	<code>get_nbatches()</code> (deepmd.utils.data_system.DataSystem method), 246
<code>get_evaluation_results()</code> (deepmd.train.trainer.DPTrainer method), 235	<code>get_nbatches()</code> (deepmd.utils.data_system.DeepmdDataSystem method), 248
<code>get_feed_dict()</code> (deepmd.descriptor.descriptor.Descriptor method), 134	<code>get_nlist()</code> (deepmd.descriptor.descriptor.Descriptor method), 135
<code>get_feed_dict()</code> (deepmd.train.trainer.DPTrainer method), 235	<code>get_nlist()</code> (deepmd.descriptor.loc_frame.DescriptorLocFrame method), 142
<code>get_file_list()</code> (deepmd.nvnmd.utils.fio.Fio method), 231	<code>get_nlist()</code> (deepmd.descriptor.se_a.DescriptorSeA method), 148
	<code>get_nlist()</code> (deepmd.descriptor.se_a_ef.DescriptorSeAEf method), 153
	<code>get_nlist()</code> (deepmd.descriptor.se_r.DescriptorSeR method), 157
	<code>get_nlist()</code> (deepmd.descriptor.se_t.DescriptorSeT method), 160

method), 160

get\_nlist\_i() (deepmd.descriptor.hybrid.DescriptHybrid method), 139

get\_normalize() (in module deepmd.nvnmd.utils.weight), 233

get\_np\_precision() (in module deepmd.common), 271

get\_nsystems() (deepmd.utils.data\_system.DataSystem method), 246

get\_nsystems() (deepmd.utils.data\_system.DeepmdDataSystem method), 248

get\_ntypes() (deepmd.descriptor.descriptor.Descriptor method), 135

get\_ntypes() (deepmd.descriptor.hybrid.DescriptHybrid method), 139

get\_ntypes() (deepmd.descriptor.loc\_frame.DescriptLocFrame method), 142

get\_ntypes() (deepmd.descriptor.se\_a.DescriptSeA method), 148

get\_ntypes() (deepmd.descriptor.se\_a\_ef.DescriptSeAEf method), 153

get\_ntypes() (deepmd.descriptor.se\_r.DescriptSeR method), 158

get\_ntypes() (deepmd.descriptor.se\_t.DescriptSeT method), 160

get\_ntypes() (deepmd.infer.deep\_pot.DeepPot method), 198

get\_ntypes() (deepmd.infer.deep\_tensor.DeepTensor method), 200

get\_ntypes() (deepmd.infer.DeepPot method), 184

get\_ntypes() (deepmd.model.ener.EnerModel method), 211

get\_ntypes() (deepmd.model.tensor.TensorModel method), 214

get\_ntypes() (deepmd.utils.data.DeepmdData method), 245

get\_ntypes() (deepmd.utils.data\_system.DataSystem method), 246

get\_ntypes() (deepmd.utils.data\_system.DeepmdDataSystem method), 248

get\_numb\_aparam() (deepmd.fit.ener.EnerFitting method), 172

get\_numb\_batch() (deepmd.utils.data.DeepmdData method), 245

get\_numb\_fparam() (deepmd.fit.ener.EnerFitting method), 172

get\_numb\_set() (deepmd.utils.data.DataSets method), 243

get\_numb\_set() (deepmd.utils.data.DeepmdData method), 245

get\_nvnmd\_jdata() (deepmd.nvnmd.utils.config.NvnmdConfig method), 227

get\_out\_size() (deepmd.fit.dipole.DipoleFittingSeA method), 169

get\_out\_size() (deepmd.fit.polar.GlobalPolarFittingSeA method), 174

get\_out\_size() (deepmd.fit.polar.PolarFittingLocFrame method), 175

get\_out\_size() (deepmd.fit.polar.PolarFittingSeA method), 176

get\_out\_size() (deepmd.fit.wfc.WFCFitting method), 177

get\_out\_size() (deepmd.model.tensor.TensorModel method), 214

get\_pattern\_nodes\_from\_graph\_def() (in module deepmd.utils.graph), 251

get\_plugin() (deepmd.utils.plugin.Plugin method), 261

get\_precision() (in module deepmd.common), 271

