
DeePMD-kit

DeepModeling

Sep 23, 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
1.4	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	10
2.1.3	Install with docker	10
2.2	Install from source code	10
2.2.1	Install the python interface	11
2.2.2	Install the C++ interface	14
2.3	Install LAMMPS	16
2.3.1	Install LAMMPS's DeePMD-kit module (built-in mode)	16
2.3.2	Install LAMMPS (plugin mode)	17
2.4	Install i-PI	17
2.5	Install GROMACS with DeepMD	18
2.5.1	Patch source code of GROMACS	18
2.5.2	Compile GROMACS with deepmd-kit	18
2.6	Building conda packages	18
3	Data	21
3.1	System	21
3.2	Formats of a system	23
3.2.1	NumPy format	23
3.2.2	HDF5 format	23
3.2.3	Raw format and data conversion	24
3.3	Prepare data with dpdata	24
4	Model	27
4.1	Overall	27
4.2	Descriptor "se_e2_a"	28
4.3	Descriptor "se_e2_r"	29

4.4	Descriptor "se_e3"	29
4.5	Descriptor "se_atten"	30
4.5.1	DPA-1: Pretraining of Attention-based Deep Potential Model for Molecular Simulation	30
4.5.2	Installation	30
4.5.3	Introduction to new features of DPA-1	30
4.5.4	Data format	32
4.5.5	Training example	33
4.6	Descriptor "hybrid"	33
4.7	Determine sel	34
4.8	Fit energy	34
4.8.1	The fitting network	34
4.8.2	Loss	34
4.9	Fit tensor like Dipole and Polarizability	35
4.9.1	The fitting Network	35
4.9.2	Loss	36
4.9.3	Training Data Preparation	36
4.9.4	Train the Model	37
4.10	Type embedding approach	37
4.10.1	Type embedding net	38
4.11	Deep potential long-range (DPLR)	39
4.11.1	Train a deep Wannier model for Wannier centroids	39
4.11.2	Train the DPLR model	40
4.11.3	Molecular dynamics simulation with DPLR	40
4.12	Deep Potential - Range Correction (DPRc)	42
4.12.1	Training data	43
4.12.2	Training the DPRc model	43
4.12.3	Run MD simulations	44
5	Training	45
5.1	Train a model	45
5.2	Advanced options	46
5.2.1	Learning rate	46
5.2.2	Training parameters	47
5.2.3	Options and environment variables	49
5.2.4	Adjust sel of a frozen model	50
5.3	Training Parameters	50
5.4	Parallel training	77
5.4.1	Tuning learning rate	77
5.4.2	Scaling test	78
5.4.3	How to use	78
5.4.4	Logging	79
5.5	TensorBoard Usage	79
5.5.1	Highlighted features	79
5.5.2	How to use Tensorboard with DeePMD-kit	79
5.5.3	Examples	80
5.5.4	Attention	85
5.6	Known limitations of using GPUs	85
6	Freeze and Compress	87
6.1	Freeze a model	87
6.2	Compress a model	87
7	Test	91
7.1	Test a model	91

7.2	Calculate Model Deviation	92
8	Inference	93
8.1	Python interface	93
8.2	C++ interface	93
9	Command line interface	95
9.1	Named Arguments	95
9.2	Valid subcommands	95
9.3	Sub-commands:	95
9.3.1	config	95
9.3.2	transfer	96
9.3.3	train	96
9.3.4	freeze	97
9.3.5	test	98
9.3.6	compress	99
9.3.7	doc-train-input	100
9.3.8	model-devi	100
9.3.9	convert-from	101
9.3.10	neighbor-stat	102
9.3.11	train-nvnmd	102
10	Integrate with third-party packages	105
10.1	Use deep potential with ASE	105
10.2	Run MD with LAMMPS	105
10.3	LAMMPS commands	106
10.3.1	Enable DeePMD-kit plugin (plugin mode)	106
10.3.2	pair_style deepmd	106
10.3.3	Compute tensorial properties	107
10.3.4	Long-range interaction	107
10.3.5	Use of the centroid/stress/atom to get the full 3x3 “atomic-virial”	108
10.3.6	Computation of heat flux	108
10.4	Run path-integral MD with i-PI	109
10.5	Running MD with GROMACS	109
10.5.1	DP/MM Simulation	109
10.5.2	All-atom DP Simulation	112
10.6	Interfaces out of DeePMD-kit	113
10.6.1	dpdata	113
10.6.2	OpenMM plugin for DeePMD-kit	113
10.6.3	AMBER interface to DeePMD-kit	113
10.6.4	DP-GEN	113
10.6.5	MAtom	113
11	Use NVNMD	115
11.1	Introduction	115
11.2	Training	115
11.2.1	Input script	116
11.2.2	Training	118
11.3	Testing	118
11.4	Running MD	118
11.4.1	Account application	119
11.4.2	Adding task	119
11.4.3	Cancelling calculation	121
11.4.4	Downloading results	121
11.4.5	Deleting record	122

11.4.6	Clearing records	122
12	FAQs	123
12.1	How to tune Fitting/embedding-net size ?	123
12.1.1	Al ₂ O ₃	123
12.1.2	Cu	124
12.1.3	Water	125
12.1.4	Mg-Al	126
12.2	How to control the number of nodes used by a job ?	126
12.3	Do we need to set rcut < half boxsize ?	127
12.4	How to set sel ?	127
12.5	Installation	127
12.5.1	Inadequate versions of gcc/g++	127
12.5.2	Build files left in DeePMD-kit	127
12.6	The temperature undulates violently during early stages of MD	128
12.7	MD: cannot run LAMMPS after installing a new version of DeePMD-kit	128
12.8	Model compatibility	128
13	Coding Conventions	129
13.1	Preface	129
13.2	Rules	129
13.3	Whitespace	130
13.4	General advice	130
13.5	Writing documentation in the code	130
13.6	Run pycodestyle on your code	131
13.7	Run mypy on your code	131
13.8	Run pydocstyle on your code	131
13.9	Run black on your code	131
14	Create a model	133
14.1	Design a new component	133
14.2	Register new arguments	133
14.3	Package new codes	134
15	Atom Type Embedding	135
15.1	Overview	135
15.2	Preliminary	135
15.3	How to use	136
15.4	Code Modification	136
15.4.1	trainer (train/trainer.py)	136
15.4.2	model (model/ener.py)	136
15.4.3	embedding net (descriptor/se*.py)	136
15.4.4	fitting net (fit/ener.py)	137
16	Python API	139
16.1	deepmd package	139
16.1.1	Subpackages	143
16.1.2	Submodules	282
16.1.3	deepmd.calculator module	282
16.1.4	deepmd.common module	284
16.1.5	deepmd.env module	289
17	OP API	291
17.1	op_module	291
17.2	op_grads_module	332

18	C++ API	339
18.1	Class Hierarchy	339
18.2	File Hierarchy	339
18.3	Full API	340
18.3.1	Namespaces	340
18.3.2	Classes and Structs	341
18.3.3	Functions	355
18.3.4	Typedefs	360
19	Core API	361
19.1	Class Hierarchy	361
19.2	File Hierarchy	362
19.3	Full API	363
19.3.1	Namespaces	363
19.3.2	Classes and Structs	367
19.3.3	Functions	376
19.3.4	Variables	412
19.3.5	Defines	413
19.3.6	Typedefs	415
20	License	417
21	Authors and Credits	419
21.1	Package Contributors	419
21.2	Other Credits	420
	Bibliography	421
	Python Module Index	423
	Index	425

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: 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
```

One may enable the environment using

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

1.1.2 Install with conda

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

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

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

Or one may want to create a GPU environment containing [CUDA Toolkit](#):

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

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

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

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

One may enable the environment using

```
conda activate deepmd
```

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.

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: 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
```

One may enable the environment using

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

2.1.2 Install with conda

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

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

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

Or one may want to create a GPU environment containing [CUDA Toolkit](#):

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

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

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

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

One may enable the environment using

```
conda activate deepmd
```

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.git 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:

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
- DTENSORFLOW_ROOT=	Path <value>	-	The Path to TensorFlow's C++ interface.
- DCMAKE_INSTALL_PREFIX=	Path <value>	-	The Path where DeePMD-kit will be installed.
- DUSE_CUDA_TOOLKIT=	TRUE FALSE	FALSE <value>	If TRUE, Build GPU support with CUDA toolkit.
- DCUDA_TOOLKIT_ROOT=	Path <value>	De- tected auto- mati- cally	The path to the CUDA toolkit directory. CUDA 7.0 or later is supported. NVCC is required.
- DUSE_ROCM_TOOLKIT=	TRUE FALSE	FALSE <value>	If TRUE, Build GPU support with ROCM toolkit.
- DROCM_ROOT=	Path <value>	De- tected auto- mati- cally	The path to the ROCM toolkit directory.
- DLAMMPS_VERSION_OR_NUMBER=	Num- ber <value>	20220723 <value>	Only necessary for LAMMPS built-in mode. The version number of LAMMPS (yyyymmdd). LAMMPS 29Oct2020 (20201029) or later is supported.
- DLAMMPS_SOURCE_ROOT=	Path <value>	-	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.
- DUSE_TF_PYTHON_LIBS=	TRUE FALSE	FALSE <value>	If TRUE, Build C++ interface with TensorFlow's Python libraries (TensorFlow's Python Interface is required). And there's no need for building TensorFlow's C++ interface.

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 lammps
```

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_update1.tar.gz
tar xf stable_23Jun2022_update1.tar.gz
```

The source code of LAMMPS is stored in directory `lammps-stable_23Jun2022_update1`. Now go into the LAMMPS code and copy the DeePMD-kit module like this

```
cd lammps-stable_23Jun2022_update1/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_update1.tar.gz
tar xf stable_23Jun2022_update1.tar.gz
```

The source code of LAMMPS is stored in directory `lammps-stable_23Jun2022_update1`. 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_update1/build/
cd lammps-stable_23Jun2022_update1/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 H₂O). To contains data with different formula, one usually need to divide data into multiple systems, which may sometimes result in sparse-frame systems. See a [new system format](#) to further combine different systems with the same atom numbers, when training with descriptor **se_atten**.

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

ID	Property	Raw file	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.opt	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 * 9	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".

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

3.2.2 HDF5 format

A system with the HDF5 format has the same structure as the Numpy format, but in 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 need 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 files 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
```

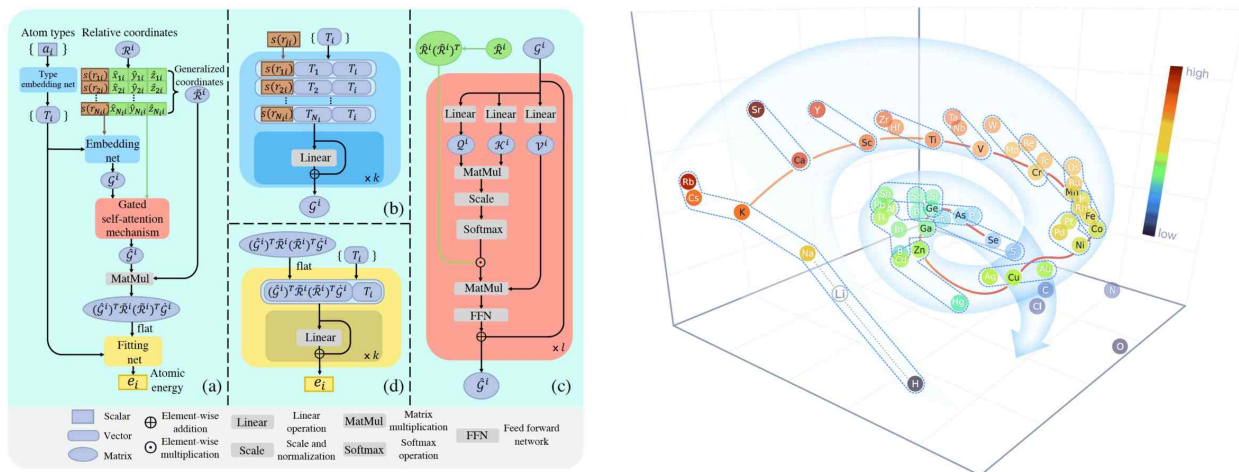
The training input script is very similar to that of `se_e2_a`. The only difference lies in the `descriptor` <model/descriptor> section

```
"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 "se_atten"

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



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

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

4.5.2 Installation

Follow the [standard installation](#) of python interface in DeePMD-kit. After that, you can smoothly use the DPA-1 model with following instructions.

4.5.3 Introduction to new features of DPA-1

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

Descriptor "se_atten"

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

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

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

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

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

```

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

```

- The `type` of the descriptor is set to `"se_atten"`, which will use DPA-1 structures.
- `rcut` is the cut-off radius for neighbor searching, and the `rcut_smth` gives where the smoothing starts.
- `sel` gives the maximum possible number of neighbors in the cut-off radius. It is an int. Note that this number highly effects the efficiency of training, which we usually use less than 200. (We use 120 for training 56 elements in [OC2M dataset](#))
- The `neuron` specifies the size of the embedding net. From left to right the members denote the sizes of each hidden layer from 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.
- 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.
- `attn` sets the length of hidden vector during scale-dot attention computation.
- `attn_layer` sets the number of layers in attention mechanism.
- `attn_mask` determines whether to mask the diagonal in the attention weights and False is recommended.
- `attn_dottr` determines whether to dot the relative coordinates on the attention weights as a gated scheme, True is recommended.

Fitting "ener"

DPA-1 only support "ener" fitting type, and you can refer [here](#) for detail information.

Type embedding

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

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

You can add these settings in input.json if you want to change the default ones, see [here](#) for detail information.

Type map

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

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

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

4.5.4 Data format

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

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

```
type.raw
type_map.raw
set.000/box.npy
set.000/coord.npy
set.000/energy.npy
set.000/force.npy
set.000/real_atom_types.npy
```

This system contains `Nframes` frames with the same atom number `Natoms`, the total number of element types contained in all frames is `Ntypes`. Note that we put all the frames in one set `set.000`. Most files are the same as those in [standard format](#), here we only list the distinct ones:

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

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

The API to generate or transfer to **mixed_type** format will be uploaded on [dpdata](#) soon for a more convenient experience.

4.5.5 Training example

Here we upload the AlMgCu example showed in the paper, you can download here: [Baidu disk](#); [Google disk](#).

4.6 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 is 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
```

4.7 Determine sel

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

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

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

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

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

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

4.8 Fit energy

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

4.8.1 The fitting network

The construction of the fitting net is 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.

4.8.2 Loss

The loss function L for training energy is given by

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

where L_e , L_f , and L_v denote the loss in energy, 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^∞ 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.

$$\text{pref_f}(t) = \text{start_pref_f} * (\text{lr}(t) / \text{start_lr}) + \text{limit_pref_f} * (1 - \text{lr}(t) / \text{start_lr})$$

The `loss` section in the `input.json` is

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

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

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

4.9 Fit 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`.

4.9.1 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.

4.9.2 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 x 9` vector which gives the elements of the polarizability tensor of that frame in order `XX, XY, XZ, YX, YY, YZ, XZ, ZY, ZZ`. By contrast, in a atomic system, each atom in `sel_type` has a `tensor` label. For example, when fitting `dipole`, each frame will provide a `#sel_atom x 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.

4.9.3 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.

4.9.4 Train the Model

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

```
dp train input.json
```

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

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

One may notice that in each step, some of 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.

4.10 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.

4.10.1 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" : {
    ...
  }
}
```

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.

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

4.11 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.

4.11.1 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
```

4.11.2 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 \AA^{-1}) gives the spread parameter controls the spread of Gaussian charges, and `ewald_h` (unit \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
```

4.11.3 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
```

(continues on next page)

(continued from previous page)

```

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    pppm/dplr 1e-5
kspace_modify   gewald ${BETA} diff ik mesh ${KMESH} ${KMESH} ${KMESH}
```

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

should be set dense enough so the long-range calculation is converged.

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

The fix command dplr calculates the position of WCs by the DW model and back-propagates the long-range interaction on virtual atoms to real 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.

4.12 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.

4.12.1 Training data

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

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

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

4.12.2 Training the DPRc model

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

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

As described in the paper, the DPRc model only corrects E_{QM} and $E_{\text{QM/MM}}$ 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"
    },
    {
      "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)))))
```

(continues on next page)

(continued from previous page)

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`.

4.12.3 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.

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

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

5.2.2 Training parameters

Other training parameters are given in the [training](#) section.

```
{
  "training": {
    "training_data": {
      "systems": ["../data_water/data_0/", "../data_water/data_1/", "../data_
↩water/data_2/"],
      "batch_size": "auto"
    },
    "validation_data": {
      "systems": ["../data_water/data_3/"],
      "batch_size": 1,
      "numb_btch": 3
    },
    "mixed_precision": {
      "output_prec": "float32",
      "compute_prec": "float16"
    },
    "numb_steps": 1000000,
    "seed": 1,
    "disp_file": "lcurve.out",
    "disp_freq": 100,
    "save_freq": 1000
  }
}
```

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

- `systems` provide paths of the training data systems. DeePMD-kit allows you to provide multiple systems with different numbers of atoms. This key can be a `list` or a `str`.
 - `list`: `systems` gives the training data systems.
 - `str`: `systems` should be a valid path. DeePMD-kit will recursively search all data systems in this path.
- At each training step, DeePMD-kit randomly 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 switched according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when use_srtab is provided.

sw_rmin:

type: float, optional
 argument path: model/sw_rmin

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

sw_rmax:

type: float, optional
 argument path: model/sw_rmax

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

type_embedding:

type: dict, optional
 argument path: model/type_embedding

The type embedding.

neuron:

type: list, optional, default: [8]
argument path: model/type_embedding/neuron

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

activation_function:

type: str, optional, default: tanh
argument path: model/type_embedding/activation_function

The activation function in the embedding net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”. 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, default: None
argument path: model/type_embedding/seed

Random seed for parameter initialization

descriptor:

type: dict
argument path: model/descriptor

The descriptor of atomic environment.

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

type:

type: str (flag key)
argument path: model/descriptor/type
possible choices: *loc_frame*, *se_e2_a*, *se_e3*, *se_a_tpe*, *se_e2_r*, *hybrid*, *se_atten*

The type of the descriptor. See explanation below.

- `loc_frame`: Defines a local frame at each atom, and the compute the descriptor as local coordinates under this frame.
- `se_e2_a`: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor.
- `se_e2_r`: Used by the smooth edition of Deep Potential. Only the distance between atoms is used to construct the descriptor.
- `se_e3`: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Three-body embedding will be used by this descriptor.
- `se_a_tpe`: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Type embedding will be used by this descriptor.
- `se_atten`: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Attention mechanism will be used by this descriptor.
- `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_types embedding nets. Otherwise, building N_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_{types} embedding nets. Otherwise, building N_{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

When `type` is set to `se_atten`:

sel:

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

argument path: model/descriptor[se_atten]/sel

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

- int. The maximum number of neighbor atoms to be considered. We recommend it to be less than 200.
- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. Only the summation of sel[i] matters, and it is recommended to be less than 200.
- str. Can be “auto:factor” or “auto”. “factor” is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors within 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_atten]/rcut

The cut-off radius.

rcut_smth:

type: float, optional, default: 0.5

argument path: model/descriptor[se_atten]/rcut_smth

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

neuron:

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

argument path: model/descriptor[se_atten]/neuron

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

axis_neuron:

type: `int`, optional, default: 4, alias: `n_axis_neuron`
argument path: `model/descriptor[se_atten]/axis_neuron`
Size of the submatrix of G (embedding matrix).

activation_function:

type: `str`, optional, default: `tanh`
argument path:
`model/descriptor[se_atten]/activation_function`
The activation function in the embedding net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu_tf”. 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_atten]/resnet_dt`
Whether to use a “Timestep” in the skip connection

type_one_side:

type: `bool`, optional, default: `False`
argument path: `model/descriptor[se_atten]/type_one_side`
Whether to consider the information from only one side or both sides.

precision:

type: `str`, optional, default: `default`
argument path: `model/descriptor[se_atten]/precision`
The precision of the embedding net parameters, supported options are “default”, “float16”, “float32”, “float64”. Default follows the interface precision.

trainable:

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

seed:

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

exclude_types:

type: `list`, optional, default: `[]`
argument path: `model/descriptor[se_atten]/exclude_types`
The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

set_davg_zero:
 type: bool, optional, default: False
 argument path: model/descriptor[se_atten]/set_davg_zero
 Set the normalization average to zero. This option should be set when atom_ener in the energy fitting is used

attn:
 type: int, optional, default: 128
 argument path: model/descriptor[se_atten]/attn
 The length of hidden vectors in attention layers

attn_layer:
 type: int, optional, default: 2
 argument path: model/descriptor[se_atten]/attn_layer
 The number of attention layers

attn_dotr:
 type: bool, optional, default: True
 argument path: model/descriptor[se_atten]/attn_dotr
 Whether to do dot product with the normalized relative coordinates

attn_mask:
 type: bool, optional, default: False
 argument path: model/descriptor[se_atten]/attn_mask
 Whether to do mask on the diagonal in the attention matrix

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 tihs list should be equal to `len(neuron)+1`.

rcond:

type: float, optional, default: `0.001`

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

The condition number used to determine the 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 \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 Å

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 definition of learning rate

scale_by_worker:

type: str, optional, default: linear

argument path: learning_rate/scale_by_worker

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

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

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 non-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 none-zero value, the atom_pref label should be provided by file atom_pref.npy in each data system. If both start_pref_atom_pref and limit_pref_atom_pref are set to 0, then the atom_pref will be ignored.

limit_pref_pf:

type: int | float, optional, default: 0.0
 argument path: loss[ener]/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/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/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.

nvnmmd:

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 *.npy 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] DEEPMO INFO    ---Summary of the training-----
[0] DEEPMO INFO    distributed
[0] DEEPMO INFO    world size:          4
[0] DEEPMO INFO    my rank:             0
[0] DEEPMO INFO    node list:           ['exp-13-57']
[0] DEEPMO INFO    running on:          exp-13-57
[0] DEEPMO INFO    computing device:    gpu:0
[0] DEEPMO INFO    CUDA_VISIBLE_DEVICES: 0,1,2,3
[0] DEEPMO INFO    Count of visible GPU: 4
[0] DEEPMO INFO    num_intra_threads:    0
[0] DEEPMO INFO    num_inter_threads:    0
[0] DEEPMO 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,
```

(continues on next page)

(continued from previous page)