get\_rcut() (deepmd.descriptor.descriptor.Descriptor method), 135

get\_rcut() (deepmd.descriptor.hybrid.DescriptHybrid method), 139

get\_rcut() (deepmd.descriptor.loc\_frame.DescriptLocFrame method), 142

get\_rcut() (deepmd.descriptor.se\_a.DescriptSeA method), 148

get\_rcut() (deepmd.descriptor.se\_a\_ef.DescriptSeAEf method), 153

get\_rcut() (deepmd.descriptor.se\_r.DescriptSeR method), 158

get\_rcut() (deepmd.descriptor.se\_t.DescriptSeT method), 161

get\_rcut() (deepmd.infer.deep\_pot.DeepPot method), 198

get\_rcut() (deepmd.infer.deep\_tensor.DeepTensor method), 200

get\_rcut() (deepmd.infer.DeepPot method), 184

get\_rcut() (deepmd.model.ener.EnerModel method), 211

get\_rcut() (deepmd.model.tensor.TensorModel method), 214

get\_resource() (in module deepmd.cluster), 130

get\_resource() (in module deepmd.cluster.local), 131

get\_resource() (in module deepmd.cluster.slurm), 131

get\_rng\_s() (in module deepmd.nvnmd.utils.weight), 234

get\_rot\_mat() (deepmd.descriptor.loc\_frame.DescriptLocFrame method), 142

get\_rot\_mat() (deepmd.descriptor.se\_a.DescriptSeA method), 148

get\_rot\_mat() (deepmd.descriptor.se\_a\_ef.DescriptSeAEf method), 153

get\_sel\_type() (deepmd.fit.dipole.DipoleFittingSeA method), 169

get\_sel\_type() (deepmd.fit.polar.GlobalPolarFittingSeA method), 174

method), 174

`get_sel_type()` (deepmd.fit.polar.PolarFittingLocFrame method), 175

`get_sel_type()` (deepmd.fit.polar.PolarFittingSeA method), 176

`get_sel_type()` (deepmd.fit.wfc.WFCFitting method), 177

`get_sel_type()` (deepmd.infer.deep\_pot.DeepPot method), 198

`get_sel_type()` (deepmd.infer.deep\_tensor.DeepTensor method), 200

`get_sel_type()` (deepmd.infer.DeepPot method), 184

`get_sel_type()` (deepmd.model.tensor.TensorModel method), 214

`get_sess()` (in module deepmd.nvnmd.utils.network), 232

`get_set()` (deepmd.utils.data.DataSets method), 243

`get_stat()` (deepmd.utils.neighbor\_stat.NeighborStat method), 254

`get_sys()` (deepmd.utils.data\_system.DataSystem method), 246

`get_sys()` (deepmd.utils.data\_system.DeepmdDataSystem method), 248

`get_sys_n_test()` (deepmd.utils.data\_system.DeepmdDataSystem method), 248

`get_sys_numb_batch()` (deepmd.utils.data.DataSets method), 243

`get_sys_numb_batch()` (deepmd.utils.data.DeepmdData method), 245

`get_tensor_by_name()` (in module deepmd.utils.graph), 251

`get_tensor_by_name_from_graph()` (in module deepmd.utils.graph), 251

`get_tensor_by_type()` (in module deepmd.utils.graph), 251

`get_tensor_names()` (deepmd.descriptor.descriptor.Descriptor method), 135

`get_tensor_names()` (deepmd.descriptor.hybrid.Descriptor method), 139

`get_tensor_names()` (deepmd.descriptor.se.Descriptor method), 144

`get_test()` (deepmd.utils.data.DataSets method), 243

`get_test()` (deepmd.utils.data.DeepmdData method), 245

`get_test()` (deepmd.utils.data\_system.DataSystem method), 246

`get_test()` (deepmd.utils.data\_system.DeepmdDataSystem method), 248

`get_training_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 227

`get_type_embedding_net_nodes_from_graph_def()` (in module deepmd.utils.graph), 252

`get_type_embedding_net_variables_from_graph_def()` (in module deepmd.utils.graph), 252

`get_type_map()` (deepmd.infer.deep\_pot.DeepPot method), 198

`get_type_map()` (deepmd.infer.deep\_tensor.DeepTensor method), 200

`get_type_map()` (deepmd.infer.DeepPot method), 184

`get_type_map()` (deepmd.model.ener.EnerModel method), 211

`get_type_map()` (deepmd.model.tensor.TensorModel method), 214

`get_type_map()` (deepmd.utils.data.DataSets method), 243

`get_type_map()` (deepmd.utils.data.DeepmdData method), 245

`get_type_map()` (deepmd.utils.data\_system.DataSystem method), 246

`get_type_map()` (deepmd.utils.data\_system.DeepmdDataSystem method), 248

`get_weight()` (in module deepmd.nvnmd.utils.weight), 234

`get_sys_numb()` (deepmd.fit.wfc.WFCFitting method), 177

`glob()` (deepmd.utils.path.DPH5Path method), 257

`glob()` (deepmd.utils.path.DPOSPath method), 258

`glob()` (deepmd.utils.path.DPPPath method), 259

`global_cvt_2_ener_float()` (in module deepmd.env), 273

`global_cvt_2_tf_float()` (in module deepmd.env), 273

`GLOBAL_ENER_FLOAT_PRECISION` (in module deepmd.env), 273

`GLOBAL_NP_FLOAT_PRECISION` (in module deepmd.env), 273

`StripPolarFittingSeA` (class in deepmd.fit.polar), 173

`StripPolarModel` (class in deepmd.model.tensor), 212

`gpu` (deepmd.train.run\_options.RunOptions attribute), 235

`GraphTooLargeError`, 249

`GraphWithoutTensorError`, 249

## H

`hex2bin()` (deepmd.nvnmd.utils.Encode method), 224

`hex2bin()` (deepmd.nvnmd.utils.encode.Encode method), 230

`hex2bin_str()` (deepmd.nvnmd.utils.Encode method), 224



---

`hex2bin_str()` (deepmd.nvnmd.utils.encode.Encode method), 230  
**I**  
`implemented_properties` (deepmd.calculator.DP attribute), 268  
`import_ops()` (in module deepmd.op), 234  
`init_ctrl()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 227  
`init_dscp()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 228  
`init_fitn()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 228  
`init_from_config()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 228  
`init_from_deepmd_input()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 228  
`init_from_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 228  
`init_nbit()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 228  
`init_net_size()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 228  
`init_size()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 228  
**L**  
`init_train_mode()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 228  
`init_value()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 228  
`init_variables()` (deepmd.descriptor.descriptor.Descriptor method), 135  
`init_variables()` (deepmd.descriptor.hybrid.DescriptorHybrid method), 139  
`init_variables()` (deepmd.descriptor.loc\_frame.DescriptorLocFrame method), 142  
`init_variables()` (deepmd.descriptor.se.DescriptorSe method), 144  
`init_variables()` (deepmd.descriptor.se\_a.DescriptorSeA method), 148  
`init_variables()` (deepmd.fit.dipole.DipoleFittingSeA method), 169  
`init_variables()` (deepmd.fit.ener.EnerFittingSeA method), 172  
`init_variables()` (deepmd.fit.fitting.FittingSeA method), 173  
`init_variables()` (deepmd.fit.polar.GlobalPolarFittingSeA method), 174  
`init_variables()` (deepmd.fit.polar.PolarFittingSeA method), 176  
`init_variables()` (deepmd.model.ener.EnerModel method), 211  
`init_variables()` (deepmd.model.model.Model method), 211  
`init_variables()` (deepmd.model.tensor.TensorModel method), 214  
`init_variables()` (deepmd.utils.type\_embed.TypeEmbedNet method), 264  
`is_chief` (deepmd.train.run\_options.RunOptions property), 235  
`is_dir()` (deepmd.utils.path.DPH5Path method), 257  
`is_dir()` (deepmd.utils.path.DPOSPath method), 258  
`is_dir()` (deepmd.utils.path.DPPath method), 259  
`is_file()` (deepmd.nvnmd.utils.fio.Fio method), 231  
`is_file()` (deepmd.utils.path.DPH5Path method), 257  
`is_file()` (deepmd.utils.path.DPOSPath method), 258  
`is_file()` (deepmd.utils.path.DPPath method), 259  
`is_config()` (deepmd.nvnmd.utils.fio.Fio method), 231  
**J**  
`j_must_have()` (in module deepmd.common), 271  
**L**  
`learning_rate_args()` (in module deepmd.utils.argcheck), 237  
`learning_rate_exp()` (in module deepmd.utils.argcheck), 237  
`learning_rate_variant_type_args()` (in module deepmd.utils.argcheck), 237  
`LearningRateExp` (class in module deepmd.utils.learning\_rate), 252  
`init_from_jdata()` (in module deepmd.utils.argcheck), 237  
`list_to_doc()` (in module deepmd.utils.argcheck), 237  
`load()` (deepmd.nvnmd.utils.fio.FioBin method), 231  
`load()` (deepmd.nvnmd.utils.fio.FioDic method), 231  
`load()` (deepmd.nvnmd.utils.fio.FioJsonDic method), 232  
`load()` (deepmd.nvnmd.utils.fio.FioNpyDic method), 232  
`load()` (deepmd.nvnmd.utils.fio.FioTxt method), 232  
`load()` (deepmd.nvnmd.utils.FioBin method), 224  
`load()` (deepmd.nvnmd.utils.FioDic method), 225  
`load()` (deepmd.nvnmd.utils.FioTxt method), 225  
`load_batch_set()` (deepmd.utils.data.DataSets method), 243  
`load_data()` (deepmd.utils.data.DataSets method), 243