```

    "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"
}

```

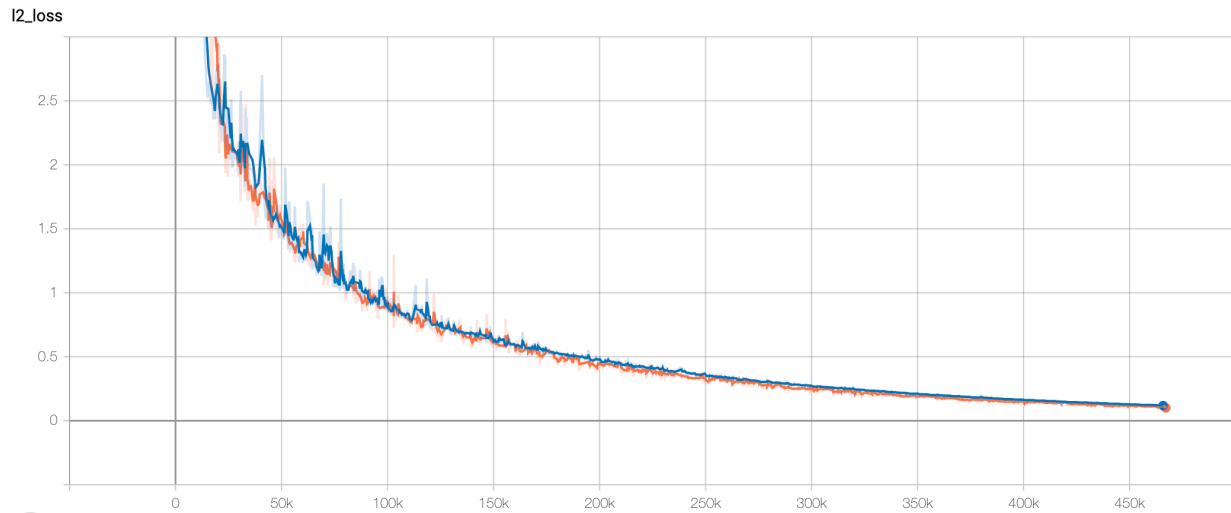
Once you have event files, run TensorBoard and provide the log directory. This should print that TensorBoard has started. Next, connect to `http://tensorboard_server_ip:6006`.

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`”.

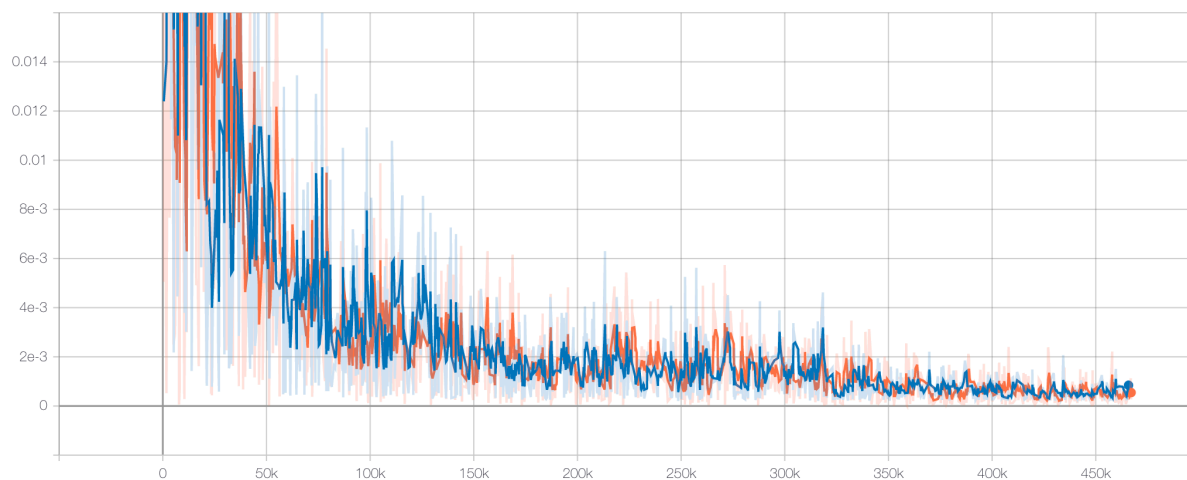
```
tensorboard --logdir path/to/logs
```

5.5.3 Examples

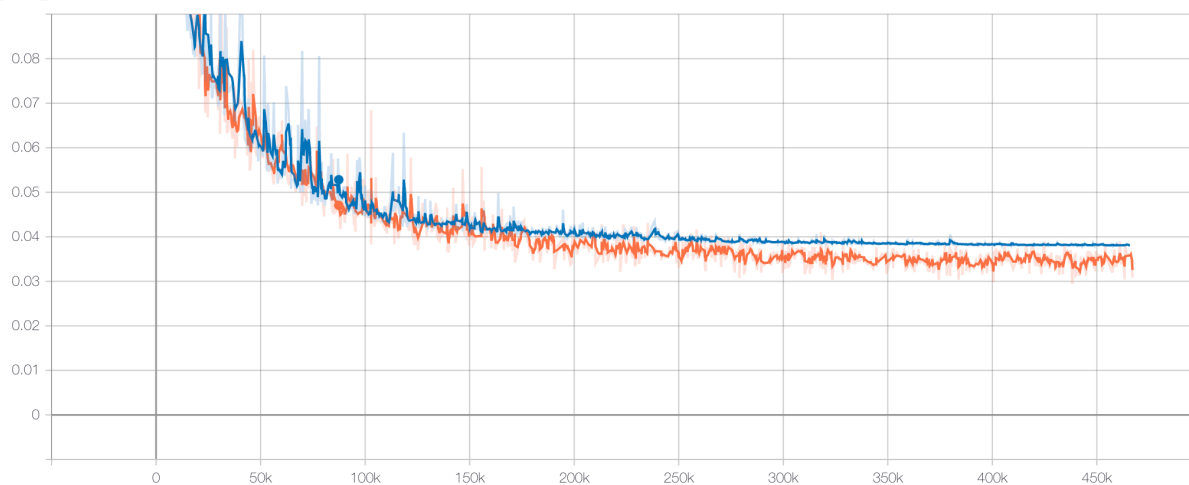
Tracking and visualizing loss metrics(red:train, blue:test)



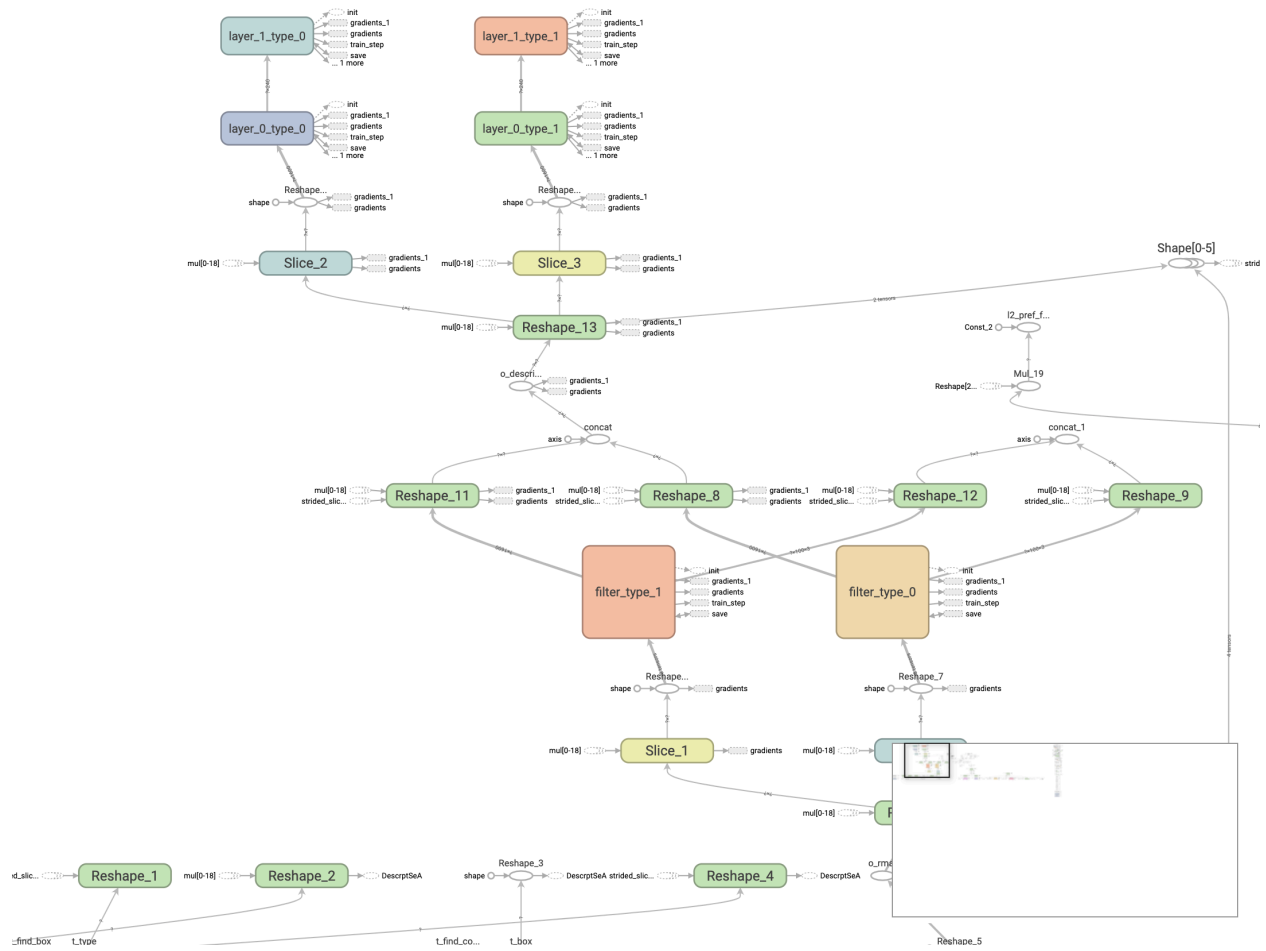
l2_ener_loss



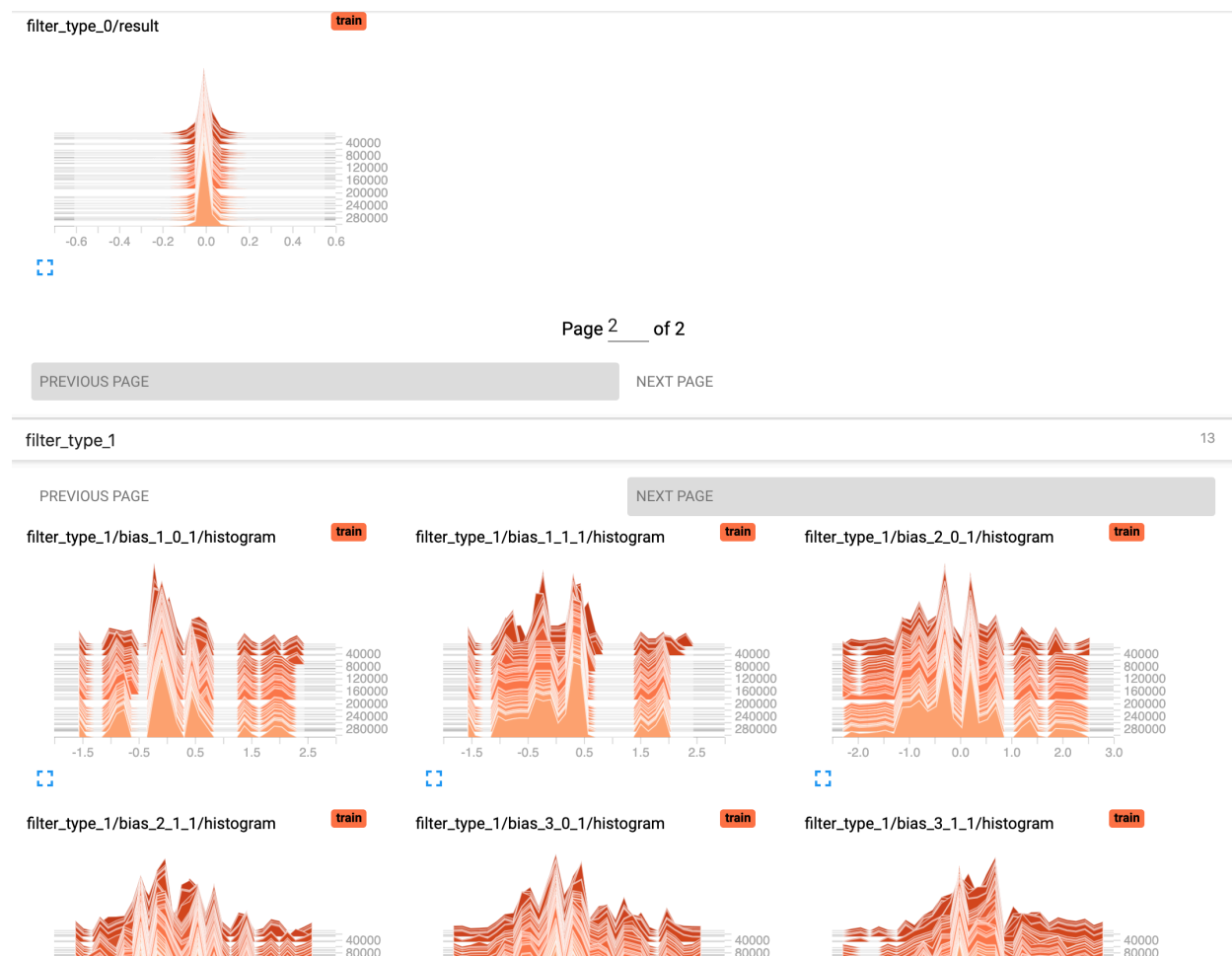
l2_force_loss



Visualizing deepmd-kit model graph



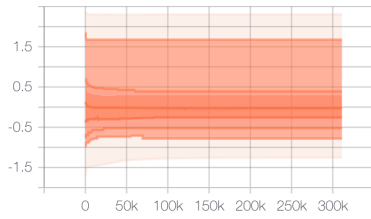
Viewing histograms of weights, biases, or other tensors as they change over time



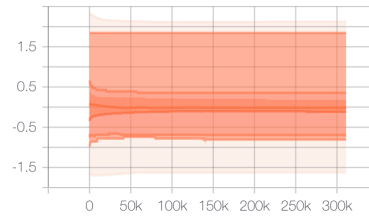
filter_type_0

[PREVIOUS PAGE](#)[NEXT PAGE](#)

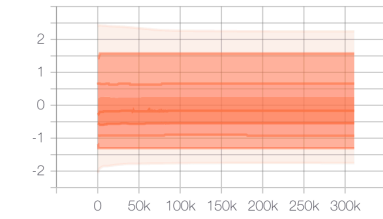
filter_type_0/bias_1_0_1/histogram



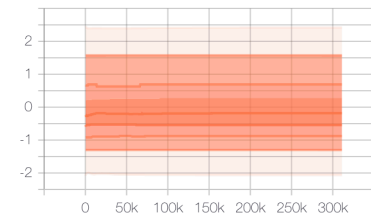
filter_type_0/bias_1_1_1/histogram



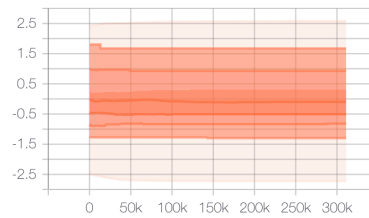
filter_type_0/bias_2_0_1/histogram



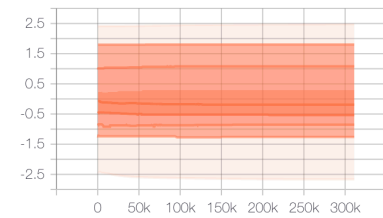
filter_type_0/bias_2_1_1/histogram



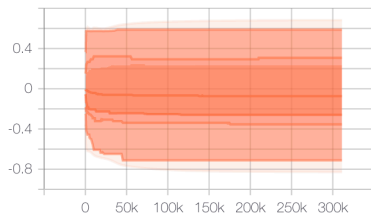
filter_type_0/bias_3_0_1/histogram



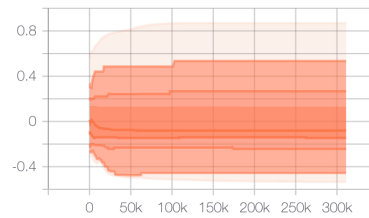
filter_type_0/bias_3_1_1/histogram



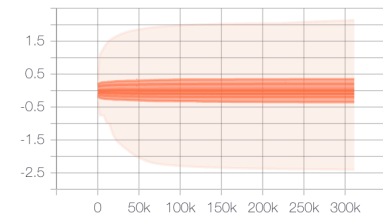
filter_type_0/matrix_1_0_1/histogram



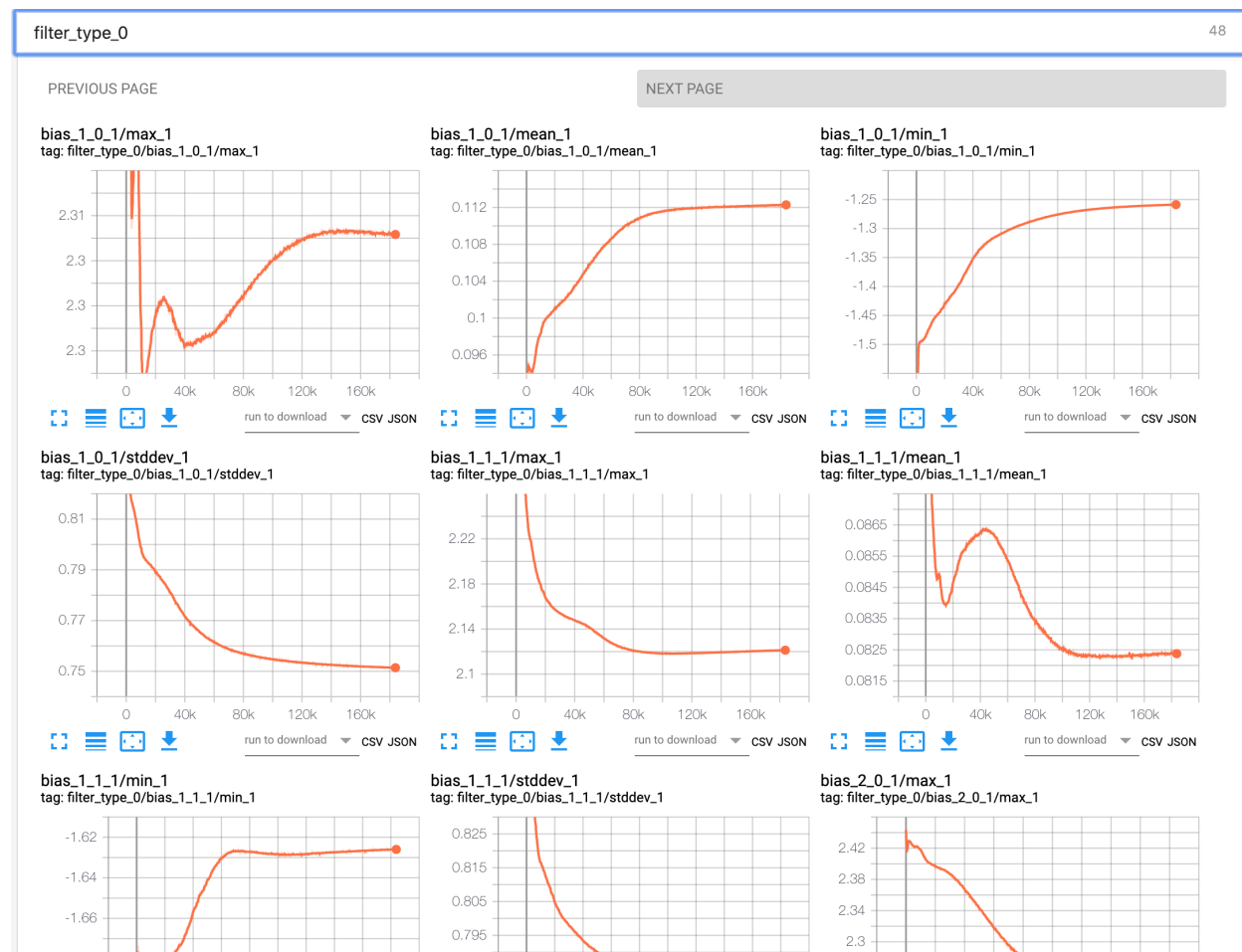
filter_type_0/matrix_1_1_1/histogram



filter_type_0/matrix_2_0_1/histogram



Viewing summaries of trainable variables



5.5.4 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>	<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>
<code>-l, --log-path</code>	<p>set log file to log messages to disk, if not specified, the logs will only be output to console</p>
<code>-o, --output</code>	<p>the output json file</p> <p>Default: “input.json”</p>

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

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

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

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

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

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

<code>--shuffle-test</code>	Shuffle test data Default: False
<code>-d, --detail-file</code>	File where details of energy force and virial accuracy will be written
<code>-a, --atomic</code>	Test the accuracy of atomic label, i.e. energy / tensor (dipole, polar) Default: False

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

9.3.6 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

<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>-m, --mpi-log</code>	Possible choices: master, collect, workers Set the manner of logging when running with MPI. 'master' logs only on main process, 'collect' broadcasts logs from workers to master and 'workers' means each process will output its own log Default: "master"
<code>-i, --input</code>	The original frozen model, which will be compressed by the code Default: "frozen_model.pb"
<code>-o, --output</code>	The compressed model Default: "frozen_model_compressed.pb"
<code>-s, --step</code>	Model compression uses fifth-order polynomials to interpolate the embedding-net. It introduces two tables with different step size to store the parameters of the polynomials. The first table covers the range of the training data, while the second table is an extrapolation of the training data. The domain of each table is uniformly divided by a given step size. And the step(parameter) denotes the step size of the first table and the second table will use 10 * step as it's step size to save the memory. Usually the value ranges from 0.1 to 0.001. Smaller step means higher accuracy and bigger model size Default: 0.01

- e, --extrapolate The domain range of the first table is automatically detected by the code: [d_low, d_up]. While the second table ranges from the first table's upper boundary(d_up) to the extrapolate(parameter) * d_up: [d_up, extrapolate * d_up]
Default: 5
- f, --frequency The frequency of tabulation overflow check(Whether the input environment matrix overflow the first or second table range). By default do not check the overflow
Default: -1
- c, --checkpoint-folder path to checkpoint folder
Default: "model-compression"
- t, --training-script The training script of the input frozen model

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

9.3.7 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

examples: `dp model-devi -m graph.000.pb graph.001.pb graph.002.pb graph.003.pb -s ./data -o model_devi.out`

9.3.9 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”

examples: dp convert-from 1.0 -i graph.pb -o graph_new.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 ...] [--one-type]
```

Named Arguments

- v, --log-level Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0
 set verbosity level by string or number, 0=ERROR, 1=WARNING,
 2=INFO and 3=DEBUG
 Default: “INFO”
- l, --log-path set log file to log messages to disk, if not specified, the logs will only be
 output to console
- s, --system The system dir. Recursively detect systems in this directory
 Default: “.”
- r, --rcut cutoff radius
- t, --type-map type map
- one-type treat all types as a single type. Used with se_atten descriptor.
 Default: False

examples: dp neighbor-stat -s data -r 6.0 -t O H

9.3.11 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  cgner  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  cgner  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 peroidic condition is passed to DeepMD.

Run Simulation

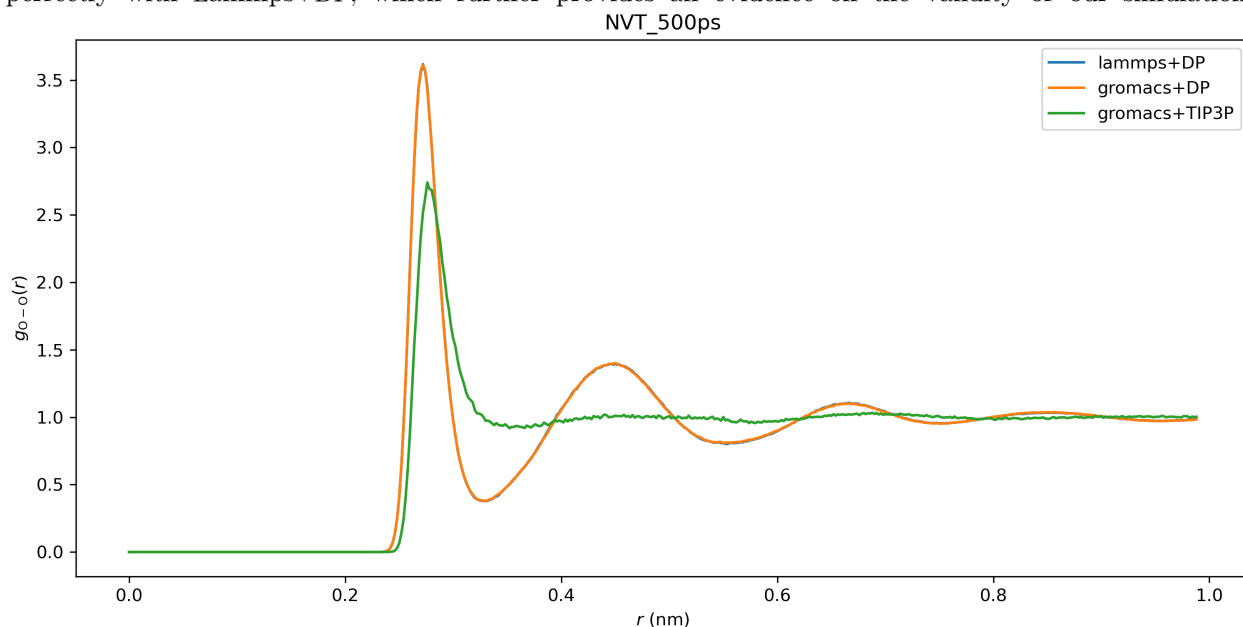
Finally, you can run GROMACS using `gmx mdrun` as usual.

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.

NVNMD

Current user:test1 [Return to home page](#)

Remaining calculation time:6:22:29

Task name	<input type="text" value="test"/>
Upload mode [?]	<input type="button" value="Manual upload"/> <input type="button" value="Automatic upload"/>
Input script	<input type="button" value="Browse..."/> in.lmp
Model file	<input type="button" value="Browse..."/> model.pb
Data files	<input type="button" value="Browse..."/> coord.lmp

- 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	Cancel			

Then, click **Refresh** to view the latest status of all calculation tasks.

11.4.3 Cancelling calculation

For the task whose calculation status is **Pending** and **Running**, you can click the corresponding **Cancel** on the homepage to stop the calculation (Figure.5).

NVNMD

Current user: test1 [Logout](#)

Remaining calculation time: 6:21:09

[Add a new task](#)

[Operation records](#)

Calculation records [Refresh](#)
[Clear calculation records](#)

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

11.4.4 Downloading results

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

Click **Package** to download a zipped package of all files including input files and output results (Figure.6).

NVNMD

Current user: test1 [Return to home page](#)

Remaining calculation time: 6:21:09

Files

Name	Size	Download directly	Download from online data storage	Upload to online data storage ^②
output.zip	1.2 MB	Download		Upload

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

NVNMD

Current user:test1 [Return to home page](#)

Remaining calculation time:6:21:09

Files

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

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

11.4.5 Deleting record

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

11.4.6 Clearing records

Click **Clear calculation records** on the homepage to clear all records.

Records cannot be retrieved after clearing.

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 Al₂O₃: (Fitting-net size: [240,240,240])

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

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

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

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

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

12.1.3 Water

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

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

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

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

12.1.4 Mg-Al

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

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

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

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

12.2 How to control the 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-DEEPMO to a LAMMPS source code folder and re-install LAMMPS.

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

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

and then install the new USER-DEEPMO.

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.

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

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.

`sess` Get TF session.

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]: $2 \leq i < \text{Ntypes} + 2$, 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.

property sess: `tensorflow.python.client.session.Session`

Get TF session.

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⁻¹

Attributes

`model_type` Get type of model.

`model_version` Get version of model.

`sess` Get TF session.

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 ####

`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]`: $2 \leq i < \text{Ntypes} + 2$, 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 statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

`data_coord [list[np.ndarray]]` The coordinates. Can be generated by `deepmd.model.stat.make_stat_input()`

`data_box [list[np.ndarray]]` The box. Can be generated by `deepmd.model.stat.make_stat_input()`

`data_atype [list[np.ndarray]]` The atom types. Can be generated by `deepmd.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.stat.make_stat_input()`

`mesh [list[np.ndarray]]` The mesh for neighbor searching. Can be generated by `deepmd.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 statisitcs (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: 'frozon_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 ≤ 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()` \rightarrow `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` = '') \rightarrow `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` = '') \rightarrow `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) \rightarrow `None`

Pass the descript_reshape tensor as well as descript_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]`: $2 \leq i < Ntypes+2$, 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`

Concatenate 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 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_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 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: 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 statisitcs (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

Reveive the mixed precision setting.

Parameters

mixed_prec The mixed precision setting used in the embedding net

get_dim_out() → int

Returns the output dimension of this descriptor

get_nlist_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 descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def

Parameters

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

prod_force_virial(atom_ener: 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.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

<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 radisu
<i>get_rot_mat</i> ()	Get rotational matrix
<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 descrypt_reshape tensor as well as descrypt_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: 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]`: $2 \leq i < Ntypes + 2$, number of type *i* atoms

Returns

force The force on atoms

virial The total virial

atom_virial The atomic virial

deepmd.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

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

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 maximum number of type `i` atoms in the cut-off radius
`neuron` [`list` [`int`]] Number of neurons in each hidden layers of the embedding net \mathcal{N}
`axis_neuron` Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)
`resnet_dt` Time-step `dt` in the resnet construction: $y = x + dt * \phi(Wx + b)$

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

References

[1]

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_rot_mat</i> ()	Get rotational matrix
<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> (descrpt_reshape, ...)	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]: $2 \leq i < \text{Ntypes} + 2$, 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
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_{types} embedding nets. Otherwise, building N_{types}^2 embedding nets

type_nchanl Number of channels for type representation

type_nlayer Number of hidden layers for the type embedding net (skip connected).

numb_aparam Number of atomic parameters. If > 0 it will be embedded with atom types.

set_davg_zero Set the shift of embedding net input to zero.

activation_function The activation function in the embedding net. Supported options are {0}

precision The precision of the embedding net parameters. Supported options are {1}

exclude_types [*List*[*List*[*int*]]] The excluded pairs of types which have no interaction with each other. For example, $[[0, 1]]$ means no interaction between type 0 and type 1.

Attributes

precision Precision of filter network.

Methods

<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]`: $2 \leq i < Ntypes+2$, 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

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 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 radisu
<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]`: $2 \leq i < Ntypes+2$, 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 $N_{\text{types}} + 2$ **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

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 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

deepmd.descriptor.se_atten module

class `deepmd.descriptor.se_atten.DescriptSeAtten(*args, **kwargs)`

Bases: `deepmd.descriptor.se_a.DescriptSeA`

Parameters

rcut The cut-off radius r_c

rcut_smth From where the environment matrix should be smoothed r_s

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

neuron [`list[int]`] Number of neurons in each hidden layers of the embedding net \mathcal{N}

axis_neuron Number of the axis neuron M_2 (number of columns of the sub-matrix of the embedding matrix)

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

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

`attn` The length of hidden vector during scale-dot attention computation.

`attn_layer` The number of layers in attention mechanism.

`attn_dotr` Whether to dot the relative coordinates on the attention weights as a gated scheme.

`attn_mask` Whether to mask the diagonal in the attention weights.

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 ≤ 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, mixed_type: bool = False, real_natoms_vec: Optional[list] = None) → None

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

Parameters

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

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

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

natoms_vec The vector for the number of atoms of the system and different types of atoms. If mixed_type is True, this para is blank. See real_natoms_vec.

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

input_dict Dictionary for additional input

mixed_type Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different natoms_vec(s), with the same nloc.

real_natoms_vec If mixed_type is True, it takes in the real natoms_vec for each frame.

init_variables(graph: 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

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 radisu
<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 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 = 'frozon_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()

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]`: $2 \leq i < Ntypes+2$, number of type *i* atoms

Returns

force The force on atoms

virial The total virial

atom_virial The atomic virial

deepmd.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 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 radisu
<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

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 * 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 training 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], one_type: bool = False, **kwargs)`

Calculate neighbor statistics.

Parameters

system [str] system to stat

rcut [float] cutoff radius

type_map [list[str]] type map

one_type [bool, optional, default=False] treat all types as a single type

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] munber of tests to do
rand_seed [Optional[int]] seed for random generator
shuffle_test [bool] whether to shuffle tests
detail_file [Optional[str]] file where test details will be output
atomic [bool] whether per atom quantities should be computed

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, 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.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], one_type: bool = False, **kwargs)`

Calculate neighbor statistics.

Parameters

`system` [`str`] system to stat

`rcut` [`float`] cutoff radius

`type_map` [`list[str]`] type map

`one_type` [`bool`, optional, default=False] treat all types as a single type

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] munber of tests to do
rand_seed [Optional[int]] seed for random generator
shuffle_test [bool] whether to shuffle tests
detail_file [Optional[str]] file where test details will be output
atomic [bool] whether per atom quantities should be computed

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. ϕ 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 ϕ 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[, mixed_type])	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 numbfparam > 0, should have input_dict['fparam'] if numbaparam > 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 numbfparam > 0 must have all_stat['fparam'] if numbaparam > 0 must have all_stat['aparam'] can be prepared by model.make_stat_input

protection Divided-by-zero protection

`compute_output_stats(all_stat: dict, mixed_type: bool = False) → None`

Compute the output statistics

Parameters

`all_stat` must have the following components: `all_stat['energy']` of shape `n_sys x n_batch x n_frame` can be prepared by `model.make_stat_input`

`mixed_type` Whether to perform the mixed_type mode. If True, the input data has the mixed_type format (see doc/model/train_se_atten.md), in which frames in a system may have different `natoms_vec(s)`, with the same `nloc`.

`enable_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, descript)
```

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(descript: 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

`descript` [`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.WFCFitting(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.
sess Get TF session.

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.
`sess` Get TF session.

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.

property sess: `tensorflow.python.client.session.Session`

Get TF session.

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.

sess Get TF session.

Methods

<i>eval</i> (coords, cells, atom_types[, atomic, ...])	Evaluate the model.
<i>eval_full</i> (coords, cells, atom_types[, ...])	Evaluate the model with interface similar to the energy model.
<i>get_dim_aparam</i> ()	Unsupported in this model.
<i>get_dim_fparam</i> ()	Unsupported in this model.
<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> ()	Get the selected atom types of 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) → `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.

sess Get TF session.

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.
sess Get TF session.

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 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.

`sess` Get TF session.

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⁻¹

Attributes

`model_type` Get type of model.

`model_version` Get version of model.

`sess` Get TF session.

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_ainparam()</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⁻¹

Attributes

model_type Get type of model.

model_version Get version of model.

sess Get TF session.

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.

`sess` Get TF session.

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.

`sess` Get TF session.

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]`) → `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.

property sess: `tensorflow.python.client.session.Session`
 Get TF session.

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.
sess Get TF session.

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.

`sess` Get TF session.

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.

`sess` Get TF session.

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 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.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.
sess Get TF session.

Methods

<i>eval</i> (coords, cells, atom_types[, atomic, ...])	Evaluate the model.
<i>eval_full</i> (coords, cells, atom_types[, ...])	Evaluate the model with interface similar to the energy model.
<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> ()	Get the selected atom types of 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 = 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.

`sess` Get TF session.

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>
<https://stackoverflow.com/questions/56085015/suppress-openmp-debug-messages-when-running-tensorflow-on-cpu>

Submodules

deepmd.loggers.loggers module

Logger initialization for package.

`deepmd.loggers.loggers.set_log_handles`(level: int, log_path: Optional[Path] = None, mpi_log: Optional[str] = None)

Set desired level for package loggers and add file handlers.

Parameters

level: int logging level

log_path: Optional[str] path to log file, if None logs will be send only to console. If the parent directory does not exist it will be automatically created, by default None

mpi_log [Optional[str], optional] mpi log type. Has three options. master will output logs to file and console only from rank==0. collect will write messages from all ranks to one file opened under rank==0 and to console. workers will open one log file for each worker designated by its rank, console behaviour is the same as for collect. If this argument is specified, package 'mpi4py' must be already installed. by default None

Raises

RuntimeError If the argument mpi_log is specified, package mpi4py is not installed.

Notes

Logging levels:

	our notation	python logging	tensorflow cpp	OpenMP
debug	10	10	0	1/on/true/yes
info	20	20	1	0/off/false/no
warning	30	30	2	0/off/false/no
error	40	40	3	0/off/false/no

References

<https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U> <https://stackoverflow.com/questions/35869137/avoid-tensorflow-print-on-standard-error>
<https://stackoverflow.com/questions/56085015/suppress-openmp-debug-messages-when-running-tensorflow-on-cpu>

deepmd.loss package

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

<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

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

<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.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

<i>init_variables</i> (graph, graph_def[, ...])	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`(descript, 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</code> (graph, graph_def[, ...])	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 nvnm title when logging

NVNMD_CITATION citation of nvnm

Submodules

deepmd.nvnmd.data.data module

deepmd.nvnmd.descriptor package

nvnm.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 `nvnm_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:

$$r_{ji}^2 \rightarrow s_{ji}$$

$$r_{ji}^2 \rightarrow sr_{ji}$$

$$r_{ji}^2 \rightarrow \mathcal{G}_{ji}$$

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:

$$y = f(x) = y_k + (x - x_k) * dy_k$$

$$y_k = f(x_k)$$

$$dy_k = \frac{f(x_{k+1}) - f(x_k)}{dx}$$

$$x_k \leq x < x_{k+1}$$

$$x_k = k * dx$$

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()

```

```
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:

$$r_{ji}^2 \rightarrow s_{ji}$$

$$r_{ji}^2 \rightarrow sr_{ji}$$

$$r_{ji}^2 \rightarrow \mathcal{G}_{ji}$$

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:

$$y = f(x) = y_k + (x - x_k) * dy_k$$

$$y_k = f(x_k)$$

$$dy_k = \frac{f(x_{k+1}) - f(x_k)}{dx}$$

$$x_k \leq x < x_{k+1}$$

$$x_k = k * dx$$

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(*, nvnm_config: Optional[str] = 'nvnm/config.npy',  
                                     nvnm_weight: Optional[str] = 'nvnm/weight.npy',  
                                     nvnm_map: Optional[str] = 'nvnm/map.npy', nvnm_model:  
                                     Optional[str] = 'nvnm/model.pb', **kwargs)
```

deepmd.nvnmd.fit package

nvnm.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([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.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~ $n_{\text{type}}-1$

`spe_j(int)` special order of neighbor atom j 0~ $n_{\text{type}}-1$

`layer_l` layer order in embedding network 1~ n_{layer}

`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~ $n_{\text{type}}-1$

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

`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_nvnmd_jdata()</i>	Generate nvnmd 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_nvnmd_jdata()

Generate nvnmd 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 taining 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, suffix='')`

`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

<i>get_all_argument</i> ([exclude_hybrid])	Get all arguments.
<i>register</i> (name[, alias])	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_atten_args()
deepmd.utils.argcheck.descrpt_se_r_args()
deepmd.utils.argcheck.descrpt_se_t_args()
deepmd.utils.argcheck.descrpt_variant_type_args(exclude_hybrid: bool = False) →
    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_numbatch(batch_size, set_idx)</code>	Get the number of batches in a set.
<code>get_numset()</code>	Get number of training sets
<code>get_sys_numbatch(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

<code>check_type_map_consistency</code>	
<code>compute_energy_shift</code>	
<code>format_name_length</code>	
<code>get_batch</code>	
<code>get_batch_size</code>	
<code>get_nbatches</code>	
<code>get_nsystems</code>	
<code>get_ntypes</code>	
<code>get_sys</code>	
<code>get_test</code>	
<code>get_type_map</code>	
<code>numb_fparam</code>	
<code>print_summary</code>	
<code>process_sys_weights</code>	

`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_nctest([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_attention_layer_nodes_from_graph_def(graph_def: tensor-
                                                            flow.core.framework.graph_pb2.GraphDef,
                                                            suffix: str = '') → Dict

Get the attention layer nodes with the given tf.GraphDef object

Parameters
    graph_def The input tf.GraphDef object
    suffix [str, optional] The scope suffix

Returns
    Dict The attention layer nodes within the given tf.GraphDef object

deepmd.utils.graph.get_attention_layer_variables_from_graph_def(graph_def: tensor-
                                                                flow.core.framework.graph_pb2.GraphDef,
                                                                suffix: str = '') → Dict

Get the attention layer variables with the given tf.GraphDef object

Parameters
    graph_def [tf.GraphDef] The input tf.GraphDef object
    suffix [str, optional] The suffix of the scope

Returns
    Dict The attention layer variables within the given tf.GraphDef object
```

`deepmd.utils.graph.get_embedding_net_nodes(model_file: str, suffix: str = '') → Dict`

Get the embedding net nodes with the given frozen model(model_file)

Parameters

`model_file` The input frozen model path

`suffix` [`str`, optional] The suffix of the scope

Returns

`Dict` The embedding net nodes with the given frozen model

`deepmd.utils.graph.get_embedding_net_nodes_from_graph_def(graph_def: 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: tensor-
                                                         flow.core.framework.graph_pb2.GraphDef,
                                                         suffix: str = '') → Dict
```

Get the fitting net nodes with the given tf.GraphDef object

Parameters

graph_def The input tf.GraphDef object

suffix suffix of the scope

Returns

Dict The fitting net nodes within the given tf.GraphDef object

```
deepmd.utils.graph.get_fitting_net_variables(model_file: str, suffix: str = '') → Dict
```

Get the fitting net variables with the given frozen model(model_file)

Parameters

model_file The input frozen model path

suffix suffix of the scope

Returns

Dict The fitting net variables within the given frozen model

```
deepmd.utils.graph.get_fitting_net_variables_from_graph_def(graph_def: tensor-
                                                            flow.core.framework.graph_pb2.GraphDef,
                                                            suffix: str = '') → Dict
```

Get the fitting net variables with the given tf.GraphDef object

Parameters

graph_def The input tf.GraphDef object

suffix suffix of the scope

Returns

Dict The fitting net variables within the given tf.GraphDef object

```
deepmd.utils.graph.get_pattern_nodes_from_graph_def(graph_def: tensor-
                                                    flow.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: tensor-
                                                                    flow.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: tensor-
                                                                    flow.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 α is the learning rate, α_0 is the starting learning rate, λ is the decay rate, and τ is the decay steps.

Parameters

`start_lr` Starting learning rate α_0

`stop_lr` Stop learning rate α_1

`decay_steps` Learning rate decay every this number of steps τ

`decay_rate` The decay rate λ . If `stop_step` is provided in `build`, then it will be determined automatically and overwritten.

Methods

<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, one_type: bool = False)

Bases: object

Class for getting training data information.

It loads data from DeepmdData object, and measures the data info, including nearest nbor distance between atoms, max nbor size of atoms and the output data range of the environment matrix.

Parameters

ntypes The num of atom types

rcut The cut-off radius

one_type [bool, optional, default=False] Treat all types as a single type.

Methods

<i>get_stat</i> (data)	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. ϕ is the activation function.

Parameters

xx [**Tensor**] Input tensor x of shape [-1,1]
network_size: list of int Size of the embedding network. For example [16,32,64]
precision: Precision of network weights. For example, tf.float64
activation_fn: Activation function ϕ
resnet_dt: 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 intial 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', scope='',
                                reuse=None, seed=None, use_timestep=False, trainable=True,
                                useBN=False, uniform_seed=False, initial_variables=None,
                                mixed_prec=None, final_layer=False)

deepmd.utils.network.one_layer_rand_seed_shift()

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

Methods

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

deepmd.utils.parallel_op module

```
class deepmd.utils.parallel_op.ParallelOp(builder: Callable[[...], Tuple[Dict[str,
    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

<i>glob</i> (pattern)	Search path using the glob pattern.
<i>is_dir</i> ()	Check if self is directory.
<i>is_file</i> ()	Check if self is file.
<i>load_numpy</i> ()	Load NumPy array.
<i>load_txt</i> ([dtype])	Load NumPy array from text.
<i>rglob</i> (pattern)	This is like calling <i>DPPath.glob()</i> with <i>**/</i> 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

<i>glob</i> (pattern)	Search path using the glob pattern.
<i>is_dir</i> ()	Check if self is directory.
<i>is_file</i> ()	Check if self is file.
<i>load_numpy</i> ()	Load NumPy array.
<i>load_txt</i> (**kwargs)	Load NumPy array from text.
<i>rglob</i> (pattern)	This is like calling <i>DPPath.glob()</i> with <i>**/</i> 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() → bool

Check if self is file.

abstract load_numpy() → numpy.ndarray

Load NumPy array.

Returns

np.ndarray loaded NumPy array

abstract load_txt(**kwargs) → numpy.ndarray

Load NumPy array from text.

Returns

np.ndarray loaded NumPy array

abstract `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

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

<code>build</code> (<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: Optional[str] = 'tanh', precision:
                                             str = 'default', trainable: bool = True, seed:
                                             Optional[int] = None, uniform_seed: bool = False,
                                             padding: 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

padding Concat the zero padding to the output, as the default embedding of empty type.

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 compatibil-
            ity, by default ["energy", "forces", "stress"]

        system_changes [List[str], optional] unused, only for function signature com-
            patibility, 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 *.npv file e.g. forces or energy

ndof [`int`] number of the degrees of freedom, this is tied to atomic parameter e.g. forces have atomic=True and ndof=3

atomic [`bool`, `optional`] specifies whwther the ndof keyworrd applies to per atom quantity or not, by default `False`

must [`bool`, `optional`] specifi if the *.npv data file must exist, by default `False`

high_prec [`bool`, `optional`] if tru load data to np.float64 else np.float32, by default `False`

type_sel [`bool`, `optional`] select only certain type of atoms, by default `None`

repeat [`int`, `optional`] if specify repaeat data repeat times, by default 1

default [`float`, `optional`, default=0.] default value of data

`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: Optional[_ACTIVATION]) → Optional[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]` corresponding 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 precision constant

Raises

`RuntimeError` if string is invalid

`deepmd.common.get_precision(precision: _PRECISION) → Any`

Convert str to TF DType constant.

Parameters

`precision` [_PRECISION] one of the allowed precisions

Returns

`tf.python.framework.dtypes.DType` appropriate TF constant

Raises

`RuntimeError` if supplied precision string does not have a corresponding TF constant

`deepmd.common.j_loader(filename: Union[str, 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 op_module

Python wrappers around TensorFlow ops.

This file is MACHINE GENERATED! Do not edit.

```
deepmd.env.op_module.Descript(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, sel_a, sel_r,  
                               axis_rule, name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **axis_rule** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist, axis, rot_mat).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. axis: A Tensor of type int32. rot_mat: A Tensor. Has the same type as coord.

```
deepmd.env.op_module.DescriptNorot(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r,  
                                   rcut_r_smth, sel_a, sel_r, name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.DescriptSeA(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r,  
                                  rcut_r_smth, sel_a, sel_r, name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.DescriptSeAEf(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.DescriptSeAEfPara(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.

- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.DescriptSeAEfVert`(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.DescriptSeR`(coord, type, natoms, box, mesh, davg, dstd, rcut, rcut_smth, sel, name=None)

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.

- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut** – A float.
- **rcut_smth** – A float.
- **sel** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.EwaldRecp(coord, charge, natoms, box, ewald_beta, ewald_h, name=None)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **charge** – A Tensor. Must have the same type as coord.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **ewald_beta** – A float.
- **ewald_h** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (energy, force, virial).

energy: A Tensor. Has the same type as coord. force: A Tensor. Has the same type as coord. virial: A Tensor. Has the same type as coord.

`deepmd.env.op_module.Gelu(x, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as x.

`deepmd.env.op_module.GeluCustom(x, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.

- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as x.

`deepmd.env.op_module.GeluGrad(dy, x, name=None)`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as dy.

`deepmd.env.op_module.GeluGradCustom(dy, x, name=None)`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as dy.

`deepmd.env.op_module.GeluGradGrad(dy, dy_, x, name=None)`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **dy_** – A Tensor. Must have the same type as dy.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as dy.

`deepmd.env.op_module.GeluGradGradCustom(dy, dy_, x, name=None)`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **dy_** – A Tensor. Must have the same type as dy.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as dy.

`deepmd.env.op_module.MapAparam(aparam, nlist, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **aparam** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.

- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as aparam.

`deepmd.env.op_module.MapNvnmd(x, v, dv, grad_v, grad_dv, prec, nbit, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **v** – A Tensor. Must have the same type as x.
- **dv** – A Tensor. Must have the same type as x.
- **grad_v** – A Tensor. Must have the same type as x.
- **grad_dv** – A Tensor. Must have the same type as x.
- **prec** – A float.
- **nbit** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as x.

`deepmd.env.op_module.MatmulNvnmd(x, w, isround, nbit1, nbit2, nbit3, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **isround** – An int.
- **nbit1** – An int.
- **nbit2** – An int.
- **nbit3** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as x.