`load_energy()` (deepmd.utils.data.DataSets method), 243  
`load_graph_def()` (in module deepmd.utils.graph), 252  
`load_numpy()` (deepmd.utils.path.DPH5Path method), 258  
`load_numpy()` (deepmd.utils.path.DPOSPath method), 259  
`load_numpy()` (deepmd.utils.path.DPPath method), 260  
`load_prefix` (deepmd.DeepEval attribute), 127  
`load_prefix` (deepmd.DipoleChargeModifier attribute), 130  
`load_prefix` (deepmd.infer.data\_modifier.DipoleChargeModifier attribute), 190  
`load_prefix` (deepmd.infer.deep\_dipole.DeepDipole attribute), 191  
`load_prefix` (deepmd.infer.deep\_eval.DeepEval attribute), 192  
`load_prefix` (deepmd.infer.deep\_polar.DeepGlobalPolar attribute), 194  
`load_prefix` (deepmd.infer.deep\_polar.DeepPolar attribute), 195  
`load_prefix` (deepmd.infer.deep\_pot.DeepPot attribute), 198  
`load_prefix` (deepmd.infer.deep\_tensor.DeepTensor attribute), 200  
`load_prefix` (deepmd.infer.deep\_wfc.DeepWFC attribute), 201  
`load_prefix` (deepmd.infer.DeepDipole attribute), 178  
`load_prefix` (deepmd.infer.DeepEval attribute), 179  
`load_prefix` (deepmd.infer.DeepGlobalPolar attribute), 181  
`load_prefix` (deepmd.infer.DeepPolar attribute), 182  
`load_prefix` (deepmd.infer.DeepPot attribute), 185  
`load_prefix` (deepmd.infer.DeepWFC attribute), 186  
`load_prefix` (deepmd.infer.DipoleChargeModifier attribute), 187  
`load_set()` (deepmd.utils.data.DataSets method), 243  
`load_test_set()` (deepmd.utils.data.DataSets method), 243  
`load_txt()` (deepmd.utils.path.DPH5Path method), 258  
`load_txt()` (deepmd.utils.path.DPOSPath method), 259  
`load_txt()` (deepmd.utils.path.DPPath method), 260  
`load_type()` (deepmd.utils.data.DataSets method), 243  
`load_type_map()` (deepmd.utils.data.DataSets method), 243  
`Loss` (class in deepmd.loss.loss), 207  
`loss_args()` (in module deepmd.utils.argcheck), 237  
`loss_ener()` (in module deepmd.utils.argcheck), 237  
`loss_tensor()` (in module deepmd.utils.argcheck), 237  
`loss_variant_type_args()` (in module deepmd.utils.argcheck), 237  
**M**  
`main()` (in module deepmd.entrypoints.main), 166  
`main_parser()` (in module deepmd.entrypoints.main), 166  
`make_default_mesh()` (in module deepmd.common), 272  
`make_index()` (in module deepmd.utils.argcheck), 237  
`make_link()` (in module deepmd.utils.argcheck), 237  
`make_model_devi()` (in module deepmd.entrypoints), 162  
`make_model_devi()` (in module deepmd.infer.model\_devi), 203  
`make_natoms_vec()` (deepmd.DeepEval method), 127  
`make_natoms_vec()` (deepmd.infer.deep\_eval.DeepEval method), 192  
`make_natoms_vec()` (deepmd.infer.DeepEval method), 179  
`make_stat_input()` (in module deepmd.model.model\_stat), 212  
`map_nvnmd()` (in module deepmd.nvnmd.utils), 226  
`map_nvnmd()` (in module deepmd.nvnmd.utils.op), 233  
`mapt()` (in module deepmd.nvnmd.entrypoints.mapt), 220  
`MapTable` (class in deepmd.nvnmd.entrypoints), 216  
`MapTable` (class in deepmd.nvnmd.entrypoints.mapt), 219  
`matmul2_qq()` (in module deepmd.nvnmd.utils.network), 232  
`matmul3_qq()` (in module deepmd.nvnmd.utils.network), 232  
`merge_bin()` (deepmd.nvnmd.utils.Encode method), 224  
`merge_bin()` (deepmd.nvnmd.utils.encode.Encode method), 230  
`merge_sys_stat()` (in module deepmd.model.model\_stat), 212  
`mixed_precision_args()` (in module deepmd.utils.argcheck), 237  
`mkdir()` (deepmd.nvnmd.utils.fio.Fio method), 231  
`Model` (class in deepmd.model.model), 211  
`model_args()` (in module deepmd.utils.argcheck), 237

`model_compression()` (in module `deepmd.utils.argcheck`), 237  
`model_compression_type_args()` (in module `deepmd.utils.argcheck`), 238  
`model_type` (`deepmd.DeepEval` property), 128  
`model_type` (`deepmd.infer.deep_eval.DeepEval` property), 192  
`model_type` (`deepmd.infer.DeepEval` property), 179  
`model_type` (`deepmd.model.ener.EnerModel` attribute), 211  
`model_version` (`deepmd.DeepEval` property), 128  
`model_version` (`deepmd.infer.deep_eval.DeepEval` property), 192  
`model_version` (`deepmd.infer.DeepEval` property), 179  
`modifier_dipole_charge()` (in module `deepmd.utils.argcheck`), 238  
`modifier_variant_type_args()` (in module `deepmd.utils.argcheck`), 238  
`modify_data()` (`deepmd.DipoleChargeModifier` method), 130  
`modify_data()` (`deepmd.infer.data_modifier.DipoleChargeModifier` method), 190  
`modify_data()` (`deepmd.infer.DipoleChargeModifier` method), 187  
`module`  
`deepmd`, 127  
`deepmd.calculator`, 265  
`deepmd.cluster`, 130  
`deepmd.cluster.local`, 131  
`deepmd.cluster.slurm`, 131  
`deepmd.common`, 268  
`deepmd.descriptor`, 131  
`deepmd.descriptor.descriptor`, 131  
`deepmd.descriptor.hybrid`, 137  
`deepmd.descriptor.loc_frame`, 140  
`deepmd.descriptor.se`, 143  
`deepmd.descriptor.se_a`, 145  
`deepmd.descriptor.se_a_ebd`, 149  
`deepmd.descriptor.se_a_ef`, 151  
`deepmd.descriptor.se_r`, 155  
`deepmd.descriptor.se_t`, 158  
`deepmd.entripoints`, 161  
`deepmd.entripoints.compress`, 164  
`deepmd.entripoints.config`, 165  
`deepmd.entripoints.convert`, 165  
`deepmd.entripoints.doc`, 165  
`deepmd.entripoints.freeze`, 165  
`deepmd.entripoints.main`, 166  
`deepmd.entripoints.neighbor_stat`, 166  
`deepmd.entripoints.test`, 167  
`deepmd.entripoints.train`, 167  
`deepmd.entripoints.transfer`, 168  
`deepmd.env`, 273  
`deepmd.fit`, 168  
`deepmd.fit.dipole`, 168  
`deepmd.fit.ener`, 170  
`deepmd.fit.fitting`, 172  
`deepmd.fit.polar`, 173  
`deepmd.fit.wfc`, 177  
`deepmd.infer`, 177  
`deepmd.infer.data_modifier`, 189  
`deepmd.infer.deep_dipole`, 191  
`deepmd.infer.deep_eval`, 192  
`deepmd.infer.deep_polar`, 193  
`deepmd.infer.deep_pot`, 195  
`deepmd.infer.deep_tensor`, 198  
`deepmd.infer.deep_wfc`, 200  
`deepmd.infer.ewald_recip`, 201  
`deepmd.infer.model_devi`, 202  
`deepmd.loggers`, 204  
`deepmd.loggers.loggers`, 204  
`deepmd.loss`, 205  
`deepmd.loss.ener`, 205  
`deepmd.loss.loss`, 207  
`deepmd.loss.tensor`, 209  
`deepmd.model`, 210  
`deepmd.model.ener`, 210  
`deepmd.model.model`, 211  
`deepmd.model.model_stat`, 212  
`deepmd.model.tensor`, 212  
`deepmd.nvnmd`, 215  
`deepmd.nvnmd.data`, 215  
`deepmd.nvnmd.data.data`, 216  
`deepmd.nvnmd.descriptor`, 216  
`deepmd.nvnmd.descriptor.se_a`, 216  
`deepmd.nvnmd.entripoints`, 216  
`deepmd.nvnmd.entripoints.freeze`, 219  
`deepmd.nvnmd.entripoints.mapt`, 219  
`deepmd.nvnmd.entripoints.train`, 221  
`deepmd.nvnmd.entripoints.wrap`, 221  
`deepmd.nvnmd.fit`, 222  
`deepmd.nvnmd.fit.ener`, 222  
`deepmd.nvnmd.utils`, 222  
`deepmd.nvnmd.utils.argcheck`, 226  
`deepmd.nvnmd.utils.config`, 226  
`deepmd.nvnmd.utils.encode`, 228  
`deepmd.nvnmd.utils.fio`, 230  
`deepmd.nvnmd.utils.network`, 232  
`deepmd.nvnmd.utils.op`, 233  
`deepmd.nvnmd.utils.weight`, 233  
`deepmd.op`, 234  
`deepmd.train`, 234  
`deepmd.train.run_options`, 234  
`deepmd.train.trainer`, 235  
`deepmd.utils`, 236  
`deepmd.utils.argcheck`, 236  
`deepmd.utils.batch_size`, 238



deepmd.utils.compat, 239  
 deepmd.utils.convert, 240  
 deepmd.utils.data, 242  
 deepmd.utils.data\_system, 246  
 deepmd.utils.errors, 249  
 deepmd.utils.graph, 249  
 deepmd.utils.learning\_rate, 252  
 deepmd.utils.neighbor\_stat, 253  
 deepmd.utils.network, 254  
 deepmd.utils.pair\_tab, 255  
 deepmd.utils.parallel\_op, 256  
 deepmd.utils.path, 257  
 deepmd.utils.plugin, 260  
 deepmd.utils.random, 262  
 deepmd.utils.sess, 262  
 deepmd.utils.tabulate, 263  
 deepmd.utils.type\_embed, 264  
 deepmd.utils.weight\_avg, 265  
 my\_device (deepmd.train.run\_options.RunOptions attribute), 235  
 my\_rank (deepmd.train.run\_options.RunOptions attribute), 235