`deepmd.env.op_module.NeighborStat(coord, type, natoms, box, mesh, rcut, name=None)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **rcut** – A float.

- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (max_nbor_size, min_nbor_dist).

max_nbor_size: A Tensor of type int32. min_nbor_dist: A Tensor. Has the same type as coord.

```
deepmd.env.op_module.PairTab(table_info, table_data, type, rij, nlist, natoms, scale, sel_a, sel_r,  
                             name=None)
```

TODO: add doc.

Parameters

- **table_info** – A Tensor of type float64.
- **table_data** – A Tensor of type float64.
- **type** – A Tensor of type int32.
- **rij** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **scale** – A Tensor. Must have the same type as rij.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (atom_energy, force, atom_virial).

atom_energy: A Tensor. Has the same type as rij. force: A Tensor. Has the same type as rij. atom_virial: A Tensor. Has the same type as rij.

```
deepmd.env.op_module.ParallelProdForceSeA(net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel,  
                                           parallel=False, start_frac=0, end_frac=1, name=None)
```

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **parallel** – An optional bool. Defaults to False.
- **start_frac** – An optional float. Defaults to 0.
- **end_frac** – An optional float. Defaults to 1.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as net_deriv.

```
deepmd.env.op_module.ProdEnvMatA(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r,
                                  rcut_r_smth, sel_a, sel_r, name=None)
```

Compute the environment matrix for descriptor se_e2_a.

Each row of the environment matrix \mathcal{R}^i can be constructed as follows

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

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

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

Note that the environment matrix is normalized by davg and dstd.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64. The coordinates of atoms.
- **type** – A Tensor of type int32. The types of atoms.
- **natoms** – A Tensor of type int32. The number of atoms. This tensor has the length of Ntypes + 2. natoms[0]: number of local atoms. natoms[1]: total number of atoms held by this processor. natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms.
- **box** – A Tensor. Must have the same type as coord. The box of frames.
- **mesh** – A Tensor of type int32. For historical reasons, only the length of the Tensor matters. If size of mesh == 6, pbc is assumed. If size of mesh == 0, no-pbc is assumed.
- **davg** – A Tensor. Must have the same type as coord. Average value of the environment matrix for normalization.
- **dstd** – A Tensor. Must have the same type as coord. Standard deviation of the environment matrix for normalization.
- **rcut_a** – A float. This argument is not used.
- **rcut_r** – A float. The cutoff radius for the environment matrix.
- **rcut_r_smth** – A float. From where the environment matrix should be smoothed.
- **sel_a** – A list of ints. sel_a[i] specifies the maximum number of type i atoms in the cut-off radius.
- **sel_r** – A list of ints. This argument is not used.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. The environment matrix.
descript_deriv: A Tensor. Has the same type as coord. The derivative of the environment matrix.
rij: A Tensor. Has the same type as coord. The distance between the atoms.
nlist: A Tensor of type int32. The neighbor list of each atom.

```
deepmd.env.op_module.ProdEnvMatAMix(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r,  
                                     rcut_r_smth, sel_a, sel_r, name=None)
```

Compute the environment matrix mixing the atom types.

The sorting of neighbor atoms depends not on atom types, but on the distance and index. The atoms in nlist matrix will gather forward and thus save space for gaps of types in ProdEnvMatA, resulting in optimized and relative small sel_a.

The additional outputs are listed as following:

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist, ntype, nmask).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same type as coord.
rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. ntype: A Tensor of type int32. The corresponding atom types in nlist. nmask: A Tensor of type bool. The atom mask in nlist.

```
deepmd.env.op_module.ProdEnvMatANvnmQuantize(coord, type, natoms, box, mesh, davg, dstd, rcut_a,  
                                              rcut_r, rcut_r_smth, sel_a, sel_r, name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.

- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.ProdEnvMatR(coord, type, natoms, box, mesh, davg, dstd, rcut, rcut_smth, sel,
                                  name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut** – A float.
- **rcut_smth** – A float.
- **sel** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.ProdForce(net_deriv, in_deriv, nlist, axis, natoms, n_a_sel, n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.

- **in_deriv** – A Tensor. Must have the same type as `net_deriv`.
- **nlist** – A Tensor of type `int32`.
- **axis** – A Tensor of type `int32`.
- **natoms** – A Tensor of type `int32`.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as `net_deriv`.

```
deepmd.env.op_module.ProdForceNorot(net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel,
                                     name=None)
```

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: `float32`, `float64`.
- **in_deriv** – A Tensor. Must have the same type as `net_deriv`.
- **nlist** – A Tensor of type `int32`.
- **natoms** – A Tensor of type `int32`.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as `net_deriv`.

```
deepmd.env.op_module.ProdForceSeA(net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: `float32`, `float64`.
- **in_deriv** – A Tensor. Must have the same type as `net_deriv`.
- **nlist** – A Tensor of type `int32`.
- **natoms** – A Tensor of type `int32`.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as `net_deriv`.

```
deepmd.env.op_module.ProdForceSeR(net_deriv, in_deriv, nlist, natoms, name=None)
```

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: `float32`, `float64`.
- **in_deriv** – A Tensor. Must have the same type as `net_deriv`.
- **nlist** – A Tensor of type `int32`.

- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as `net_deriv`.

```
deepmd.env.op_module.ProdVirial(net_deriv, in_deriv, rij, nlist, axis, natoms, n_a_sel, n_r_sel,
                                name=None)
```

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as `net_deriv`.
- **rij** – A Tensor. Must have the same type as `net_deriv`.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as `net_deriv`. atom_virial: A Tensor. Has the same type as `net_deriv`.

```
deepmd.env.op_module.ProdVirialNorot(net_deriv, in_deriv, rij, nlist, natoms, n_a_sel, n_r_sel,
                                      name=None)
```

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as `net_deriv`.
- **rij** – A Tensor. Must have the same type as `net_deriv`.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as `net_deriv`. atom_virial: A Tensor. Has the same type as `net_deriv`.

```
deepmd.env.op_module.ProdVirialSeA(net_deriv, in_deriv, rij, nlist, natoms, n_a_sel, n_r_sel,
                                   name=None)
```

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **rij** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

```
deepmd.env.op_module.ProdVirialSeR(net_deriv, in_deriv, rij, nlist, natoms, name=None)
```

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **rij** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

```
deepmd.env.op_module.QuantizeNvnmd(x, isround, nbit1, nbit2, nbit3, name=None)
```

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **isround** – An int.
- **nbit1** – An int.
- **nbit2** – An int.
- **nbit3** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as x.

`deepmd.env.op_module.SoftMinForce(du, sw_deriv, nlist, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **du** – A Tensor. Must be one of the following types: float32, float64.
- **sw_deriv** – A Tensor. Must have the same type as du.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as du.

`deepmd.env.op_module.SoftMinSwitch(type, rij, nlist, natoms, sel_a, sel_r, alpha, rmin, rmax, name=None)`

TODO: add doc.

Parameters

- **type** – A Tensor of type int32.
- **rij** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **alpha** – A float.
- **rmin** – A float.
- **rmax** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (sw_value, sw_deriv).

sw_value: A Tensor. Has the same type as rij. sw_deriv: A Tensor. Has the same type as rij.

`deepmd.env.op_module.SoftMinVirial(du, sw_deriv, rij, nlist, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **du** – A Tensor. Must be one of the following types: float32, float64.
- **sw_deriv** – A Tensor. Must have the same type as du.
- **rij** – A Tensor. Must have the same type as du.
- **nlist** – A Tensor of type int32.

- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as du. atom_virial: A Tensor. Has the same type as du.

`deepmd.env.op_module.TabulateFusion(table, table_info, em_x, em, last_layer_size, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionGrad(table, table_info, em_x, em, dy, descriptor, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionGradGrad(table, table_info, em_x, em, dz_dy_dem_x, dz_dy_dem, descriptor, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.

- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionSeA(table, table_info, em_x, em, last_layer_size, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionSeAGrad(table, table_info, em_x, em, dy, descriptor, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionSeAGradGrad(table, table_info, em_x, em, dz_dy_dem_x, dz_dy_dem, descriptor, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.

- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionSeR(table, table_info, em, last_layer_size, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionSeRGrad(table, table_info, em, dy, descriptor, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionSeRGradGrad(table, table_info, em, dz_dy_dem, descriptor,
name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionSeT(table, table_info, em_x, em, last_layer_size, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionSeTGrad(table, table_info, em_x, em, dy, descriptor, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionSeTGradGrad(table, table_info, em_x, em, dz_dy_dem_x, dz_dy_dem, descriptor, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

`deepmd.env.op_module.Tanh2Nvnmd(x, isround, nbit1, nbit2, nbit3, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **isround** – An int.
- **nbit1** – An int.
- **nbit2** – An int.
- **nbit3** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as x.

`deepmd.env.op_module.Tanh4Nvnmd(x, isround, nbit1, nbit2, nbit3, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **isround** – An int.
- **nbit1** – An int.
- **nbit2** – An int.
- **nbit3** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as x.

`deepmd.env.op_module.UnaggregatedDy2Dx(z, w, dy_dx, dy2_dx, ybar, functype, name=None)`

TODO: add doc.

Parameters

- **z** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as z.
- **dy_dx** – A Tensor. Must have the same type as z.
- **dy2_dx** – A Tensor. Must have the same type as z.
- **ybar** – A Tensor. Must have the same type as z.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as z.

`deepmd.env.op_module.UnaggregatedDy2DxS(y, dy, w, xbar, functype, name=None)`

TODO: add doc.

Parameters

- **y** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as y.
- **w** – A Tensor. Must have the same type as y.

- **xbar** – A Tensor. Must have the same type as y.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as y.

`deepmd.env.op_module.UnaggregatedDyDx(z, w, dy_dx, ybar, functype, name=None)`

TODO: add doc.

Parameters

- **z** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as z.
- **dy_dx** – A Tensor. Must have the same type as z.
- **ybar** – A Tensor. Must have the same type as z.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as z.

`deepmd.env.op_module.UnaggregatedDyDxS(y, w, xbar, functype, name=None)`

TODO: add doc.

Parameters

- **y** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as y.
- **xbar** – A Tensor. Must have the same type as y.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as y.

`deepmd.env.op_module.descript(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, sel_a, sel_r, axis_rule, name=None)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **sel_a** – A list of ints.

- **sel_r** – A list of ints.
- **axis_rule** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist, axis, rot_mat).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same

type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

axis: A Tensor of type int32. rot_mat: A Tensor. Has the same type as coord.

```
deepmd.env.op_module.descript_norot(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r,  
                                     rcut_r_smth, sel_a, sel_r, name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same

type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.descript_se_a(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r,  
                                    rcut_r_smth, sel_a, sel_r, name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.

- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.descript_se_a_ef(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r,
                                     rcut_r_smth, sel_a, sel_r, name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.descript_se_a_ef_para(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a,
                                           rcut_r, rcut_r_smth, sel_a, sel_r, name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.descript_se_a_ef_vert(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a,  
                                           rcut_r, rcut_r_smth, sel_a, sel_r, name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.descript_se_r(coord, type, natoms, box, mesh, davg, dstd, rcut, rcut_smth, sel,
                                   name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut** – A float.
- **rcut_smth** – A float.
- **sel** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.ewald_recp(coord, charge, natoms, box, ewald_beta, ewald_h, name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **charge** – A Tensor. Must have the same type as coord.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **ewald_beta** – A float.
- **ewald_h** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (energy, force, virial).

energy: A Tensor. Has the same type as coord. force: A Tensor. Has the same type as coord. virial: A Tensor. Has the same type as coord.

`deepmd.env.op_module.gelu(x, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as x.

`deepmd.env.op_module.gelu_custom(x, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as x.

`deepmd.env.op_module.gelu_grad(dy, x, name=None)`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as dy.

`deepmd.env.op_module.gelu_grad_custom(dy, x, name=None)`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as dy.

`deepmd.env.op_module.gelu_grad_grad(dy, dy_, x, name=None)`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **dy_** – A Tensor. Must have the same type as dy.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as dy.

`deepmd.env.op_module.gelu_grad_grad_custom(dy, dy_, x, name=None)`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as dy.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as dy.

`deepmd.env.op_module.map_aparam(aparam, nlist, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **aparam** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as aparam.

`deepmd.env.op_module.map_nvnmmd(x, v, dv, grad_v, grad_dv, prec, nbit, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **v** – A Tensor. Must have the same type as x.
- **dv** – A Tensor. Must have the same type as x.
- **grad_v** – A Tensor. Must have the same type as x.
- **grad_dv** – A Tensor. Must have the same type as x.
- **prec** – A float.
- **nbit** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as x.

`deepmd.env.op_module.matmul_nvnmmd(x, w, isround, nbit1, nbit2, nbit3, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **isround** – An int.
- **nbit1** – An int.
- **nbit2** – An int.
- **nbit3** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as x.

`deepmd.env.op_module.neighbor_stat(coord, type, natoms, box, mesh, rcut, name=None)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **rcut** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (max_nbor_size, min_nbor_dist).

max_nbor_size: A Tensor of type int32. min_nbor_dist: A Tensor. Has the same type as coord.

`deepmd.env.op_module.pair_tab(table_info, table_data, type, rij, nlist, natoms, scale, sel_a, sel_r, name=None)`

TODO: add doc.

Parameters

- **table_info** – A Tensor of type float64.
- **table_data** – A Tensor of type float64.
- **type** – A Tensor of type int32.
- **rij** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **scale** – A Tensor. Must have the same type as rij.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (atom_energy, force, atom_virial).

atom_energy: A Tensor. Has the same type as rij. force: A Tensor. Has the same type as rij. atom_virial: A Tensor. Has the same type as rij.

`deepmd.env.op_module.parallel_prod_force_se_a(net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel, parallel=False, start_frac=0, end_frac=1, name=None)`

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **parallel** – An optional bool. Defaults to False.
- **start_frac** – An optional float. Defaults to 0.
- **end_frac** – An optional float. Defaults to 1.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as net_deriv.

`deepmd.env.op_module.prod_env_mat_a(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)`

Compute the environment matrix for descriptor se_e2_a.

Each row of the environment matrix \mathcal{R}^i can be constructed as follows

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

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

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

Note that the environment matrix is normalized by davg and dstd.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64. The coordinates of atoms.
- **type** – A Tensor of type int32. The types of atoms.
- **natoms** – A Tensor of type int32. The number of atoms. This tensor has the length of Ntypes + 2. natoms[0]: number of local atoms. natoms[1]: total number of atoms held by this processor. natoms[i]: 2 <= i < Ntypes+2, number of type i atoms.
- **box** – A Tensor. Must have the same type as coord. The box of frames.
- **mesh** – A Tensor of type int32. For historical reasons, only the length of the Tensor matters. If size of mesh == 6, pbc is assumed. If size of mesh == 0, no-pbc is assumed.

- **davg** – A Tensor. Must have the same type as coord. Average value of the environment matrix for normalization.
- **dstd** – A Tensor. Must have the same type as coord. Standard deviation of the environment matrix for normalization.
- **rcut_a** – A float. This argument is not used.
- **rcut_r** – A float. The cutoff radius for the environment matrix.
- **rcut_r_smth** – A float. From where the environment matrix should be smoothed.
- **sel_a** – A list of ints. sel_a[i] specifies the maximum number of type i atoms in the cut-off radius.
- **sel_r** – A list of ints. This argument is not used.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. The environment matrix.
descript_deriv: A Tensor. Has the same type as coord. The derivative of the environment matrix.
rij: A Tensor. Has the same type as coord. The distance between the atoms.
nlist: A Tensor of type int32. The neighbor list of each atom.

```
deepmd.env.op_module.prod_env_mat_a_mix(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r,  
                                         rcut_r_smth, sel_a, sel_r, name=None)
```

Compute the environment matrix mixing the atom types.

The sorting of neighbor atoms depends not on atom types, but on the distance and index. The atoms in nlist matrix will gather forward and thus save space for gaps of types in ProdEnvMatA, resulting in optimized and relative small sel_a.

The additional outputs are listed as following:

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist, ntype, nmask).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. ntype: A Tensor of type int32. The corresponding atom types in nlist. nmask: A Tensor of type bool. The atom mask in nlist.

```
deepmd.env.op_module.prod_env_mat_a_nvnmmd_quantize(coord, type, natoms, box, mesh, davg, dstd,
                                                    rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r,
                                                    name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut_a** – A float.
- **rcut_r** – A float.
- **rcut_r_smth** – A float.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.prod_env_mat_r(coord, type, natoms, box, mesh, davg, dstd, rcut, rcut_smth, sel,
                                    name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.

- **rcut** – A float.
- **rcut_smth** – A float.
- **sel** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.prod_force(net_deriv, in_deriv, nlist, axis, natoms, n_a_sel, n_r_sel,
                                name=None)
```

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as net_deriv.

```
deepmd.env.op_module.prod_force_norot(net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel,
                                       name=None)
```

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in_deriv** – A Tensor. Must have the same type as net_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as net_deriv.

```
deepmd.env.op_module.prod_force_se_a(net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel,
                                      name=None)
```

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: float32, float64.

- **in_deriv** – A Tensor. Must have the same type as `net_deriv`.
- **nlist** – A Tensor of type `int32`.
- **natoms** – A Tensor of type `int32`.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as `net_deriv`.

`deepmd.env.op_module.prod_force_se_r(net_deriv, in_deriv, nlist, natoms, name=None)`

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: `float32`, `float64`.
- **in_deriv** – A Tensor. Must have the same type as `net_deriv`.
- **nlist** – A Tensor of type `int32`.
- **natoms** – A Tensor of type `int32`.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as `net_deriv`.

`deepmd.env.op_module.prod_virial(net_deriv, in_deriv, rij, nlist, axis, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: `float32`, `float64`.
- **in_deriv** – A Tensor. Must have the same type as `net_deriv`.
- **rij** – A Tensor. Must have the same type as `net_deriv`.
- **nlist** – A Tensor of type `int32`.
- **axis** – A Tensor of type `int32`.
- **natoms** – A Tensor of type `int32`.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (`virial`, `atom_virial`).

`virial`: A Tensor. Has the same type as `net_deriv`. `atom_virial`: A Tensor. Has the same type as `net_deriv`.

`deepmd.env.op_module.prod_virial_norot(net_deriv, in_deriv, rij, nlist, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: `float32`, `float64`.

- **in_deriv** – A Tensor. Must have the same type as `net_deriv`.
- **rij** – A Tensor. Must have the same type as `net_deriv`.
- **nlist** – A Tensor of type `int32`.
- **natoms** – A Tensor of type `int32`.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (`virial`, `atom_virial`).

`virial`: A Tensor. Has the same type as `net_deriv`. `atom_virial`: A Tensor. Has the same type as `net_deriv`.

```
deepmd.env.op_module.prod_virial_se_a(net_deriv, in_deriv, rij, nlist, natoms, n_a_sel, n_r_sel,
                                     name=None)
```

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: `float32`, `float64`.
- **in_deriv** – A Tensor. Must have the same type as `net_deriv`.
- **rij** – A Tensor. Must have the same type as `net_deriv`.
- **nlist** – A Tensor of type `int32`.
- **natoms** – A Tensor of type `int32`.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (`virial`, `atom_virial`).

`virial`: A Tensor. Has the same type as `net_deriv`. `atom_virial`: A Tensor. Has the same type as `net_deriv`.

```
deepmd.env.op_module.prod_virial_se_r(net_deriv, in_deriv, rij, nlist, natoms, name=None)
```

TODO: add doc.

Parameters

- **net_deriv** – A Tensor. Must be one of the following types: `float32`, `float64`.
- **in_deriv** – A Tensor. Must have the same type as `net_deriv`.
- **rij** – A Tensor. Must have the same type as `net_deriv`.
- **nlist** – A Tensor of type `int32`.
- **natoms** – A Tensor of type `int32`.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as net_deriv. atom_virial: A Tensor. Has the same type as net_deriv.

`deepmd.env.op_module.quantize_nvnmmd(x, isround, nbit1, nbit2, nbit3, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **isround** – An int.
- **nbit1** – An int.
- **nbit2** – An int.
- **nbit3** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as x.

`deepmd.env.op_module.soft_min_force(du, sw_deriv, nlist, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **du** – A Tensor. Must be one of the following types: float32, float64.
- **sw_deriv** – A Tensor. Must have the same type as du.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as du.

`deepmd.env.op_module.soft_min_switch(type, rij, nlist, natoms, sel_a, sel_r, alpha, rmin, rmax, name=None)`

TODO: add doc.

Parameters

- **type** – A Tensor of type int32.
- **rij** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **sel_a** – A list of ints.
- **sel_r** – A list of ints.
- **alpha** – A float.
- **rmin** – A float.

- **rmax** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (sw_value, sw_deriv).

sw_value: A Tensor. Has the same type as rij. sw_deriv: A Tensor. Has the same type as rij.

`deepmd.env.op_module.soft_min_virial(du, sw_deriv, rij, nlist, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **du** – A Tensor. Must be one of the following types: float32, float64.
- **sw_deriv** – A Tensor. Must have the same type as du.
- **rij** – A Tensor. Must have the same type as du.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom_virial).

virial: A Tensor. Has the same type as du. atom_virial: A Tensor. Has the same type as du.

`deepmd.env.op_module.tabulate_fusion(table, table_info, em_x, em, last_layer_size, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

`deepmd.env.op_module.tabulate_fusion_grad(table, table_info, em_x, em, dy, descriptor, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.

- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_grad_grad(table, table_info, em_x, em, dz_dy_dem_x,
                                              dz_dy_dem, descriptor, name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_a(table, table_info, em_x, em, last_layer_size,
                                          name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_a_grad(table, table_info, em_x, em, dy, descriptor,
                                              name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.

- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_a_grad_grad(table, table_info, em_x, em, dz_dy_dem_x,  
                                                    dz_dy_dem, descriptor, name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_r(table, table_info, em, last_layer_size, name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_r_grad(table, table_info, em, dy, descriptor, name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.

- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_r_grad_grad(table, table_info, em, dz_dy_dem, descriptor,
                                                    name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_t(table, table_info, em_x, em, last_layer_size,
                                           name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last_layer_size** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_t_grad(table, table_info, em_x, em, dy, descriptor,
                                                name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy_dem_x, dy_dem).

dy_dem_x: A Tensor. Has the same type as table. dy_dem: A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_t_grad_grad(table, table_info, em_x, em, dz_dy_dem_x,
                                                    dz_dy_dem, descriptor, name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table_info** – A Tensor. Must have the same type as table.
- **em_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz_dy_dem_x** – A Tensor. Must have the same type as table.
- **dz_dy_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as table.

```
deepmd.env.op_module.tanh2_nvnmmd(x, isround, nbit1, nbit2, nbit3, name=None)
```

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **isround** – An int.
- **nbit1** – An int.
- **nbit2** – An int.
- **nbit3** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as x.

```
deepmd.env.op_module.tanh4_nvnmmd(x, isround, nbit1, nbit2, nbit3, name=None)
```

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **isround** – An int.
- **nbit1** – An int.
- **nbit2** – An int.
- **nbit3** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as x.

`deepmd.env.op_module.unaggregated_dy2_dx(z, w, dy_dx, dy2_dx, ybar, functype, name=None)`

TODO: add doc.

Parameters

- **z** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as z.
- **dy_dx** – A Tensor. Must have the same type as z.
- **dy2_dx** – A Tensor. Must have the same type as z.
- **ybar** – A Tensor. Must have the same type as z.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as z.

`deepmd.env.op_module.unaggregated_dy2_dx_s(y, dy, w, xbar, functype, name=None)`

TODO: add doc.

Parameters

- **y** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as y.
- **w** – A Tensor. Must have the same type as y.
- **xbar** – A Tensor. Must have the same type as y.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as y.

`deepmd.env.op_module.unaggregated_dy_dx(z, w, dy_dx, ybar, functype, name=None)`

TODO: add doc.

Parameters

- **z** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as z.
- **dy_dx** – A Tensor. Must have the same type as z.
- **ybar** – A Tensor. Must have the same type as z.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as z.

`deepmd.env.op_module.unaggregated_dy_dx_s(y, w, xbar, functype, name=None)`

TODO: add doc.

Parameters

- **y** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as y.
- **xbar** – A Tensor. Must have the same type as y.

- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as y.

17.2 op_grads_module

Python wrappers around TensorFlow ops.

This file is MACHINE GENERATED! Do not edit.