## N

name (deepmd.calculator.DP attribute), 268  
 neighbor\_stat() (in module deepmd.entrypoints), 162  
 neighbor\_stat() (in module deepmd.entrypoints.neighbor\_stat), 166  
 NeighborStat (class in deepmd.utils.neighbor\_stat), 253  
 nodelist (deepmd.train.run\_options.RunOptions attribute), 235  
 nodename (deepmd.train.run\_options.RunOptions attribute), 235  
 normalize() (in module deepmd.utils.argcheck), 238  
 normalize\_hybrid\_list() (in module deepmd.utils.argcheck), 238  
 normalized\_input() (in module deepmd.nvnmd.entrypoints.train), 221  
 normalized\_input\_qnn() (in module deepmd.nvnmd.entrypoints.train), 221  
 numb\_aparam() (deepmd.utils.data.DataSets method), 243  
 numb\_fparam() (deepmd.utils.data.DataSets method), 243  
 numb\_fparam() (deepmd.utils.data\_system.DataSystem method), 246  
 nvnmd\_args() (in module deepmd.nvnmd.utils), 226  
 nvnmd\_args() (in module deepmd.nvnmd.utils.argcheck), 226  
 NvnmdConfig (class in deepmd.nvnmd.utils.config), 226

## O

one\_layer() (in module deepmd.nvnmd.utils), 226  
 one\_layer() (in module deepmd.nvnmd.utils.network), 233  
 one\_layer() (in module deepmd.utils.network), 255  
 one\_layer\_rand\_seed\_shift() (in module deepmd.utils.network), 255  
 one\_layer\_wb() (in module deepmd.nvnmd.utils.network), 233  
 OutOfMemoryError, 249

## P

PairTab (class in deepmd.utils.pair\_tab), 255  
 ParallelOp (class in deepmd.utils.parallel\_op), 256  
 parse() (deepmd.common.ClassArg method), 269  
 parse\_args() (in module deepmd.entrypoints.main), 166  
 pass\_tensors\_from\_frz\_model() (deepmd.descriptor.descriptor.Descriptor method), 136  
 pass\_tensors\_from\_frz\_model() (deepmd.descriptor.hybrid.DescriptHybrid method), 139  
 pass\_tensors\_from\_frz\_model() (deepmd.descriptor.se.DescriptSe method), 144  
 Plugin (class in deepmd.utils.plugin), 260  
 PluginVariant (class in deepmd.utils.plugin), 261  
 PolarFittingLocFrame (class in deepmd.fit.polar), 174  
 PolarFittingSeA (class in deepmd.fit.polar), 175  
 PolarModel (class in deepmd.model.tensor), 213  
 precision (deepmd.descriptor.se.DescriptSe property), 145  
 precision (deepmd.fit.fitting.Fitting property), 173  
 print\_header() (deepmd.loss.ener.EnerDipoleLoss static method), 206  
 print\_header() (deepmd.loss.ener.EnerStdLoss method), 207  
 print\_header() (deepmd.loss.tensor.TensorLoss method), 209  
 print\_header() (deepmd.train.trainer.DPTrainer static method), 235  
 print\_on\_training() (deepmd.loss.ener.EnerDipoleLoss method), 206  
 print\_on\_training() (deepmd.loss.ener.EnerStdLoss method), 207  
 print\_on\_training() (deepmd.loss.tensor.TensorLoss method), 209  
 print\_on\_training() (deepmd.train.trainer.DPTrainer static method), 235

method), 236  
 print\_resource\_summary()  
     (deepmd.train.run\_options.RunOptions  
     method), 235  
 print\_summary() (deepmd.utils.data\_system.DataSystem  
     method), 246  
 print\_summary() (deepmd.utils.data\_system.DeepmdDataSystem  
     method), 248  
 process\_sys\_weights()  
     (deepmd.utils.data\_system.DataSystem  
     method), 247  
 prod\_force\_virial()  
     (deepmd.descriptor.descriptor.Descriptor  
     method), 136  
 prod\_force\_virial()  
     (deepmd.descriptor.hybrid.DescriptHybrid  
     method), 139  
 prod\_force\_virial()  
     (deepmd.descriptor.loc\_frame.DescriptLocFrame  
     method), 142  
 prod\_force\_virial()  
     (deepmd.descriptor.se\_a.DescriptSeA  
     method), 148  
 prod\_force\_virial()  
     (deepmd.descriptor.se\_a\_ef.DescriptSeAEf  
     method), 153  
 prod\_force\_virial()  
     (deepmd.descriptor.se\_r.DescriptSeR  
     method), 158  
 prod\_force\_virial()  
     (deepmd.descriptor.se\_t.DescriptSeT  
     method), 161