```
deepmd.env.op_grads_module.ProdForceGrad(grad, net_deriv, in_deriv, nlist, axis, natoms, n_a_sel,
                                           n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.ProdForceSeAGrad(grad, net_deriv, in_deriv, nlist, natoms, n_a_sel,
                                              n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.ProdForceSeRGrad(grad, net_deriv, in_deriv, nlist, natoms, name=None)`

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.ProdVirialGrad(grad, net_deriv, in_deriv, rij, nlist, axis, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.ProdVirialSeAGrad(grad, net_deriv, in_deriv, rij, nlist, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.

- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.ProdVirialSeRGrad(grad, net_deriv, in_deriv, rij, nlist, natoms,  
                                              name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.SoftMinForceGrad(grad, du, sw_deriv, nlist, natoms, n_a_sel, n_r_sel,  
                                             name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **du** – A Tensor. Must have the same type as grad.
- **sw_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.SoftMinVirialGrad(grad, du, sw_deriv, rij, nlist, natoms, n_a_sel, n_r_sel,  
                                              name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **du** – A Tensor. Must have the same type as grad.
- **sw_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.

- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.prod_force_grad(grad, net_deriv, in_deriv, nlist, axis, natoms, n_a_sel,
                                             n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.prod_force_se_a_grad(grad, net_deriv, in_deriv, nlist, natoms, n_a_sel,
                                                  n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.prod_force_se_r_grad(grad, net_deriv, in_deriv, nlist, natoms,
                                                  name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.

- **in_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.prod_virial_grad(grad, net_deriv, in_deriv, rij, nlist, axis, natoms,  
                                             n_a_sel, n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.prod_virial_se_a_grad(grad, net_deriv, in_deriv, rij, nlist, natoms,  
                                                  n_a_sel, n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.prod_virial_se_r_grad(grad, net_deriv, in_deriv, rij, nlist, natoms,  
                                                  name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net_deriv** – A Tensor. Must have the same type as grad.
- **in_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.soft_min_force_grad(grad, du, sw_deriv, nlist, natoms, n_a_sel, n_r_sel,
                                                name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **du** – A Tensor. Must have the same type as grad.
- **sw_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.soft_min_virial_grad(grad, du, sw_deriv, rij, nlist, natoms, n_a_sel,
                                                n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **du** – A Tensor. Must have the same type as grad.
- **sw_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n_a_sel** – An int.
- **n_r_sel** – An int.
- **name** – A name for the operation (optional).

Returns A Tensor. Has the same type as grad.

18.1 Class Hierarchy

- Namespace deepmd
 - Struct deepmd_exception
 - Struct NeighborListData
 - Struct tf_exception
 - Template Class AtomMap
 - Class DeepPot
 - Class DeepPotModelDevi
 - Class DeepTensor
 - Class DipoleChargeModifier

18.2 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

18.3 Full API

18.3.1 Namespaces

Namespace `deepmd`

Contents

- Classes
- Functions
- Typedefs

Classes

- Struct `deepmd_exception`
- 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

18.3.2 Classes and Structs

Struct `deepmd_exception`

- Defined in `file_source_api_cc_include_common.h`

Inheritance Relationships

Derived Type

- `public deepmd::tf_exception` (`Struct tf_exception`)

Struct Documentation

struct `deepmd_exception`

Subclassed by `deepmd::tf_exception`

Struct NeighborListData

- Defined in file_source_api_cc_include_common.h

Struct Documentation

struct **NeighborListData**

Public Functions

```
void copy_from_nlist(const InputNlist &inlist)
void shuffle(const std::vector<int> &fwd_map)
void shuffle(const deepmd::AtomMap<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.
- **imp_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()

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 evaluate, usually would be the atomic tensor.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.

```
void compute(std::vector<VALUETYPE> &value, const std::vector<VALUETYPE> &coord, const
            std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost,
            const InputNlist &inlist)
```

Evaluate the value by using this model.

Parameters

- **value** – [out] The value to evaluate, usually would be the atomic tensor.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.

```
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 = "")

~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

18.3.3 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 = "")
```

18.3.4 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
```


19.1 Class Hierarchy

- Namespace deepmd
 - Struct deepmd_exception
 - Struct deepmd_exception_oom
 - Template Struct EwaldParameters
 - Struct InputNlist
 - Template Struct Region
- Template Struct DescrptSeRGPUExecuteFunctor
- Template Struct GeluGPUExecuteFunctor
- Template Struct GeluGradGPUExecuteFunctor
- Template Struct GeluGradGradGPUExecuteFunctor
- Template Struct ProdForceSeAGPUExecuteFunctor
- Template Struct ProdForceSeRGPUExecuteFunctor
- Template Struct ProdVirialSeAGPUExecuteFunctor
- Template Struct ProdVirialSeRGPUExecuteFunctor
- Template Struct TabulateCheckerGPUExecuteFunctor
- Template Struct TabulateFusionGPUExecuteFunctor
- Template Struct TabulateFusionGradGPUExecuteFunctor
- Template Class SimulationRegion

19.2 File Hierarchy

- dir_source
 - dir_source_lib
 - * dir_source_lib_include
 - file_source_lib_include_ComputeDescriptor.h
 - file_source_lib_include_coord.h
 - file_source_lib_include_device.h
 - file_source_lib_include_DeviceFunctor.h
 - file_source_lib_include_env_mat.h
 - file_source_lib_include_env_mat_nvnmd.h
 - file_source_lib_include_errors.h
 - file_source_lib_include_ewald.h
 - file_source_lib_include_fmt_nlist.h
 - file_source_lib_include_gelu.h
 - file_source_lib_include_gpu_cuda.h
 - file_source_lib_include_gpu_rocm.h
 - file_source_lib_include_map_aparam.h
 - file_source_lib_include_neighbor_list.h
 - file_source_lib_include_pair_tab.h
 - file_source_lib_include_prod_env_mat.h
 - file_source_lib_include_prod_env_mat_nvnmd.h
 - file_source_lib_include_prod_force.h
 - file_source_lib_include_prod_force_grad.h
 - file_source_lib_include_prod_virial.h
 - file_source_lib_include_prod_virial_grad.h
 - file_source_lib_include_region.h
 - file_source_lib_include_SimulationRegion.h
 - file_source_lib_include_SimulationRegion_Impl.h
 - file_source_lib_include_soft_min_switch.h
 - file_source_lib_include_soft_min_switch_force.h
 - file_source_lib_include_soft_min_switch_force_grad.h
 - file_source_lib_include_soft_min_switch_virial.h
 - file_source_lib_include_soft_min_switch_virial_grad.h
 - file_source_lib_include_switcher.h
 - file_source_lib_include_tabulate.h

· file_source_lib_include_utilities.h

19.3 Full API

19.3.1 Namespaces

Namespace deepmd

Contents

- Classes
- Functions
- Variables

Classes

- Struct deepmd_exception
- Struct deepmd_exception_oom
- Template Struct EwaldParameters
- Struct InputNlist
- Template Struct Region

Functions

- Template Function deepmd::build_nlist_cpu
- Template Function deepmd::build_nlist_gpu
- Template Function deepmd::compute_cell_info
- Function deepmd::convert_nlist
- Function deepmd::convert_nlist_gpu_device
- Template Function deepmd::convert_to_inter_cpu
- Template Function deepmd::convert_to_inter_gpu
- Template Function deepmd::convert_to_phys_cpu
- Template Function deepmd::convert_to_phys_gpu
- Template Function deepmd::copy_coord_cpu
- Template Function deepmd::copy_coord_gpu
- Function deepmd::cos_switch(double&, double&, const double&, const double&, const double&)
- Function deepmd::cos_switch(const double&, const double&, const double&)
- Template Function deepmd::cprod

- Function `deepmd::cum_sum`
- Template Function `deepmd::delete_device_memory`
- Template Function `deepmd::dot1`
- Template Function `deepmd::dot2`
- Template Function `deepmd::dot3`
- Template Function `deepmd::dot4`
- Template Function `deepmd::dotmv3`
- Function `deepmd::DPGetDeviceCount`
- Function `deepmd::DPSetDevice`
- Template Function `deepmd::env_mat_a_cpu`
- Template Function `deepmd::env_mat_a_nvnmmd_quantize_cpu`
- Function `deepmd::env_mat_nbor_update`
- Template Function `deepmd::env_mat_r_cpu`
- Template Function `deepmd::ewald_recip`
- Template Function `deepmd::format_nbor_list_gpu_cuda`
- Template Function `deepmd::format_nlist_cpu`
- Function `deepmd::free_nlist_gpu_device`
- Template Function `deepmd::gelu_cpu`
- Template Function `deepmd::gelu_gpu_cuda`
- Template Function `deepmd::gelu_grad_cpu`
- Template Function `deepmd::gelu_grad_gpu_cuda`
- Template Function `deepmd::gelu_grad_grad_cpu`
- Template Function `deepmd::gelu_grad_grad_gpu_cuda`
- Template Function `deepmd::init_region_cpu`
- Template Function `deepmd::invsqrt`
- Specialized Template Function `deepmd::invsqrt< double >`
- Specialized Template Function `deepmd::invsqrt< float >`
- Template Function `deepmd::malloc_device_memory(FPTYPE *amp, std::vector<FPTYPE>&)`
- Template Function `deepmd::malloc_device_memory(FPTYPE *amp, const std::vector<FPTYPE>&)`
- Template Function `deepmd::malloc_device_memory(FPTYPE *amp, const int)`
- Template Function `deepmd::malloc_device_memory_sync(FPTYPE *amp, const std::vector<FPTYPE>&)`
- Template Function `deepmd::malloc_device_memory_sync(FPTYPE *amp, const FPTYPE *, const int)`
- Template Function `deepmd::malloc_device_memory_sync(FPTYPE *amp, std::vector<FPTYPE>&)`
- Template Function `deepmd::map_aparam_cpu`
- Function `deepmd::max_numneigh`

- Template Function `deepmd::memcpy_device_to_host(FPTYPE *, std::vector<FPTYPE>&)`
- Template Function `deepmd::memcpy_device_to_host(const FPTYPE *, std::vector<FPTYPE>&)`
- Template Function `deepmd::memcpy_device_to_host(const FPTYPE *, FPTYPE *, const int)`
- Template Function `deepmd::memcpy_host_to_device(FPTYPE *, std::vector<FPTYPE>&)`
- Template Function `deepmd::memcpy_host_to_device(FPTYPE *, const std::vector<FPTYPE>&)`
- Template Function `deepmd::memcpy_host_to_device(FPTYPE *, const FPTYPE *, const int)`
- Template Function `deepmd::memset_device_memory`
- Template Function `deepmd::normalize_coord_cpu`
- Template Function `deepmd::normalize_coord_gpu`
- Template Function `deepmd::pair_tab_cpu`
- Template Function `deepmd::prod_env_mat_a_cpu`
- Template Function `deepmd::prod_env_mat_a_gpu_cuda`
- Template Function `deepmd::prod_env_mat_a_nvnmmd_quantize_cpu`
- Template Function `deepmd::prod_env_mat_r_cpu`
- Template Function `deepmd::prod_env_mat_r_gpu_cuda`
- Template Function `deepmd::prod_force_a_cpu`
- Template Function `deepmd::prod_force_a_gpu_cuda`
- Template Function `deepmd::prod_force_grad_a_cpu`
- Template Function `deepmd::prod_force_grad_a_gpu_cuda`
- Template Function `deepmd::prod_force_grad_r_cpu`
- Template Function `deepmd::prod_force_grad_r_gpu_cuda`
- Template Function `deepmd::prod_force_r_cpu`
- Template Function `deepmd::prod_force_r_gpu_cuda`
- Template Function `deepmd::prod_virial_a_cpu`
- Template Function `deepmd::prod_virial_a_gpu_cuda`
- Template Function `deepmd::prod_virial_grad_a_cpu`
- Template Function `deepmd::prod_virial_grad_a_gpu_cuda`
- Template Function `deepmd::prod_virial_grad_r_cpu`
- Template Function `deepmd::prod_virial_grad_r_gpu_cuda`
- Template Function `deepmd::prod_virial_r_cpu`
- Template Function `deepmd::prod_virial_r_gpu_cuda`
- Template Function `deepmd::soft_min_switch_cpu`
- Template Function `deepmd::soft_min_switch_force_cpu`
- Template Function `deepmd::soft_min_switch_force_grad_cpu`
- Template Function `deepmd::soft_min_switch_virial_cpu`
- Template Function `deepmd::soft_min_switch_virial_grad_cpu`

- Function `deepmd::spline3_switch`
- Template Function `deepmd::spline5_switch`
- Template Function `deepmd::tabulate_fusion_se_a_cpu`
- Template Function `deepmd::tabulate_fusion_se_a_gpu_cuda`
- Template Function `deepmd::tabulate_fusion_se_a_grad_cpu`
- Template Function `deepmd::tabulate_fusion_se_a_grad_gpu_cuda`
- Template Function `deepmd::tabulate_fusion_se_a_grad_grad_cpu`
- Template Function `deepmd::tabulate_fusion_se_a_grad_grad_gpu_cuda`
- Template Function `deepmd::tabulate_fusion_se_r_cpu`
- Template Function `deepmd::tabulate_fusion_se_r_gpu_cuda`
- Template Function `deepmd::tabulate_fusion_se_r_grad_cpu`
- Template Function `deepmd::tabulate_fusion_se_r_grad_gpu_cuda`
- Template Function `deepmd::tabulate_fusion_se_r_grad_grad_cpu`
- Template Function `deepmd::tabulate_fusion_se_r_grad_grad_gpu_cuda`
- Template Function `deepmd::tabulate_fusion_se_t_cpu`
- Template Function `deepmd::tabulate_fusion_se_t_gpu_cuda`
- Template Function `deepmd::tabulate_fusion_se_t_grad_cpu`
- Template Function `deepmd::tabulate_fusion_se_t_grad_gpu_cuda`
- Template Function `deepmd::tabulate_fusion_se_t_grad_grad_cpu`
- Template Function `deepmd::tabulate_fusion_se_t_grad_grad_gpu_cuda`
- Template Function `deepmd::test_encoding_decoding_nbor_info_gpu_cuda`
- Function `deepmd::use_nei_info_cpu`
- Function `deepmd::use_nei_info_gpu`
- Function `deepmd::use_nlist_map`
- Template Function `deepmd::volume_cpu`
- Template Function `deepmd::volume_gpu`

Variables

- Variable `deepmd::ElectrostaticConversion`

Namespace `std`

19.3.2 Classes and Structs

Struct `deepmd_exception`

- Defined in `file_source_lib_include_errors.h`

Inheritance Relationships

Base Type

- `public std::runtime_error`

Derived Type

- `public deepmd::deepmd_exception_oom (Struct deepmd_exception_oom)`

Struct Documentation

struct `deepmd_exception` : `public std::runtime_error`

General DeePMD-kit exception. Throw if anything doesn't work.

Subclassed by `deepmd::deepmd_exception_oom`

Public Functions

`inline deepmd_exception()`

`inline deepmd_exception(const std::string &msg)`

Struct `deepmd_exception_oom`

- Defined in `file_source_lib_include_errors.h`

Inheritance Relationships

Base Type

- `public deepmd::deepmd_exception (Struct deepmd_exception)`

Struct Documentation

struct `deepmd_exception_oom` : public `deepmd::deepmd_exception`

Public Functions

inline `deepmd_exception_oom()`

inline `deepmd_exception_oom(const std::string &msg)`

Template Struct `EwaldParameters`

- Defined in `file_source_lib_include_ewald.h`

Struct Documentation

template<typename `VALUETYPE`>

struct `EwaldParameters`

Public Members

`VALUETYPE rcut` = 6.0

`VALUETYPE beta` = 2

`VALUETYPE spacing` = 4

Struct `InputNlist`

- Defined in `file_source_lib_include_neighbor_list.h`

Struct Documentation

struct `InputNlist`

Construct `InputNlist` with the input LAMMPS nbor list info.

Public Functions

```
inline InputNlist()
```

```
inline InputNlist(int inum_, int *ilist_, int *numneigh_, int **firstneigh_)
```

```
inline ~InputNlist()
```

Public Members

```
int inum
```

Number of core region atoms.

```
int *ilist
```

Array stores the core region atom's index.

```
int *numneigh
```

Array stores the core region atom's neighbor atom number.

```
int **firstneigh
```

Array stores the core region atom's neighbor index.

Template Struct Region

- Defined in file_source_lib_include_region.h

Struct Documentation

```
template<typename FPTYPE>
```

```
struct Region
```

Public Functions

```
Region()
```

```
~Region()
```

Public Members

```
FPTYPE *boxt
```

```
FPTYPE *rec_boxt
```

Template Struct DescriptSeRGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

```
template<typename FPTYPE>
struct DescriptSeRGPUExecuteFunctor
```

Public Functions

```
void operator() (const FPTYPE *coord, const int *type, const int *ilist, const int *jrange, const int
                 *jlist, int *array_int, unsigned long long *array_longlong, const FPTYPE *avg,
                 const FPTYPE *std, FPTYPE *descript, FPTYPE *descript_deriv, FPTYPE *rij,
                 int *nlist, const int nloc, const int nall, const int nnei, const int ndescript, const float
                 rcut_r, const float rcut_r_smth, const std::vector<int> sec_a, const bool fill_nei_a,
                 const int MAGIC_NUMBER)
```

Template Struct GeluGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

```
template<typename FPTYPE>
struct GeluGPUExecuteFunctor
```

Public Functions

```
void operator() (const FPTYPE *in, FPTYPE *out, const int size)
```

Template Struct GeluGradGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

```
template<typename FPTYPE>
struct GeluGradGPUExecuteFunctor
```

Public Functions

void **operator()** (const FPTYPE *dy, const FPTYPE *in, FPTYPE *out, const int size)

Template Struct GeluGradGradGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename FPTYPE>

struct **GeluGradGradGPUExecuteFunctor**

Public Functions

void **operator()** (const FPTYPE *dy, const FPTYPE *dy_, const FPTYPE *in, FPTYPE *out, const int size)

Template Struct ProdForceSeAGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

template<typename FPTYPE>

struct **ProdForceSeAGPUExecuteFunctor**

Public Functions

void **operator()** (FPTYPE *force, const FPTYPE *net_derive, const FPTYPE *in_deriv, const int *nlist, const int nloc, const int nall, const int nnei, const int ndescript, const int n_a_sel, const int n_a_shift)

Template Struct ProdForceSeRGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

```
template<typename FPTYPE>
```

```
struct ProdForceSeRGPUExecuteFunctor
```

Public Functions

```
void operator() (FPTYPE *force, const FPTYPE *net_derive, const FPTYPE *in_deriv, const int  
                *nlist, const int nloc, const int nall, const int nnei, const int ndescript)
```

Template Struct ProdVirialSeAGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

```
template<typename FPTYPE>
```

```
struct ProdVirialSeAGPUExecuteFunctor
```

Public Functions

```
void operator() (FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE  
                *in_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int  
                nnei, const int ndescript, const int n_a_sel, const int n_a_shift)
```

Template Struct ProdVirialSeRGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

```
template<typename FPTYPE>
```

```
struct ProdVirialSeRGPUExecuteFunctor
```

Public Functions

```
void operator() (FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE  
                *in_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int  
                nnei, const int ndescript)
```

Template Struct TabulateCheckerGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

```
template<typename FPTYPE>
struct TabulateCheckerGPUExecuteFunctor
```

Public Functions

```
void operator() (const FPTYPE *table_info, const FPTYPE *in, int *out, const int nloc, const int
                 nnei)
```

Template Struct TabulateFusionGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

```
template<typename FPTYPE>
struct TabulateFusionGPUExecuteFunctor
```

Public Functions

```
void operator() (const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *in, const
                 FPTYPE *ff, const int nloc, const int nnei, const int last_layer_size, FPTYPE *out)
```

Template Struct TabulateFusionGradGPUExecuteFunctor

- Defined in file_source_lib_include_DeviceFunctor.h

Struct Documentation

```
template<typename FPTYPE>
struct TabulateFusionGradGPUExecuteFunctor
```

Public Functions

```
void operator()(const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *in, const
                FPTYPE *ff, const FPTYPE *dy, const int nloc, const int nnei, const int
                last_layer_size, FPTYPE *dy_dx, FPTYPE *dy_df)
```

Template Class SimulationRegion

- Defined in file_source_lib_include_SimulationRegion.h

Class Documentation

```
template<typename VALUETYPE>
class SimulationRegion
```

Public Functions

```
inline void reinitBox(const double *boxv)
inline void affineTransform(const double *affine_map)
inline void reinitOrigin(const double *orig)
inline void reinitOrigin(const std::vector<double> &orig)
void backup()
void recover()
SimulationRegion()
~SimulationRegion()
inline double *getBoxTensor()
inline const double *getBoxTensor() const
inline double *getRecBoxTensor()
inline const double *getRecBoxTensor() const
inline double *getBoxOrigin()
inline const double *getBoxOrigin() const
inline double getVolume() const
inline void toFaceDistance(double *dd) const
inline void phys2Inter(double *i_v, const VALUETYPE *p_v) const
inline void inter2Phys(VALUETYPE *p_v, const double *i_v) const
inline bool isPeriodic(const int dim) const
```

```

inline double *getShiftVec(const int index = 0)

inline const double *getShiftVec(const int index = 0) const

inline int getShiftIndex(const int *idx) const

inline int getNullShiftIndex() const

inline void shiftCoord(const int *idx, VALUETYPE &x, VALUETYPE &y, VALUETYPE &z) const

inline void diffNearestNeighbor(const VALUETYPE *r0, const VALUETYPE *r1, VALUETYPE
                                *phys) const

inline virtual void diffNearestNeighbor(const VALUETYPE x0, const VALUETYPE y0, const
                                         VALUETYPE z0, const VALUETYPE x1, const
                                         VALUETYPE y1, const VALUETYPE z1, VALUETYPE
                                         &dx, VALUETYPE &dy, VALUETYPE &dz) const

inline virtual void diffNearestNeighbor(const VALUETYPE x0, const VALUETYPE y0, const
                                         VALUETYPE z0, const VALUETYPE x1, const
                                         VALUETYPE y1, const VALUETYPE z1, VALUETYPE
                                         &dx, VALUETYPE &dy, VALUETYPE &dz, int &shift_x,
                                         int &shift_y, int &shift_z) const

inline virtual void diffNearestNeighbor(const VALUETYPE x0, const VALUETYPE y0, const
                                         VALUETYPE z0, const VALUETYPE x1, const
                                         VALUETYPE y1, const VALUETYPE z1, VALUETYPE
                                         &dx, VALUETYPE &dy, VALUETYPE &dz,
                                         VALUETYPE &shift_x, VALUETYPE &shift_y,
                                         VALUETYPE &shift_z) const

```

Public Static Functions

```

static inline int compactIndex(const int *idx)

static inline int getNumbShiftVec()

static inline int getShiftVecTotalSize()

```

Protected Functions

```

void computeShiftVec()

inline double *getInterShiftVec(const int index = 0)

inline const double *getInterShiftVec(const int index = 0) const

```

Protected Attributes

```
double shift_vec[shift_vec_size]
```

```
double inter_shift_vec[shift_vec_size]
```

Protected Static Functions

```
static inline int index3to1(const int tx, const int ty, const int tz)
```