## Q

qc() (deepmd.nvnmd.utils.Encode method), 224  
 qc() (deepmd.nvnmd.utils.encode.Encode method),  
     230  
 qf() (deepmd.nvnmd.utils.Encode method), 224  
 qf() (deepmd.nvnmd.utils.encode.Encode method),  
     230  
 qf() (in module deepmd.nvnmd.utils.network), 233  
 qqq() (deepmd.nvnmd.entrypoints.maptable.MapTable  
     method), 220  
 qqq() (deepmd.nvnmd.entrypoints.MapTable  
     method), 217  
 qr() (deepmd.nvnmd.utils.Encode method), 224  
 qr() (deepmd.nvnmd.utils.encode.Encode method),  
     230  
 qr() (in module deepmd.nvnmd.utils.network), 233

## R

random() (in module deepmd.utils.random), 262  
 reduce() (deepmd.utils.data.DeepmdData method),  
     245  
 reduce() (deepmd.utils.data\_system.DeepmdDataSystem  
     method), 248  
 register() (deepmd.descriptor.descriptor.Descriptor  
     static method), 136  
 register() (deepmd.utils.argcheck.ArgsPlugin  
     method), 236  
 register() (deepmd.utils.plugin.Plugin method),  
     261  
 reinit() (deepmd.utils.pair\_tab.PairTab method),  
     256  
 remove\_decay\_rate() (in module  
     deepmd.utils.compat), 240  
 replace\_path() (in module  
     deepmd.nvnmd.entrypoints.train), 221  
 reset\_default\_tf\_session\_config() (in module  
     deepmd.env), 273  
 reset\_get\_batch() (deepmd.utils.data.DeepmdData  
     method), 245  
 reset\_iter() (deepmd.utils.data.DataSets method),  
     243  
 reverse\_bin() (deepmd.nvnmd.utils.Encode  
     method), 224  
 reverse\_bin() (deepmd.nvnmd.utils.encode.Encode  
     method), 230  
 reverse\_map() (deepmd.DeepEval static method),  
     128  
 reverse\_map() (deepmd.infer.deep\_eval.DeepEval  
     static method), 192  
 reverse\_map() (deepmd.infer.DeepEval static  
     method), 179  
 rglob() (deepmd.utils.path.DPH5Path method), 258  
 rglob() (deepmd.utils.path.DPOSPath method), 259  
 rglob() (deepmd.utils.path.DPPPath method), 260  
 run\_s2G() (deepmd.nvnmd.entrypoints.maptable.MapTable  
     method), 220  
 run\_s2G() (deepmd.nvnmd.entrypoints.MapTable  
     method), 218  
 run\_sess() (in module deepmd.utils.sess), 262  
 run\_u2s() (deepmd.nvnmd.entrypoints.maptable.MapTable  
     method), 220  
 run\_u2s() (deepmd.nvnmd.entrypoints.MapTable  
     method), 218  
 RunOptions (class in deepmd.train.run\_options), 234

## S

safe\_cast\_tensor() (in module deepmd.common),  
     272  
 save() (deepmd.nvnmd.utils.config.NvnmdConfig  
     method), 228  
 save() (deepmd.nvnmd.utils.fio.FioBin method), 231  
 save() (deepmd.nvnmd.utils.fio.FioDic method), 231  
 save() (deepmd.nvnmd.utils.fio.FioJsonDic  
     method), 232

`save()` (deepmd.nvnmd.utils.fio.FioNpyDic method), 232  
`save()` (deepmd.nvnmd.utils.fio.FioTxt method), 232  
`save()` (deepmd.nvnmd.utils.FioBin method), 224  
`save()` (deepmd.nvnmd.utils.FioDic method), 225  
`save()` (deepmd.nvnmd.utils.FioTxt method), 225  
`save_checkpoint()` (deepmd.train.trainer.DPTrainer method), 236  
`save_compressed()` (deepmd.train.trainer.DPTrainer method), 236  
`save_weight()` (in module deepmd.nvnmd.entrypoints), 219  
`save_weight()` (in module deepmd.nvnmd.entrypoints.freeze), 219  
`seed()` (in module deepmd.utils.random), 262  
`select_idx_map()` (in module deepmd.common), 272  
`set_log_handles()` (in module deepmd.loggers), 204  
`set_log_handles()` (in module deepmd.loggers.loggers), 204  
`set_numb_batch()` (deepmd.utils.data.DataSets method), 243  
`set_sys_probs()` (deepmd.utils.data\_system.DeepmdDataSystem method), 249  
`shuffle()` (in module deepmd.utils.random), 262  
`sort_input()` (deepmd.DeepEval static method), 128  
`sort_input()` (deepmd.infer.deep\_eval.DeepEval static method), 193  
`sort_input()` (deepmd.infer.DeepEval static method), 179  
`split_bin()` (deepmd.nvnmd.utils.Encode method), 224  
`split_bin()` (deepmd.nvnmd.utils.encode.Encode method), 230  
`start_lr()` (deepmd.utils.learning\_rate.LearningRateExp method), 253  
`start_pref()` (in module deepmd.utils.argcheck), 238  
`stats_energy()` (deepmd.utils.data.DataSets method), 243