Protected Static Attributes

```
static const int SPACENDIM = 3
```

```
static const int DBOX_XX = 1
```

```
static const int DBOX_YY = 1
```

```
static const int DBOX_ZZ = 1
```

```
static const int NBOX_XX = DBOX_XX * 2 + 1
```

```
static const int NBOX_YY = DBOX_YY * 2 + 1
```

```
static const int NBOX_ZZ = DBOX_ZZ * 2 + 1
```

```
static const int shift_info_size = NBOX_XX * NBOX_YY * NBOX_ZZ
```

```
static const int shift_vec_size = SPACENDIM * shift_info_size
```

19.3.3 Functions

Function `build_nlist(std::vector<std::vector<int>>&, std::vector<std::vector<int>>&, const std::vector<double>&, const int&, const double&, const double&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const SimulationRegion<double>&, const std::vector<int>&)`

- Defined in `file_source_lib_include_neighbor_list.h`

Function Documentation

```
void build_nlist(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const
                std::vector<double> &coord, const int &nloc, const double &rc0, const double &rc1,
                const std::vector<int> &nat_stt_, const std::vector<int> &nat_end_, const
                std::vector<int> &ext_stt_, const std::vector<int> &ext_end_, const
                SimulationRegion<double> &region, const std::vector<int> &global_grid)
```

Function `build_nlist(std::vector<std::vector<int>>&, std::vector<std::vector<int>>&, const std::vector<double>&, const double&, const double&, const std::vector<int>&, const SimulationRegion<double>&)`

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

```
void build_nlist(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const
                std::vector<double> &coord, const double &rc0, const double &rc1, const
                std::vector<int> &grid, const SimulationRegion<double> &region)
```

Function `build_nlist(std::vector<std::vector<int>>&, std::vector<std::vector<int>>&, const std::vector<double>&, const std::vector<int>&, const std::vector<int>&, const double&, const double&, const std::vector<int>&, const SimulationRegion<double>&)`

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

```
void build_nlist(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const
                std::vector<double> &coord, const std::vector<int> &sel0, const std::vector<int>
                &sel1, const double &rc0, const double &rc1, const std::vector<int> &grid, const
                SimulationRegion<double> &region)
```

Function `build_nlist(std::vector<std::vector<int>>&, std::vector<std::vector<int>>&, const std::vector<double>&, const double&, const double&, const SimulationRegion<double> *)`

- Defined in file_source_lib_include_neighbor_list.h

Function Documentation

```
void build_nlist(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const
                std::vector<double> &coord, const double &rc0_, const double &rc1_, const
                SimulationRegion<double> *region = NULL)
```

Function `compute_descriptor(std::vector<double>&, std::vector<double>&, std::vector<double>&, const std::vector<double>&, const int&, const std::vector<int>&, const SimulationRegion<double>&, const bool&, const int&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const int, const int, const int, const int)`

- Defined in file `_source_lib_include_ComputeDescriptor.h`

Function Documentation

```
inline void compute_descriptor(std::vector<double> &descript_a, std::vector<double> &descript_r,
                             std::vector<double> &rot_mat, const std::vector<double> &posi, const
                             int &ntypes, const std::vector<int> &type, const
                             SimulationRegion<double> &region, const bool &b_pbc, const int
                             &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int>
                             &fmt_nlist_r, const std::vector<int> &sec_a, const std::vector<int>
                             &sec_r, const int axis0_type, const int axis0_idx, const int axis1_type,
                             const int axis1_idx)
```

Function `compute_descriptor(std::vector<double>&, std::vector<double>&, std::vector<double>&, std::vector<double>&, std::vector<double>&, std::vector<double>&, const std::vector<double>&, const int&, const std::vector<int>&, const SimulationRegion<double>&, const bool&, const int&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const int, const int, const int, const int)`

- Defined in file `_source_lib_include_ComputeDescriptor.h`

Function Documentation

```
inline void compute_descriptor(std::vector<double> &descript_a, std::vector<double>
                             &descript_a_deriv, std::vector<double> &descript_r,
                             std::vector<double> &descript_r_deriv, std::vector<double> &rij_a,
                             std::vector<double> &rij_r, std::vector<double> &rot_mat, const
                             std::vector<double> &posi, const int &ntypes, const std::vector<int>
                             &type, const SimulationRegion<double> &region, const bool &b_pbc,
                             const int &i_idx, const std::vector<int> &fmt_nlist_a, const
                             std::vector<int> &fmt_nlist_r, const std::vector<int> &sec_a, const
                             std::vector<int> &sec_r, const int axis0_type, const int axis0_idx, const
                             int axis1_type, const int axis1_idx)
```

Function `compute_descriptor_se_a_ef_para`

- Defined in file `_source_lib_include_ComputeDescriptor.h`

Function Documentation

```
inline void compute_descriptor_se_a_ef_para(std::vector<double> &descript_a, std::vector<double>
&descript_a_deriv, std::vector<double> &rij_a, const
std::vector<double> &posi, const int &ntypes, const
std::vector<int> &type, const
SimulationRegion<double> &region, const bool &b_pbc,
const std::vector<double> &efield, const int &i_idx, const
std::vector<int> &fmt_nlist_a, const std::vector<int>
&sec_a, const double &rmin, const double &rmax)
```

Function compute_descriptor_se_a_ef_vert

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

```
inline void compute_descriptor_se_a_ef_vert(std::vector<double> &descript_a, std::vector<double>
&descript_a_deriv, std::vector<double> &rij_a, const
std::vector<double> &posi, const int &ntypes, const
std::vector<int> &type, const
SimulationRegion<double> &region, const bool &b_pbc,
const std::vector<double> &efield, const int &i_idx, const
std::vector<int> &fmt_nlist_a, const std::vector<int>
&sec_a, const double &rmin, const double &rmax)
```

Function compute_descriptor_se_a_extf

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

```
inline void compute_descriptor_se_a_extf(std::vector<double> &descript_a, std::vector<double>
&descript_a_deriv, std::vector<double> &rij_a, const
std::vector<double> &posi, const int &ntypes, const
std::vector<int> &type, const SimulationRegion<double>
&region, const bool &b_pbc, const std::vector<double>
&efield, const int &i_idx, const std::vector<int>
&fmt_nlist_a, const std::vector<int> &sec_a, const double
&rmin, const double &rmax)
```

Function compute_dRdT

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function “compute_dRdT” with arguments (double (*), const double*, const double*, const double*) in doxygen xml output for project “core” from directory: _build/core/xml/. Potential matches:

```
- void compute_dRdT(double (*dRdT)[9], const double *r1, const double *r2, const double *rot)
```

Function compute_dRdT_1

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function “compute_dRdT_1” with arguments (double (*), const double*, const double*, const double*) in doxygen xml output for project “core” from directory: _build/core/xml/. Potential matches:

```
- void compute_dRdT_1(double (*dRdT)[9], const double *r1, const double *r2, const double *rot)
```

Function compute_dRdT_2

- Defined in file_source_lib_include_ComputeDescriptor.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function “compute_dRdT_2” with arguments (double (*), const double*, const double*, const double*) in doxygen xml output for project “core” from directory: _build/core/xml/. Potential matches:

```
- void compute_dRdT_2(double (*dRdT)[9], const double *r1, const double *r2, const double *rot)
```

Function `copy_coord`

- Defined in file `_source_lib_include_neighbor_list.h`

Function Documentation

```
void copy_coord(std::vector<double> &out_c, std::vector<int> &out_t, std::vector<int> &mapping,
               std::vector<int> &ncell, std::vector<int> &ngcell, const std::vector<double> &in_c,
               const std::vector<int> &in_t, const double &rc, const SimulationRegion<double>
               &region)
```

Template Function `deepmd::build_nlist_cpu`

- Defined in file `_source_lib_include_neighbor_list.h`

Function Documentation

```
template<typename FPTYPE>
int deepmd::build_nlist_cpu(InputNlist &nlist, int *max_list_size, const FPTYPE *c_cpy, const int
                           &nloc, const int &nall, const int &mem_size, const float &rcut)
```

Template Function `deepmd::build_nlist_gpu`

- Defined in file `_source_lib_include_neighbor_list.h`

Function Documentation

```
template<typename FPTYPE>
int deepmd::build_nlist_gpu(InputNlist &nlist, int *max_list_size, int *nlist_data, const FPTYPE
                           *c_cpy, const int &nloc, const int &nall, const int &mem_size, const float
                           &rcut)
```

Template Function `deepmd::compute_cell_info`

- Defined in file `_source_lib_include_coord.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::compute_cell_info(int *cell_info, const float &rcut, const deepmd::Region<FPTYPE>
                              &region)
```

Function `deepmd::convert_nlist`

- Defined in `file_source_lib_include_neighbor_list.h`

Function Documentation

```
void deepmd::convert_nlist(InputNlist &to_nlist, std::vector<std::vector<int>> &from_nlist)
```

Construct the `InputNlist` with a two-dimensional vector.

Parameters

- **to_nlist** – `InputNlist` struct which stores the neighbor information of the core region atoms.
- **from_nlist** – Vector which stores the neighbor information of the core region atoms.

Function `deepmd::convert_nlist_gpu_device`

- Defined in `file_source_lib_include_neighbor_list.h`

Function Documentation

```
void deepmd::convert_nlist_gpu_device(InputNlist &gpu_nlist, InputNlist &cpu_nlist, int  
                                     *&gpu_memory, const int &max_nbor_size)
```

Convert the a host memory `InputNlist` to a device memory `InputNlist`.

Parameters

- **cpu_nlist** – Host memory `InputNlist` struct which stores the neighbor information of the core region atoms
- **gpu_nlist** – Device memory `InputNlist` struct which stores the neighbor information of the core region atoms
- **gpu_memory** – Device array which stores the elements of `gpu_nlist`
- **max_nbor_size** –

Template Function `deepmd::convert_to_inter_cpu`

- Defined in `file_source_lib_include_region.h`

Function Documentation

```
template<typename FPTYPE>  
void deepmd::convert_to_inter_cpu(FPTYPE *ri, const Region<FPTYPE> &region, const FPTYPE  
                                  *rp)
```

Template Function `deepmd::convert_to_inter_gpu`

- Defined in file `_source_lib_include_region.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::convert_to_inter_gpu(FPTYPE *ri, const Region<FPTYPE> &region, const FPTYPE
                                *rp)
```

Template Function `deepmd::convert_to_phys_cpu`

- Defined in file `_source_lib_include_region.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::convert_to_phys_cpu(FPTYPE *rp, const Region<FPTYPE> &region, const FPTYPE *ri)
```

Template Function `deepmd::convert_to_phys_gpu`

- Defined in file `_source_lib_include_region.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::convert_to_phys_gpu(FPTYPE *rp, const Region<FPTYPE> &region, const FPTYPE *ri)
```

Template Function `deepmd::copy_coord_cpu`

- Defined in file `_source_lib_include_coord.h`

Function Documentation

```
template<typename FPTYPE>
int deepmd::copy_coord_cpu(FPTYPE *out_c, int *out_t, int *mapping, int *nall, const FPTYPE *in_c,
                           const int *in_t, const int &nloc, const int &mem_nall, const float &rcut,
                           const deepmd::Region<FPTYPE> &region)
```

Template Function `deepmd::copy_coord_gpu`

- Defined in `file_source_lib_include_coord.h`

Function Documentation

```
template<typename FPTYPE>
int deepmd::copy_coord_gpu(FPTYPE *out_c, int *out_t, int *mapping, int *nall, int *int_data, const
                           FPTYPE *in_c, const int *in_t, const int &nloc, const int &mem_nall, const
                           int &loc_cellnum, const int &total_cellnum, const int *cell_info, const
                           deepmd::Region<FPTYPE> &region)
```

Function `deepmd::cos_switch(const double&, const double&, const double&)`

- Defined in `file_source_lib_include_switcher.h`

Function Documentation

```
inline double deepmd::cos_switch(const double &xx, const double &rmin, const double &rmax)
```

Function `deepmd::cos_switch(double&, double&, const double&, const double&, const double&)`

- Defined in `file_source_lib_include_switcher.h`

Function Documentation

```
inline void deepmd::cos_switch(double &vv, double &dd, const double &xx, const double &rmin, const
                               double &rmax)
```

Template Function `deepmd::cprod`

- Defined in `file_source_lib_include_utilities.h`

Function Documentation

```
template<typename TYPE>
inline void deepmd::cprod(const TYPE *r0, const TYPE *r1, TYPE *r2)
```


Function deepmd::cum_sum

- Defined in file_source_lib_include_utilities.h

Function Documentation

```
void deepmd::cum_sum(std::vector<int> &zsec, const std::vector<int> &n_sel)
```

Template Function deepmd::delete_device_memory

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::delete_device_memory(FPTYPE *&device)
```

Template Function deepmd::dot1

- Defined in file_source_lib_include_utilities.h

Function Documentation

```
template<typename TYPE>
inline TYPE deepmd::dot1(const TYPE *r0, const TYPE *r1)
```

Template Function deepmd::dot2

- Defined in file_source_lib_include_utilities.h

Function Documentation

```
template<typename TYPE>
inline TYPE deepmd::dot2(const TYPE *r0, const TYPE *r1)
```

Template Function deepmd::dot3

- Defined in file_source_lib_include_utilities.h

Function Documentation

```
template<typename TYPE>
inline TYPE deepmd::dot3(const TYPE *r0, const TYPE *r1)
```

Template Function deepmd::dot4

- Defined in file_source_lib_include_utilities.h

Function Documentation

```
template<typename TYPE>
inline TYPE deepmd::dot4(const TYPE *r0, const TYPE *r1)
```

Template Function deepmd::dotmv3

- Defined in file_source_lib_include_utilities.h

Function Documentation

```
template<typename TYPE>
inline void deepmd::dotmv3(TYPE *vec_o, const TYPE *tensor, const TYPE *vec_i)
```

Function deepmd::DPGetDeviceCount

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

```
inline void deepmd::DPGetDeviceCount(int &gpu_num)
```

Function deepmd::DPSetDevice

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

```
inline cudaError_t deepmd::DPSetDevice(int rank)
```

Template Function `deepmd::env_mat_a_cpu`

- Defined in file `_source_lib_include_env_mat.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::env_mat_a_cpu(std::vector<FPTYPE> &descript_a, std::vector<FPTYPE>
    &descript_a_deriv, std::vector<FPTYPE> &rij_a, const
    std::vector<FPTYPE> &posi, const std::vector<int> &type, const int
    &i_idx, const std::vector<int> &fmt_nlist, const std::vector<int> &sec,
    const float &rmin, const float &rmax)
```

Template Function `deepmd::env_mat_a_nvnmmd_quantize_cpu`

- Defined in file `_source_lib_include_env_mat_nvnmmd.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve function “`deepmd::env_mat_a_nvnmmd_quantize_cpu`” with arguments (`std::vector<FPTYPE>&`, `std::vector<FPTYPE>&`, `std::vector<FPTYPE>&`, `const std::vector<FPTYPE>&`, `const std::vector<int>&`, `const int&`, `const std::vector<int>&`, `const std::vector<int>&`, `const float&`, `const float&`, `const FPTYPE`) in doxygen xml output for project “core” from directory: `_build/core/xml/`. Potential matches:

```
- template<typename FPTYPE> void env_mat_a_nvnmmd_quantize_cpu(std::vector<FPTYPE> &descript_a,
    ↪std::vector<FPTYPE> &descript_a_deriv, std::vector<FPTYPE> &rij_a, const std::vector<FPTYPE> &
    ↪posi, const std::vector<int> &type, const int &i_idx, const std::vector<int> &fmt_nlist,
    ↪const std::vector<int> &sec, const float &rmin, const float &rmax, const FPTYPE prec[3])
```

Function `deepmd::env_mat_nbor_update`

- Defined in file `_source_lib_include_prod_env_mat.h`

Function Documentation

```
void deepmd::env_mat_nbor_update(InputNlist &inlist, InputNlist &gpu_inlist, int &max_nbor_size, int
    *&nbor_list_dev, const int *mesh, const int size)
```

Template Function `deepmd::env_mat_r_cpu`

- Defined in `file_source_lib_include_env_mat.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::env_mat_r_cpu(std::vector<FPTYPE> &descript_a, std::vector<FPTYPE>
    &descript_a_deriv, std::vector<FPTYPE> &rij_a, const
    std::vector<FPTYPE> &posi, const std::vector<int> &type, const int
    &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int>
    &sec_a, const float &rmin, const float &rmax)
```

Template Function `deepmd::ewald_recip`

- Defined in `file_source_lib_include_ewald.h`

Function Documentation

```
template<typename VALUETYPE>
void deepmd::ewald_recip(VALUETYPE &ener, std::vector<VALUETYPE> &force,
    std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE>
    &coord, const std::vector<VALUETYPE> &charge, const
    deepmd::Region<VALUETYPE> &region, const
    EwaldParameters<VALUETYPE> &param)
```

Template Function `deepmd::format_nbor_list_gpu_cuda`

- Defined in `file_source_lib_include_fmt_nlist.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::format_nbor_list_gpu_cuda(int *nlist, const FPTYPE *coord, const int *type, const
    deepmd::InputNlist &gpu_inlist, int *array_int, uint_64
    *array_longlong, const int max_nbor_size, const int nloc,
    const int nall, const float rcut, const std::vector<int> sec)
```

Template Function `deepmd::format_nlist_cpu`

- Defined in file `_source_lib_include_fmt_nlist.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::format_nlist_cpu(int *nlist, const InputNlist &in_nlist, const FPTYPE *coord, const int
                             *type, const int nloc, const int nall, const float rcut, const
                             std::vector<int> sec)
```

Function `deepmd::free_nlist_gpu_device`

- Defined in file `_source_lib_include_neighbor_list.h`

Function Documentation

```
void deepmd::free_nlist_gpu_device(InputNlist &gpu_nlist)
```

Reclaim the allocated device memory of struct `InputNlist`.

Parameters `gpu_nlist` – Device memory `InputNlist` struct which stores the neighbor information of the core region atoms

Template Function `deepmd::gelu_cpu`

- Defined in file `_source_lib_include_gelu.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::gelu_cpu(FPTYPE *out, const FPTYPE *xx, const int_64 size)
```

Template Function `deepmd::gelu_gpu_cuda`

- Defined in file `_source_lib_include_gelu.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::gelu_gpu_cuda(FPTYPE *out, const FPTYPE *xx, const int_64 size)
```

Template Function `deepmd::gelu_grad_cpu`

- Defined in file `_source_lib_include_gelu.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::gelu_grad_cpu(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const int_64 size)
```

Template Function `deepmd::gelu_grad_gpu_cuda`

- Defined in file `_source_lib_include_gelu.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::gelu_grad_gpu_cuda(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const int_64
                                size)
```

Template Function `deepmd::gelu_grad_grad_cpu`

- Defined in file `_source_lib_include_gelu.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::gelu_grad_grad_cpu(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const
                                FPTYPE *dy_2, const int_64 size)
```

Template Function `deepmd::gelu_grad_grad_gpu_cuda`

- Defined in file `_source_lib_include_gelu.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::gelu_grad_grad_gpu_cuda(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const
                                      FPTYPE *dy_2, const int_64 size)
```

Template Function `deepmd::init_region_cpu`

- Defined in `file_source_lib_include_region.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::init_region_cpu(Region<FPTYPE> &region, const FPTYPE *boxt)
```

Template Function `deepmd::invsqrt`

- Defined in `file_source_lib_include_utilities.h`

Function Documentation

```
template<typename TYPE>
inline TYPE deepmd::invsqrt(const TYPE x)
```

Specialized Template Function `deepmd::invsqrt< double >`

- Defined in `file_source_lib_include_utilities.h`

Function Documentation

```
template<>
inline double deepmd::invsqrt<double>(const double x)
```

Specialized Template Function `deepmd::invsqrt< float >`

- Defined in `file_source_lib_include_utilities.h`

Function Documentation

```
template<>
inline float deepmd::invsqrt<float>(const float x)
```

Template Function `deepmd::malloc_device_memory(FPTYPE *&, const std::vector<FPTYPE>&)`

- Defined in `file_source_lib_include_gpu_cuda.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::malloc_device_memory(FPTYPE *&device, const std::vector<FPTYPE> &host)
```

Template Function deepmd::malloc_device_memory(FPTYPE *&, const int)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::malloc_device_memory(FPTYPE *&device, const int size)
```

Template Function deepmd::malloc_device_memory(FPTYPE *&, std::vector<FPTYPE> &)

- Defined in file_source_lib_include_gpu_rocm.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::malloc_device_memory(FPTYPE *&device, std::vector<FPTYPE> &host)
```

Template	Function	deepmd::malloc_device_memory_sync(FPTYPE	*&,	const
std::vector<FPTYPE> &)				

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::malloc_device_memory_sync(FPTYPE *&device, const std::vector<FPTYPE> &host)
```

Template Function deepmd::malloc_device_memory_sync(FPTYPE *&, const FPTYPE *, const int)

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::malloc_device_memory_sync(FPTYPE *&device, const FPTYPE *host, const int size)
```

Template Function `deepmd::malloc_device_memory_sync(FPTYPE *&, std::vector<FPTYPE>&)`

- Defined in `file_source_lib_include_gpu_rocm.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::malloc_device_memory_sync(FPTYPE *&device, std::vector<FPTYPE> &host)
```

Template Function `deepmd::map_aparam_cpu`

- Defined in `file_source_lib_include_map_aparam.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::map_aparam_cpu(FPTYPE *output, const FPTYPE *aparam, const int *nlist, const int
                             &nloc, const int &nnei, const int &numb_aparam)
```

Function `deepmd::max_numneigh`

- Defined in `file_source_lib_include_neighbor_list.h`

Function Documentation

```
int deepmd::max_numneigh(const InputNlist &to_nlist)
```

Compute the max number of neighbors within the core region atoms.

Parameters `to_nlist` – `InputNlist` struct which stores the neighbor information of the core region atoms.

Return values `max` – number of neighbors

Returns integer

Template Function `deepmd::memcpy_device_to_host(const FPTYPE *, std::vector<FPTYPE>&)`

- Defined in file `_source_lib_include_gpu_cuda.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::memcpy_device_to_host(const FPTYPE *device, std::vector<FPTYPE> &host)
```

Template Function `deepmd::memcpy_device_to_host(const FPTYPE *, FPTYPE *, const int)`

- Defined in file `_source_lib_include_gpu_cuda.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::memcpy_device_to_host(const FPTYPE *device, FPTYPE *host, const int size)
```

Template Function `deepmd::memcpy_device_to_host(FPTYPE *, std::vector<FPTYPE>&)`

- Defined in file `_source_lib_include_gpu_rocm.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::memcpy_device_to_host(FPTYPE *device, std::vector<FPTYPE> &host)
```

Template Function `deepmd::memcpy_host_to_device(FPTYPE *, const std::vector<FPTYPE>&)`

- Defined in file `_source_lib_include_gpu_cuda.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::memcpy_host_to_device(FPTYPE *device, const std::vector<FPTYPE> &host)
```

Template Function `deepmd::memcpy_host_to_device(FPTYPE *, const FPTYPE *, const int)`

- Defined in file `_source_lib_include_gpu_cuda.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::memcpy_host_to_device(FPTYPE *device, const FPTYPE *host, const int size)
```

Template Function deepmd::memcpy_host_to_device(FPTYPE *, std::vector<FPTYPE> &)

- Defined in file_source_lib_include_gpu_rocm.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::memcpy_host_to_device(FPTYPE *device, std::vector<FPTYPE> &host)
```

Template Function deepmd::memset_device_memory

- Defined in file_source_lib_include_gpu_cuda.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::memset_device_memory(FPTYPE *device, const int var, const int size)
```

Template Function deepmd::normalize_coord_cpu

- Defined in file_source_lib_include_coord.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::normalize_coord_cpu(FPTYPE *coord, const int natom, const
                                deepmd::Region<FPTYPE> &region)
```

Template Function deepmd::normalize_coord_gpu

- Defined in file_source_lib_include_coord.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::normalize_coord_gpu(FPTYPE *coord, const int natom, const
                                deepmd::Region<FPTYPE> &region)
```