## T

`tanh2()` (in module deepmd.nvnmd.utils.network), 233  
`tanh4()` (in module deepmd.nvnmd.utils.network), 233  
`TensorLoss` (class in deepmd.loss.tensor), 209  
`TensorModel` (class in deepmd.model.tensor), 213  
`tensors` (deepmd.infer.deep\_tensor.DeepTensor attribute), 200  
`test()` (in module deepmd.entrypoints), 163  
`test()` (in module deepmd.entrypoints.test), 167

`train()` (deepmd.train.trainer.DPTrainer method), 236  
`train()` (in module deepmd.entrypoints.train), 167  
`train_dp()` (in module deepmd.entrypoints), 163  
`train_nvnmd()` (in module deepmd.nvnmd.entrypoints.train), 221  
`training_args()` (in module deepmd.utils.argcheck), 238  
`training_data_args()` (in module deepmd.utils.argcheck), 238  
`transfer()` (in module deepmd.entrypoints), 163  
`transfer()` (in module deepmd.entrypoints.transfer), 168  
`type_embedding_args()` (in module deepmd.utils.argcheck), 238  
`TypeEmbedNet` (class in deepmd.utils.type\_embed), 264

## U

`update()` (deepmd.nvnmd.utils.fio.FioDic method), 231  
`update()` (deepmd.nvnmd.utils.FioDic method), 225  
`update_deepmd_input()` (in module deepmd.utils.compat), 240

## V

`valid_on_the_fly()` (deepmd.train.trainer.DPTrainer method), 236  
`validation_data_args()` (in module deepmd.utils.argcheck), 238  
`value()` (deepmd.utils.learning\_rate.LearningRateExp method), 253  
`variable_summaries()` (in module deepmd.utils.network), 255  
`VariantABCMeta` (class in deepmd.utils.plugin), 261  
`VariantMeta` (class in deepmd.utils.plugin), 261

## W

`weighted_average()` (in module deepmd.utils.weight\_avg), 265  
`WFCFitting` (class in deepmd.fit.wfc), 177  
`WFCModel` (class in deepmd.model.tensor), 214  
`world_size` (deepmd.train.run\_options.RunOptions attribute), 235  
`Wrap` (class in deepmd.nvnmd.entrypoints), 218  
`Wrap` (class in deepmd.nvnmd.entrypoints.wrap), 221  
`wrap()` (deepmd.nvnmd.entrypoints.Wrap method), 218  
`wrap()` (deepmd.nvnmd.entrypoints.wrap.Wrap method), 222  
`wrap()` (in module deepmd.nvnmd.entrypoints.wrap), 222  
`wrap_bias()` (deepmd.nvnmd.entrypoints.Wrap method), 218

`wrap_bias()` (deepmd.nvnmd.entrypoints.wrap.Wrap  
method), [222](#)  
`wrap_dscp()` (deepmd.nvnmd.entrypoints.Wrap  
method), [218](#)  
`wrap_dscp()` (deepmd.nvnmd.entrypoints.wrap.Wrap  
method), [222](#)  
`wrap_fitn()` (deepmd.nvnmd.entrypoints.Wrap  
method), [218](#)  
`wrap_fitn()` (deepmd.nvnmd.entrypoints.wrap.Wrap  
method), [222](#)  
`wrap_head()` (deepmd.nvnmd.entrypoints.Wrap  
method), [218](#)  
`wrap_head()` (deepmd.nvnmd.entrypoints.wrap.Wrap  
method), [222](#)  
`wrap_map()` (deepmd.nvnmd.entrypoints.Wrap  
method), [218](#)  
`wrap_map()` (deepmd.nvnmd.entrypoints.wrap.Wrap  
method), [222](#)  
`wrap_weight()` (deepmd.nvnmd.entrypoints.Wrap  
method), [219](#)  
`wrap_weight()` (deepmd.nvnmd.entrypoints.wrap.Wrap  
method), [222](#)  
`write_model_devi_out()` (in module  
deepmd.infer.model\_devi), [203](#)