Template Function deepmd::pair_tab_cpu

- Defined in file_source_lib_include_pair_tab.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::pair_tab_cpu(FPTYPE *energy, FPTYPE *force, FPTYPE *virial, const double
                          *table_info, const double *table_data, const FPTYPE *rij, const FPTYPE
                          *scale, const int *type, const int *nlist, const int *natoms, const
                          std::vector<int> &sel_a, const std::vector<int> &sel_r)
```

Template Function deepmd::prod_env_mat_a_cpu

- Defined in file_source_lib_include_prod_env_mat.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_env_mat_a_cpu(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const
                                FPTYPE *coord, const int *type, const InputNlist &inlist, const int
                                max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int
                                nloc, const int nall, const float rcut, const float rcut_smth, const
                                std::vector<int> sec, const int *f_type = NULL)
```

Template Function deepmd::prod_env_mat_a_gpu_cuda

- Defined in file_source_lib_include_prod_env_mat.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_env_mat_a_gpu_cuda(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist,
                                      const FPTYPE *coord, const int *type, const InputNlist
                                      &gpu_inlist, int *array_int, unsigned long long
                                      *array_longlong, const int max_nbor_size, const FPTYPE *avg,
                                      const FPTYPE *std, const int nloc, const int nall, const float
                                      rcut, const float rcut_smth, const std::vector<int> sec, const int
                                      *f_type = NULL)
```

Template Function `deepmd::prod_env_mat_a_nvnmmd_quantize_cpu`

- Defined in file `_source_lib_include_prod_env_mat_nvnmmd.h`

Function Documentation

Warning: doxygenfunction: Unable to resolve function “`deepmd::prod_env_mat_a_nvnmmd_quantize_cpu`” with arguments (FPTYPE*, FPTYPE*, FPTYPE*, int*, const FPTYPE*, const int*, const InputNlist&, const int, const FPTYPE*, const FPTYPE*, const int, const int, const float, const float, const std::vector<int>, const FPTYPE) in doxygen xml output for project “core” from directory: `_build/core/xml/`. Potential matches:

```
- template<typename FPTYPE> void prod_env_mat_a_nvnmmd_quantize_cpu(FPTYPE *em, FPTYPE *em_deriv,
  ↳ FPTYPE *rij, int *nlist, const FPTYPE *coord, const int *type, const InputNlist &inlist,
  ↳ const int max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int nloc, const int nall,
  ↳ const float rcut, const float rcut_smth, const std::vector<int> sec, const FPTYPE prec[3])
```

Template Function `deepmd::prod_env_mat_r_cpu`

- Defined in file `_source_lib_include_prod_env_mat.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_env_mat_r_cpu(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const
                                FPTYPE *coord, const int *type, const InputNlist &inlist, const int
                                max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int
                                nloc, const int nall, const float rcut, const float rcut_smth, const
                                std::vector<int> sec)
```

Template Function `deepmd::prod_env_mat_r_gpu_cuda`

- Defined in file `_source_lib_include_prod_env_mat.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_env_mat_r_gpu_cuda(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist,
                                     const FPTYPE *coord, const int *type, const InputNlist
                                     &gpu_inlist, int *array_int, unsigned long long
                                     *array_longlong, const int max_nbor_size, const FPTYPE *avg,
                                     const FPTYPE *std, const int nloc, const int nall, const float
                                     rcut, const float rcut_smth, const std::vector<int> sec)
```

Template Function `deepmd::prod_force_a_cpu`

- Defined in file `_source_lib_include_prod_force.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_force_a_cpu(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE *in_deriv,
                             const int *nlist, const int nloc, const int nall, const int nnei, const int
                             start_index = 0)
```

Template Function `deepmd::prod_force_a_gpu_cuda`

- Defined in file `_source_lib_include_prod_force.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_force_a_gpu_cuda(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE
                                   *in_deriv, const int *nlist, const int nloc, const int nall, const int
                                   nnei)
```

Template Function `deepmd::prod_force_grad_a_cpu`

- Defined in file `_source_lib_include_prod_force_grad.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_force_grad_a_cpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                   *env_deriv, const int *nlist, const int nloc, const int nnei)
```

Template Function `deepmd::prod_force_grad_a_gpu_cuda`

- Defined in file `_source_lib_include_prod_force_grad.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_force_grad_a_gpu_cuda(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                         *env_deriv, const int *nlist, const int nloc, const int nnei)
```

Template Function `deepmd::prod_force_grad_r_cpu`

- Defined in file `source_lib_include_prod_force_grad.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_force_grad_r_cpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                   *env_deriv, const int *nlist, const int nloc, const int nnei)
```

Template Function `deepmd::prod_force_grad_r_gpu_cuda`

- Defined in file `source_lib_include_prod_force_grad.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_force_grad_r_gpu_cuda(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                         *env_deriv, const int *nlist, const int nloc, const int nnei)
```

Template Function `deepmd::prod_force_r_cpu`

- Defined in file `source_lib_include_prod_force.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_force_r_cpu(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE *in_deriv,
                              const int *nlist, const int nloc, const int nall, const int nnei)
```

Template Function `deepmd::prod_force_r_gpu_cuda`

- Defined in file `source_lib_include_prod_force.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_force_r_gpu_cuda(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE
                                   *in_deriv, const int *nlist, const int nloc, const int nall, const int
                                   nnei)
```

Template Function `deepmd::prod_virial_a_cpu`

- Defined in file `_source_lib_include_prod_virial.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_virial_a_cpu(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv,
                              const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const
                              int nloc, const int nall, const int nnei)
```

Template Function `deepmd::prod_virial_a_gpu_cuda`

- Defined in file `_source_lib_include_prod_virial.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_virial_a_gpu_cuda(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE
                                     *net_deriv, const FPTYPE *env_deriv, const FPTYPE *rij, const
                                     int *nlist, const int nloc, const int nall, const int nnei)
```

Template Function `deepmd::prod_virial_grad_a_cpu`

- Defined in file `_source_lib_include_prod_virial_grad.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_virial_grad_a_cpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                     *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc,
                                     const int nnei)
```

Template Function `deepmd::prod_virial_grad_a_gpu_cuda`

- Defined in file `_source_lib_include_prod_virial_grad.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_virial_grad_a_gpu_cuda(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                           *env_deriv, const FPTYPE *rij, const int *nlist, const int
                                           nloc, const int nnei)
```


Template Function `deepmd::prod_virial_grad_r_cpu`

- Defined in file `source_lib_include_prod_virial_grad.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_virial_grad_r_cpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                   *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc,
                                   const int nnei)
```

Template Function `deepmd::prod_virial_grad_r_gpu_cuda`

- Defined in file `source_lib_include_prod_virial_grad.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_virial_grad_r_gpu_cuda(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                         *env_deriv, const FPTYPE *rij, const int *nlist, const int
                                         nloc, const int nnei)
```

Template Function `deepmd::prod_virial_r_cpu`

- Defined in file `source_lib_include_prod_virial.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_virial_r_cpu(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv,
                              const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const
                              int nloc, const int nall, const int nnei)
```

Template Function `deepmd::prod_virial_r_gpu_cuda`

- Defined in file `source_lib_include_prod_virial.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_virial_r_gpu_cuda(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE
                                    *net_deriv, const FPTYPE *env_deriv, const FPTYPE *rij, const
                                    int *nlist, const int nloc, const int nall, const int nnei)
```

Template Function `deepmd::soft_min_switch_cpu`

- Defined in file `_source_lib_include_soft_min_switch.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::soft_min_switch_cpu(FPTYPE *sw_value, FPTYPE *sw_deriv, const FPTYPE *rij, const
                                int *nlist, const int &nloc, const int &nnei, const FPTYPE &alpha,
                                const FPTYPE &rmin, const FPTYPE &rmax)
```

Template Function `deepmd::soft_min_switch_force_cpu`

- Defined in file `_source_lib_include_soft_min_switch_force.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::soft_min_switch_force_cpu(FPTYPE *force, const FPTYPE *du, const FPTYPE
                                       *sw_deriv, const int *nlist, const int nloc, const int nall, const
                                       int nnei)
```

Template Function `deepmd::soft_min_switch_force_grad_cpu`

- Defined in file `_source_lib_include_soft_min_switch_force_grad.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::soft_min_switch_force_grad_cpu(FPTYPE *grad_net, const FPTYPE *grad, const
                                             FPTYPE *sw_deriv, const int *nlist, const int nloc,
                                             const int nnei)
```

Template Function `deepmd::soft_min_switch_virial_cpu`

- Defined in file `_source_lib_include_soft_min_switch_virial.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::soft_min_switch_virial_cpu(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE
                                         *du, const FPTYPE *sw_deriv, const FPTYPE *rij, const int
                                         *nlist, const int nloc, const int nall, const int nnei)
```

Template Function `deepmd::soft_min_switch_virial_grad_cpu`

- Defined in file `_source_lib_include_soft_min_switch_virial_grad.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::soft_min_switch_virial_grad_cpu(FPTYPE *grad_net, const FPTYPE *grad, const
                                             FPTYPE *sw_deriv, const FPTYPE *rij, const int
                                             *nlist, const int nloc, const int nnei)
```

Function `deepmd::spline3_switch`

- Defined in file `_source_lib_include_switcher.h`

Function Documentation

```
inline void deepmd::spline3_switch(double &vv, double &dd, const double &xx, const double &rmin,
                                   const double &rmax)
```

Template Function `deepmd::spline5_switch`

- Defined in file `_source_lib_include_switcher.h`

Function Documentation

```
template<typename FPTYPE>
inline void deepmd::spline5_switch(FPTYPE &vv, FPTYPE &dd, const FPTYPE &xx, const float
                                   &rmin, const float &rmax)
```

Template Function `deepmd::tabulate_fusion_se_a_cpu`

- Defined in file `_source_lib_include_tabulate.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_a_cpu(FPTYPE *out, const FPTYPE *table, const FPTYPE
                                       *table_info, const FPTYPE *em_x, const FPTYPE *em, const
                                       int nloc, const int nnei, const int last_layer_size)
```

Template Function `deepmd::tabulate_fusion_se_a_gpu_cuda`

- Defined in `file_source_lib_include_tabulate.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_a_gpu_cuda(FPTYPE *out, const FPTYPE *table, const FPTYPE
                                           *table_info, const FPTYPE *em_x, const FPTYPE *em,
                                           const int nloc, const int nnei, const int last_layer_size)
```

Template Function `deepmd::tabulate_fusion_se_a_grad_cpu`

- Defined in `file_source_lib_include_tabulate.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_a_grad_cpu(FPTYPE *dy_dem_x, FPTYPE *dy_dem, const
                                           FPTYPE *table, const FPTYPE *table_info, const
                                           FPTYPE *em_x, const FPTYPE *em, const FPTYPE
                                           *dy, const int nloc, const int nnei, const int
                                           last_layer_size)
```

Template Function `deepmd::tabulate_fusion_se_a_grad_gpu_cuda`

- Defined in `file_source_lib_include_tabulate.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_a_grad_gpu_cuda(FPTYPE *dy_dem_x, FPTYPE *dy_dem, const
                                                 FPTYPE *table, const FPTYPE *table_info, const
                                                 FPTYPE *em_x, const FPTYPE *em, const
                                                 FPTYPE *dy, const int nloc, const int nnei, const
                                                 int last_layer_size)
```

Template Function `deepmd::tabulate_fusion_se_a_grad_grad_cpu`

- Defined in `file_source_lib_include_tabulate.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_a_grad_grad_cpu(FPTYPE *dz_dy, const FPTYPE *table, const
FPTYPE *table_info, const FPTYPE *em_x, const
FPTYPE *em, const FPTYPE *dz_dy_dem_x,
const FPTYPE *dz_dy_dem, const int nloc, const
int nnei, const int last_layer_size)
```

Template Function deepmd::tabulate_fusion_se_a_grad_grad_gpu_cuda

- Defined in file_source_lib_include_tabulate.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_a_grad_grad_gpu_cuda(FPTYPE *dz_dy, const FPTYPE *table,
const FPTYPE *table_info, const FPTYPE
*em_x, const FPTYPE *em, const FPTYPE
*dz_dy_dem_x, const FPTYPE *dz_dy_dem,
const int nloc, const int nnei, const int
last_layer_size)
```

Template Function deepmd::tabulate_fusion_se_r_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_cpu(FPTYPE *out, const FPTYPE *table, const FPTYPE
*table_info, const FPTYPE *em, const int nloc, const int nnei,
const int last_layer_size)
```

Template Function deepmd::tabulate_fusion_se_r_gpu_cuda

- Defined in file_source_lib_include_tabulate.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_gpu_cuda(FPTYPE *out, const FPTYPE *table, const FPTYPE
*table_info, const FPTYPE *em, const int nloc, const int
nnei, const int last_layer_size)
```

Template Function `deepmd::tabulate_fusion_se_r_grad_cpu`

- Defined in `file_source_lib_include_tabulate.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_grad_cpu(FPTYPE *dy_dem, const FPTYPE *table, const
                                           FPTYPE *table_info, const FPTYPE *em, const
                                           FPTYPE *dy, const int nloc, const int nnei, const int
                                           last_layer_size)
```

Template Function `deepmd::tabulate_fusion_se_r_grad_gpu_cuda`

- Defined in `file_source_lib_include_tabulate.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_grad_gpu_cuda(FPTYPE *dy_dem, const FPTYPE *table, const
                                                  FPTYPE *table_info, const FPTYPE *em, const
                                                  FPTYPE *dy, const int nloc, const int nnei, const
                                                  int last_layer_size)
```

Template Function `deepmd::tabulate_fusion_se_r_grad_grad_cpu`

- Defined in `file_source_lib_include_tabulate.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_grad_grad_cpu(FPTYPE *dz_dy, const FPTYPE *table, const
                                                  FPTYPE *table_info, const FPTYPE *em, const
                                                  FPTYPE *dz_dy_dem, const int nloc, const int
                                                  nnei, const int last_layer_size)
```

Template Function `deepmd::tabulate_fusion_se_r_grad_grad_gpu_cuda`

- Defined in `file_source_lib_include_tabulate.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_grad_grad_gpu_cuda(FPTYPE *dz_dy, const FPTYPE *table,
                                                    const FPTYPE *table_info, const FPTYPE
                                                    *em, const FPTYPE *dz_dy_dem, const int
                                                    nloc, const int nnei, const int last_layer_size)
```

Template Function deepmd::tabulate_fusion_se_t_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_cpu(FPTYPE *out, const FPTYPE *table, const FPTYPE
                                     *table_info, const FPTYPE *em_x, const FPTYPE *em, const
                                     int nloc, const int nnei_i, const int nnei_j, const int
                                     last_layer_size)
```

Template Function deepmd::tabulate_fusion_se_t_gpu_cuda

- Defined in file_source_lib_include_tabulate.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_gpu_cuda(FPTYPE *out, const FPTYPE *table, const FPTYPE
                                           *table_info, const FPTYPE *em_x, const FPTYPE *em,
                                           const int nloc, const int nnei_i, const int nnei_j, const int
                                           last_layer_size)
```

Template Function deepmd::tabulate_fusion_se_t_grad_cpu

- Defined in file_source_lib_include_tabulate.h

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_grad_cpu(FPTYPE *dy_dem_x, FPTYPE *dy_dem, const
                                           FPTYPE *table, const FPTYPE *table_info, const
                                           FPTYPE *em_x, const FPTYPE *em, const FPTYPE
                                           *dy, const int nloc, const int nnei_i, const int nnei_j,
                                           const int last_layer_size)
```

Template Function `deepmd::tabulate_fusion_se_t_grad_gpu_cuda`

- Defined in file `source_lib_include_tabulate.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_grad_gpu_cuda(FPTYPE *dy_dem_x, FPTYPE *dy_dem, const
FPTYPE *table, const FPTYPE *table_info, const
FPTYPE *em_x, const FPTYPE *em, const
FPTYPE *dy, const int nloc, const int nnei_i, const
int nnei_j, const int last_layer_size)
```

Template Function `deepmd::tabulate_fusion_se_t_grad_grad_cpu`

- Defined in file `source_lib_include_tabulate.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_grad_grad_cpu(FPTYPE *dz_dy, const FPTYPE *table, const
FPTYPE *table_info, const FPTYPE *em_x, const
FPTYPE *em, const FPTYPE *dz_dy_dem_x,
const FPTYPE *dz_dy_dem, const int nloc, const
int nnei_i, const int nnei_j, const int
last_layer_size)
```

Template Function `deepmd::tabulate_fusion_se_t_grad_grad_gpu_cuda`

- Defined in file `source_lib_include_tabulate.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_grad_grad_gpu_cuda(FPTYPE *dz_dy, const FPTYPE *table,
const FPTYPE *table_info, const FPTYPE
*em_x, const FPTYPE *em, const FPTYPE
*dz_dy_dem_x, const FPTYPE *dz_dy_dem,
const int nloc, const int nnei_i, const int
nnei_j, const int last_layer_size)
```


Template Function `deepmd::test_encoding_decoding_nbor_info_gpu_cuda`

- Defined in file `_source_lib_include_fmt_nlist.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::test_encoding_decoding_nbor_info_gpu_cuda(uint_64 *key, int *out_type, int
                                                         *out_index, const int *in_type, const
                                                         FPTYPE *in_dist, const int *in_index,
                                                         const int size_of_array)
```

Function `deepmd::use_nei_info_cpu`

- Defined in file `_source_lib_include_neighbor_list.h`

Function Documentation

```
void deepmd::use_nei_info_cpu(int *nlist, int *ntype, bool *nmask, const int *type, const int *nlist_map,
                              const int nloc, const int nnei, const int ntypes, const bool b_nlist_map)
```

Function `deepmd::use_nei_info_gpu`

- Defined in file `_source_lib_include_neighbor_list.h`

Function Documentation

```
void deepmd::use_nei_info_gpu(int *nlist, int *ntype, bool *nmask, const int *type, const int *nlist_map,
                              const int nloc, const int nnei, const int ntypes, const bool b_nlist_map)
```

Function `deepmd::use_nlist_map`

- Defined in file `_source_lib_include_neighbor_list.h`

Function Documentation

```
void deepmd::use_nlist_map(int *nlist, const int *nlist_map, const int nloc, const int nnei)
```

Template Function `deepmd::volume_cpu`

- Defined in `file_source_lib_include_region.h`

Function Documentation

```
template<typename FPTYPE>
FPTYPE deepmd::volume_cpu(const Region<FPTYPE> &region)
```

Template Function `deepmd::volume_gpu`

- Defined in `file_source_lib_include_region.h`

Function Documentation

```
template<typename FPTYPE>
void deepmd::volume_gpu(FPTYPE *volume, const Region<FPTYPE> &region)
```

Function `DPAssert(cudaError_t, const char *, int, bool)`

- Defined in `file_source_lib_include_gpu_cuda.h`

Function Documentation

```
inline void DPAssert(cudaError_t code, const char *file, int line, bool abort = true)
```

Function `DPAssert(hipError_t, const char *, int, bool)`

- Defined in `file_source_lib_include_gpu_rocm.h`

Function Documentation

```
inline void DPAssert(hipError_t code, const char *file, int line, bool abort = true)
```

Function `env_mat_a`

- Defined in `file_source_lib_include_env_mat.h`

Function Documentation

```
void env_mat_a(std::vector<double> &descript_a, std::vector<double> &descript_a_deriv,
               std::vector<double> &rij_a, const std::vector<double> &posi, const int &ntypes, const
               std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc,
               const int &i_idx, const std::vector<int> &fmt_nlist, const std::vector<int> &sec, const
               double &rmin, const double &rmax)
```

Function `env_mat_r`

- Defined in `file_source_lib_include_env_mat.h`

Function Documentation

```
void env_mat_r(std::vector<double> &descript_r, std::vector<double> &descript_r_deriv,
               std::vector<double> &rij_r, const std::vector<double> &posi, const int &ntypes, const
               std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc,
               const int &i_idx, const std::vector<int> &fmt_nlist, const std::vector<int> &sec, const
               double &rmin, const double &rmax)
```

Template Function `format_nlist_i_cpu`

- Defined in `file_source_lib_include_fmt_nlist.h`

Function Documentation

```
template<typename FPTYPE>
int format_nlist_i_cpu(std::vector<int> &fmt_nei_idx_a, const std::vector<FPTYPE> &posi, const
                      std::vector<int> &type, const int &i_idx, const std::vector<int> &nei_idx_a,
                      const float &rcut, const std::vector<int> &sec_a)
```

Function `format_nlist_i_fill_a`

- Defined in `file_source_lib_include_fmt_nlist.h`

Function Documentation

```
int format_nlist_i_fill_a(std::vector<int> &fmt_nei_idx_a, std::vector<int> &fmt_nei_idx_r, const
                          std::vector<double> &posi, const int &ntypes, const std::vector<int> &type,
                          const SimulationRegion<double> &region, const bool &b_pbc, const int
                          &i_idx, const std::vector<int> &nei_idx_a, const std::vector<int>
                          &nei_idx_r, const double &rcut, const std::vector<int> &sec_a, const
                          std::vector<int> &sec_r)
```

Function `nborAssert(cudaError_t, const char *, int, bool)`

- Defined in `file_source_lib_include_gpu_cuda.h`

Function Documentation

```
inline void nborAssert(cudaError_t code, const char *file, int line, bool abort = true)
```

Function `nborAssert(hipError_t, const char *, int, bool)`

- Defined in `file_source_lib_include_gpu_rocm.h`

Function Documentation

```
inline void nborAssert(hipError_t code, const char *file, int line, bool abort = true)
```

Function `omp_get_num_threads`

- Defined in `file_source_lib_include_ewald.h`

Function Documentation

```
int omp_get_num_threads()
```

Function `omp_get_thread_num`

- Defined in `file_source_lib_include_ewald.h`

Function Documentation

```
int omp_get_thread_num()
```

19.3.4 Variables

Variable `deepmd::ElectrostaticConversion`

- Defined in `file_source_lib_include_ewald.h`

Variable Documentation

`const double deepmd::ElectrostaticConversion = 14.39964535475696995031`

19.3.5 Defines

Define DPErrcheck

- Defined in file `_source_lib_include_gpu_cuda.h`

Define Documentation

`DPErrcheck(res)`

Define DPErrcheck

- Defined in file `_source_lib_include_gpu_rocm.h`

Define Documentation

`DPErrcheck(res)`

Define GPU_MAX_NBOR_SIZE

- Defined in file `_source_lib_include_gpu_cuda.h`

Define Documentation

`GPU_MAX_NBOR_SIZE`

Define GPU_MAX_NBOR_SIZE

- Defined in file `_source_lib_include_gpu_rocm.h`

Define Documentation

`GPU_MAX_NBOR_SIZE`

Define MOASPNDIM

- Defined in file_source_lib_include_SimulationRegion.h

Define Documentation

MOASPNDIM

Define nborErrcheck

- Defined in file_source_lib_include_gpu_cuda.h

Define Documentation

nborErrcheck(res)

Define nborErrcheck

- Defined in file_source_lib_include_gpu_rocm.h

Define Documentation

nborErrcheck(res)

Define SQRT_2_PI

- Defined in file_source_lib_include_device.h

Define Documentation

SQRT_2_PI

Define TPB

- Defined in file_source_lib_include_device.h

Define Documentation

TPB

19.3.6 Typedefs

Typedef int_64

- Defined in file_source_lib_include_device.h

Typedef Documentation

typedef long long **int_64**

Typedef uint_64

- Defined in file_source_lib_include_device.h

Typedef Documentation

typedef unsigned long long **uint_64**

LICENSE

The project DeePMD-kit is licensed under [GNU LGPLv3.0](#).

AUTHORS AND CREDITS

21.1 Package Contributors

- AnguseZhang
- Chenxing Luo
- Chun Cai
- Davide Tisi
- Denghui Lu
- Duo
- GeiduanLiu
- Han Wang
- Jia-Xin Zhu
- Jiequn Han
- Jingchao Zhang
- Jinzhe Zeng
- Linfeng Zhang
- LiuGroupHNU
- Lu
- Marián Rynik
- Nick Lin
- Rhys Goodall
- Shaochen Shi
- Xia, Yu
- YWolfee
- Ye Ding
- Yifan Li黎一帆
- Yingze Wang
- Yixiao Chen

- Zeyu Li
- ZhengdQin
- ZiyaoLi
- baohan
- bwang-ecnu
- deepmodeling
- denghuilu
- haidi
- hlyang
- hsulab
- iProzd
- jxxiaoshaoye
- liangadam
- likefallwind
- marian-code
- njzjz
- pkulzy
- readthedocs-assistant
- tuoping
- wsyxbcl
- ziyao

21.2 Other Credits

- Zhang ZiXuan for designing the Deepmodeling logo.
- Everyone on the Deepmodeling mailing list for contributing to many discussions and decisions!
- 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, 139
deepmd.calculator, 282
deepmd.cluster, 143
deepmd.cluster.local, 143
deepmd.cluster.slurm, 143
deepmd.common, 284
deepmd.descriptor, 144
deepmd.descriptor.descriptor, 144
deepmd.descriptor.hybrid, 149
deepmd.descriptor.loc_frame, 153
deepmd.descriptor.se, 156
deepmd.descriptor.se_a, 158
deepmd.descriptor.se_a_ebd, 162
deepmd.descriptor.se_a_ef, 164
deepmd.descriptor.se_atten, 168
deepmd.descriptor.se_r, 171
deepmd.descriptor.se_t, 174
deepmd.entrypoints, 177
deepmd.entrypoints.compress, 180
deepmd.entrypoints.config, 181
deepmd.entrypoints.convert, 181
deepmd.entrypoints.doc, 181
deepmd.entrypoints.freeze, 181
deepmd.entrypoints.main, 182
deepmd.entrypoints.neighbor_stat, 182
deepmd.entrypoints.test, 183
deepmd.entrypoints.train, 183
deepmd.entrypoints.transfer, 184
deepmd.env, 289
deepmd.env.op_grads_module, 332
deepmd.env.op_module, 291
deepmd.fit, 184
deepmd.fit.dipole, 184
deepmd.fit.ener, 186
deepmd.fit.fitting, 188
deepmd.fit.polar, 189
deepmd.fit.wfc, 193
deepmd.infer, 193
deepmd.infer.data_modifier, 205
deepmd.infer.deep_dipole, 207
deepmd.infer.deep_eval, 208
deepmd.infer.deep_polar, 209
deepmd.infer.deep_pot, 212
deepmd.infer.deep_tensor, 215
deepmd.infer.deep_wfc, 217
deepmd.infer.ewald_recip, 218
deepmd.infer.model_devi, 219
deepmd.loggers, 220
deepmd.loggers.loggers, 221
deepmd.loss, 222
deepmd.loss.ener, 222
deepmd.loss.loss, 224
deepmd.loss.tensor, 225
deepmd.model, 226
deepmd.model.ener, 226
deepmd.model.model, 227
deepmd.model.model_stat, 228
deepmd.model.tensor, 228
deepmd.nvnmd, 231
deepmd.nvnmd.data, 231
deepmd.nvnmd.data.data, 232
deepmd.nvnmd.descriptor, 232
deepmd.nvnmd.descriptor.se_a, 232
deepmd.nvnmd.entrypoints, 232
deepmd.nvnmd.entrypoints.freeze, 235
deepmd.nvnmd.entrypoints.mapt, 235
deepmd.nvnmd.entrypoints.train, 237
deepmd.nvnmd.entrypoints.wrap, 237
deepmd.nvnmd.fit, 238
deepmd.nvnmd.fit.ener, 239
deepmd.nvnmd.utils, 239
deepmd.nvnmd.utils.argcheck, 242
deepmd.nvnmd.utils.config, 242
deepmd.nvnmd.utils.encode, 244
deepmd.nvnmd.utils.fio, 246
deepmd.nvnmd.utils.network, 248
deepmd.nvnmd.utils.op, 249
deepmd.nvnmd.utils.weight, 249
deepmd.op, 250
deepmd.train, 250
deepmd.train.run_options, 250
deepmd.train.trainer, 251
deepmd.utils, 252

`deepmd.utils.argcheck`, 252
`deepmd.utils.batch_size`, 254
`deepmd.utils.compat`, 255
`deepmd.utils.convert`, 256
`deepmd.utils.data`, 258
`deepmd.utils.data_system`, 262
`deepmd.utils.errors`, 265
`deepmd.utils.graph`, 265
`deepmd.utils.learning_rate`, 269
`deepmd.utils.neighbor_stat`, 270
`deepmd.utils.network`, 271
`deepmd.utils.pair_tab`, 272
`deepmd.utils.parallel_op`, 273
`deepmd.utils.path`, 274
`deepmd.utils.plugin`, 277
`deepmd.utils.random`, 278
`deepmd.utils.sess`, 279
`deepmd.utils.tabulate`, 279
`deepmd.utils.type_embed`, 280
`deepmd.utils.weight_avg`, 282

A

activation_function: (Argument), 63
 model/descriptor[se_a_tpe]/activation_function (Argument), 58
 model/descriptor[se_atten]/activation_function (Argument), 62
 model/descriptor[se_e2_a]/activation_function (Argument), 54
 model/descriptor[se_e2_r]/activation_function (Argument), 60
 model/descriptor[se_e3]/activation_function (Argument), 56
 model/fitting_net[dipole]/activation_function (Argument), 65
 model/fitting_net[ener]/activation_function (Argument), 64
 model/fitting_net[polar]/activation_function (Argument), 66
 model/type_embedding/activation_function (Argument), 52
 add() (deepmd.common.ClassArg method), 284
 add() (deepmd.utils.data.DeepmdData method), 260
 add() (deepmd.utils.data_system.DeepmdDataSystem method), 263
 add_data_requirement() (in module deepmd.common), 285
 add_dict() (deepmd.utils.data_system.DeepmdDataSystem method), 264
 add_path() (in module deepmd.nvnmd.entrypoints.train), 237
 ArgsPlugin (class in deepmd.utils.argcheck), 252
 atom_ener:
 model/fitting_net[ener]/atom_ener (Argument), 65
 attn:
 model/descriptor[se_atten]/attn (Argument), 63
 attn_dotr:
 model/descriptor[se_atten]/attn_dotr (Argument), 63
 attn_layer:
 model/descriptor[se_atten]/attn_layer (Argument), 63
 attn_mask:
 model/descriptor[se_atten]/attn_mask (Argument), 63
 auto_prob:
 training/training_data/auto_prob (Argument), 72
 training/validation_data/auto_prob (Argument), 73
 AutoBatchSize (class in deepmd.utils.batch_size), 254
 avg() (deepmd.utils.data.DeepmdData method), 260
 axis_neuron:
 model/descriptor[se_a_tpe]/axis_neuron (Argument), 58
 model/descriptor[se_atten]/axis_neuron (Argument), 62
 model/descriptor[se_e2_a]/axis_neuron (Argument), 54
 axis_rule:
 model/descriptor[loc_frame]/axis_rule (Argument), 53

B

batch_size:
 training/training_data/batch_size (Argument), 72
 training/validation_data/batch_size (Argument), 73
 bin2hex() (deepmd.nvnmd.utils.Encode method), 239
 bin2hex() (deepmd.nvnmd.utils.encode.Encode method), 245
 bin2hex_str() (deepmd.nvnmd.utils.Encode method), 239
 bin2hex_str() (deepmd.nvnmd.utils.encode.Encode method), 245
 build() (deepmd.descriptor.descriptor.Descriptor method), 144
 build() (deepmd.descriptor.hybrid.DescriptHybrid method), 150
 build() (deepmd.descriptor.loc_frame.DescriptLocFrame

method), 154

build() (deepmd.descriptor.se_a.DescriptSeA method), 160

build() (deepmd.descriptor.se_a_ebd.DescriptSeAEbd method), 163

build() (deepmd.descriptor.se_a_ef.DescriptSeAEf method), 165

build() (deepmd.descriptor.se_a_ef.DescriptSeAEfLower method), 167

build() (deepmd.descriptor.se_atten.DescriptSeAtten method), 169

build() (deepmd.descriptor.se_r.DescriptSeR method), 172

build() (deepmd.descriptor.se_t.DescriptSeT method), 175

build() (deepmd.fit.dipole.DipoleFittingSeA method), 185

build() (deepmd.fit.ener.EnerFitting method), 187

build() (deepmd.fit.polar.GlobalPolarFittingSeA method), 190

build() (deepmd.fit.polar.PolarFittingLocFrame method), 191

build() (deepmd.fit.polar.PolarFittingSeA method), 192

build() (deepmd.fit.wfc.WFCFitting method), 193

build() (deepmd.loss.ener.EnerDipoleLoss method), 222

build() (deepmd.loss.ener.EnerStdLoss method), 223

build() (deepmd.loss.loss.Loss method), 224

build() (deepmd.loss.tensor.TensorLoss method), 225

build() (deepmd.model.ener.EnerModel method), 227

build() (deepmd.model.tensor.TensorModel method), 230

build() (deepmd.train.trainer.DPTrainer method), 251

build() (deepmd.utils.learning_rate.LearningRateExp method), 269

build() (deepmd.utils.tabulate.DPTabulate method), 280

build() (deepmd.utils.type_embed.TypeEmbedNet method), 281

build_davg_dstd() (in module deepmd.nvnmd.descriptor.se_a), 232

build_dG_ds() (deepmd.nvnmd.entrypoints.mapt.MapTable method), 236

build_dG_ds() (deepmd.nvnmd.entrypoints.MapTable method), 233

build_ds_dr() (deepmd.nvnmd.entrypoints.mapt.MapTable method), 236

build_ds_dr() (deepmd.nvnmd.entrypoints.MapTable method), 233

build_fv_graph() (deepmd.DipoleChargeModifier method), 142

build_fv_graph() (deepmd.infer.data_modifier.DipoleChargeModifier method), 206

build_fv_graph() (deepmd.infer.DipoleChargeModifier method), 203

build_map() (deepmd.nvnmd.entrypoints.mapt.MapTable method), 236

build_map() (deepmd.nvnmd.entrypoints.MapTable method), 233

build_nlist (C++ function), 377

build_op_descriptor() (in module deepmd.nvnmd.descriptor.se_a), 232

build_r2s() (deepmd.nvnmd.entrypoints.mapt.MapTable method), 236

build_r2s() (deepmd.nvnmd.entrypoints.MapTable method), 233

build_r2s_r2ds() (deepmd.nvnmd.entrypoints.mapt.MapTable method), 236

build_r2s_r2ds() (deepmd.nvnmd.entrypoints.MapTable method), 234

build_s2G() (deepmd.nvnmd.entrypoints.mapt.MapTable method), 236

build_s2G() (deepmd.nvnmd.entrypoints.MapTable method), 234

build_s2G_s2dG() (deepmd.nvnmd.entrypoints.mapt.MapTable method), 236

build_s2G_s2dG() (deepmd.nvnmd.entrypoints.MapTable method), 234

C

calc_model_devi() (in module deepmd.infer), 204

calc_model_devi() (in module deepmd.infer.model_devi), 219

calc_model_devi_e() (in module deepmd.infer.model_devi), 219

calc_model_devi_f() (in module deepmd.infer.model_devi), 219

calc_model_devi_v() (in module deepmd.infer.model_devi), 219

calculate() (deepmd.calculator.DP method), 284

cast_precision() (in module deepmd.common), 285

check_batch_size() (deepmd.utils.data.DataSets method), 258

check_batch_size() (deepmd.utils.data.DeepmdData method), 260

check_dec() (deepmd.nvnmd.utils.Encode method), 239

check_dec() (deepmd.nvnmd.utils.encode.Encode method), 245

check_test_size() (deepmd.utils.data.DataSets method), 258

`check_test_size()` (deepmd.utils.data.DeepmdData method), 260
`check_type_map_consistency()` (deepmd.utils.data_system.DataSystem method), 262
`choice()` (in module deepmd.utils.random), 278
`ClassArg` (class in deepmd.common), 284
`compress()` (in module deepmd.entrypoints), 177
`compress()` (in module deepmd.entrypoints.compress), 180
`compress:`
 `model/compress` (Argument), 68
`compute_descriptor` (C++ function), 378
`compute_descriptor_se_a_ef_para` (C++ function), 379
`compute_descriptor_se_a_ef_vert` (C++ function), 379
`compute_descriptor_se_a_extf` (C++ function), 379
`compute_energy_shift()` (deepmd.utils.data_system.DataSystem method), 262
`compute_energy_shift()` (deepmd.utils.data_system.DeepmdDataSystem method), 264
`compute_input_stats()` (deepmd.descriptor.descriptor.Descriptor method), 145
`compute_input_stats()` (deepmd.descriptor.hybrid.DescriptHybrid method), 150
`compute_input_stats()` (deepmd.descriptor.loc_frame.DescriptLocFrame method), 154
`compute_input_stats()` (deepmd.descriptor.se_a.DescriptSeA method), 160
`compute_input_stats()` (deepmd.descriptor.se_a_ef.DescriptSeAEf method), 165
`compute_input_stats()` (deepmd.descriptor.se_a_ef.DescriptSeAEfLower method), 168
`compute_input_stats()` (deepmd.descriptor.se_atten.DescriptSeAtten method), 170
`compute_input_stats()` (deepmd.descriptor.se_r.DescriptSeR method), 172
`compute_input_stats()` (deepmd.descriptor.se_t.DescriptSeT method), 175
`compute_input_stats()` (deepmd.fit.ener.EnerFitting method), 187
`compute_input_stats()` (deepmd.fit.polar.PolarFittingSeA method), 192
`compute_output_stats()` (deepmd.fit.ener.EnerFitting method), 187
`compute_prec:`
 `training/mixed_precision/compute_prec` (Argument), 74
`config()` (in module deepmd.entrypoints), 177
`config()` (in module deepmd.entrypoints.config), 181
`config_file:`
 `nvnmmd/config_file` (Argument), 76
`convert()` (in module deepmd.entrypoints), 178
`convert()` (in module deepmd.entrypoints.convert), 181
`convert_012_to_21()` (in module deepmd.utils.convert), 256
`convert_10_to_21()` (in module deepmd.utils.convert), 256
`convert_12_to_21()` (in module deepmd.utils.convert), 256
`convert_13_to_21()` (in module deepmd.utils.convert), 257
`convert_20_to_21()` (in module deepmd.utils.convert), 257
`convert_dp012_to_dp10()` (in module deepmd.utils.convert), 257
`convert_dp10_to_dp11()` (in module deepmd.utils.convert), 257
`convert_dp12_to_dp13()` (in module deepmd.utils.convert), 257
`convert_dp13_to_dp20()` (in module deepmd.utils.convert), 257
`convert_dp20_to_dp21()` (in module deepmd.utils.convert), 257
`convert_input_v0_v1()` (in module deepmd.utils.compat), 255
`convert_input_v1_v2()` (in module deepmd.utils.compat), 256
`convert_pb_to_pbtxt()` (in module deepmd.utils.convert), 257
`convert_pbtxt_to_pb()` (in module deepmd.utils.convert), 257
`copy_coord` (C++ function), 381
`create_file_path()` (deepmd.nvnmmd.utils.fio.Fio method), 246

D

`data_stat()` (deepmd.model.ener.EnerModel method), 227

```

data_stat()    (deepmd.model.tensor.TensorModel
               method), 230
data_stat_nbatch:
    model/data_stat_nbatch (Argument), 51
data_stat_protect:
    model/data_stat_protect (Argument), 51
DataSets (class in deepmd.utils.data), 258
DataSystem (class in deepmd.utils.data_system), 262
dec2bin() (deepmd.nvnmd.utils.Encode method),
          239
dec2bin()    (deepmd.nvnmd.utils.encode.Encode
               method), 245
decay_steps:
    learning_rate[exp]/decay_steps
    (Argument), 69
DeepDipole (class in deepmd.infer), 193
DeepDipole (class in deepmd.infer.deep_dipole), 207
DeepEval (class in deepmd), 139
DeepEval (class in deepmd.infer), 194
DeepEval (class in deepmd.infer.deep_eval), 208
DeepGlobalPolar (class in deepmd.infer), 196
DeepGlobalPolar (class in deepmd.infer.deep_polar), 209
deepmd
    module, 139
deepmd.calculator
    module, 282
deepmd.cluster
    module, 143
deepmd.cluster.local
    module, 143
deepmd.cluster.slurm
    module, 143
deepmd.common
    module, 284
deepmd.descriptor
    module, 144
deepmd.descriptor.descriptor
    module, 144
deepmd.descriptor.hybrid
    module, 149
deepmd.descriptor.loc_frame
    module, 153
deepmd.descriptor.se
    module, 156
deepmd.descriptor.se_a
    module, 158
deepmd.descriptor.se_a_ebd
    module, 162
deepmd.descriptor.se_a_ef
    module, 164
deepmd.descriptor.se_atten
    module, 168
deepmd.descriptor.se_r
    module, 171
deepmd.descriptor.se_t
    module, 174
deepmd.entrypoints
    module, 177
deepmd.entrypoints.compress
    module, 180
deepmd.entrypoints.config
    module, 181
deepmd.entrypoints.convert
    module, 181
deepmd.entrypoints.doc
    module, 181
deepmd.entrypoints.freeze
    module, 181
deepmd.entrypoints.main
    module, 182
deepmd.entrypoints.neighbor_stat
    module, 182
deepmd.entrypoints.test
    module, 183
deepmd.entrypoints.train
    module, 183
deepmd.entrypoints.transfer
    module, 184
deepmd.env
    module, 289
deepmd.env.op_grads_module
    module, 332
deepmd.env.op_module
    module, 291
deepmd.fit
    module, 184
deepmd.fit.dipole
    module, 184
deepmd.fit.ener
    module, 186
deepmd.fit.fitting
    module, 188
deepmd.fit.polar
    module, 189
deepmd.fit.wfc
    module, 193
deepmd.infer
    module, 193
deepmd.infer.data_modifier
    module, 205
deepmd.infer.deep_dipole
    module, 207
deepmd.infer.deep_eval
    module, 208
deepmd.infer.deep_polar
    module, 209
deepmd.infer.deep_pot
    module, 209

```

- module, 212
- deepmd.infer.deep_tensor
 - module, 215
- deepmd.infer.deep_wfc
 - module, 217
- deepmd.infer.ewald_recip
 - module, 218
- deepmd.infer.model_devi
 - module, 219
- deepmd.loggers
 - module, 220
- deepmd.loggers.loggers
 - module, 221
- deepmd.loss
 - module, 222
- deepmd.loss.ener
 - module, 222
- deepmd.loss.loss
 - module, 224
- deepmd.loss.tensor
 - module, 225
- deepmd.model
 - module, 226
- deepmd.model.ener
 - module, 226
- deepmd.model.model
 - module, 227
- deepmd.model.model_stat
 - module, 228
- deepmd.model.tensor
 - module, 228
- deepmd.nvnmd
 - module, 231
- deepmd.nvnmd.data
 - module, 231
- deepmd.nvnmd.data.data
 - module, 232
- deepmd.nvnmd.descriptor
 - module, 232
- deepmd.nvnmd.descriptor.se_a
 - module, 232
- deepmd.nvnmd.entrypoints
 - module, 232
- deepmd.nvnmd.entrypoints.freeze
 - module, 235
- deepmd.nvnmd.entrypoints.mapt
 - module, 235
- deepmd.nvnmd.entrypoints.train
 - module, 237
- deepmd.nvnmd.entrypoints.wrap
 - module, 237
- deepmd.nvnmd.fit
 - module, 238
- deepmd.nvnmd.fit.ener
 - module, 239
- deepmd.nvnmd.utils
 - module, 239
- deepmd.nvnmd.utils.argcheck
 - module, 242
- deepmd.nvnmd.utils.config
 - module, 242
- deepmd.nvnmd.utils.encode
 - module, 244
- deepmd.nvnmd.utils.fio
 - module, 246
- deepmd.nvnmd.utils.network
 - module, 248
- deepmd.nvnmd.utils.op
 - module, 249
- deepmd.nvnmd.utils.weight
 - module, 249
- deepmd.op
 - module, 250
- deepmd.train
 - module, 250
- deepmd.train.run_options
 - module, 250
- deepmd.train.trainer
 - module, 251
- deepmd.utils
 - module, 252
- deepmd.utils.argcheck
 - module, 252
- deepmd.utils.batch_size
 - module, 254
- deepmd.utils.compat
 - module, 255
- deepmd.utils.convert
 - module, 256
- deepmd.utils.data
 - module, 258
- deepmd.utils.data_system
 - module, 262
- deepmd.utils.errors
 - module, 265
- deepmd.utils.graph
 - module, 265
- deepmd.utils.learning_rate
 - module, 269
- deepmd.utils.neighbor_stat
 - module, 270
- deepmd.utils.network
 - module, 271
- deepmd.utils.pair_tab
 - module, 272
- deepmd.utils.parallel_op
 - module, 273
- deepmd.utils.path

module, 274
 deepmd.utils.plugin
 module, 277
 deepmd.utils.random
 module, 278
 deepmd.utils.sess
 module, 279
 deepmd.utils.tabulate
 module, 279
 deepmd.utils.type_embed
 module, 280
 deepmd.utils.weight_avg
 module, 282
 deepmd::AtomMap (C++ class), 343
 deepmd::AtomMap::AtomMap (C++ function), 343
 deepmd::AtomMap::backward (C++ function), 343
 deepmd::AtomMap::forward (C++ function), 343
 deepmd::AtomMap::get_bkw_map (C++ function), 343
 deepmd::AtomMap::get_fwd_map (C++ function), 343
 deepmd::AtomMap::get_type (C++ function), 343
 deepmd::build_nlist_cpu (C++ function), 381
 deepmd::build_nlist_gpu (C++ function), 381
 deepmd::check_status (C++ function), 355
 deepmd::compute_cell_info (C++ function), 381
 deepmd::convert_nlist (C++ function), 382
 deepmd::convert_nlist_gpu_device (C++ function), 382
 deepmd::convert_pbtxt_to_pb (C++ function), 355
 deepmd::convert_to_inter_cpu (C++ function), 382
 deepmd::convert_to_inter_gpu (C++ function), 383
 deepmd::convert_to_phys_cpu (C++ function), 383
 deepmd::convert_to_phys_gpu (C++ function), 383
 deepmd::copy_coord_cpu (C++ function), 383
 deepmd::copy_coord_gpu (C++ function), 384
 deepmd::cos_switch (C++ function), 384
 deepmd::cprod (C++ function), 384
 deepmd::cum_sum (C++ function), 385
 deepmd::deepmd_exception (C++ struct), 341, 367
 deepmd::deepmd_exception::deepmd_exception (C++ function), 367
 deepmd::deepmd_exception_oom (C++ struct), 368
 deepmd::deepmd_exception_oom::deepmd_exception_oom (C++ function), 368
 deepmd::DeepPot (C++ class), 344
 deepmd::DeepPot::~DeepPot (C++ function), 344
 deepmd::DeepPot::compute (C++ function), 344--346
 deepmd::DeepPot::cutoff (C++ function), 347
 deepmd::DeepPot::DeepPot (C++ function), 344
 deepmd::DeepPot::dim_aparam (C++ function), 347
 deepmd::DeepPot::dim_fparam (C++ function), 347
 deepmd::DeepPot::get_type_map (C++ function), 347
 deepmd::DeepPot::init (C++ function), 344
 deepmd::DeepPot::numb_types (C++ function), 347
 deepmd::DeepPot::print_summary (C++ function), 344
 deepmd::DeepPotModelDevi (C++ class), 347
 deepmd::DeepPotModelDevi::~DeepPotModelDevi (C++ function), 347
 deepmd::DeepPotModelDevi::compute (C++ function), 348
 deepmd::DeepPotModelDevi::compute_avg (C++ function), 349, 350
 deepmd::DeepPotModelDevi::compute_relative_std (C++ function), 350
 deepmd::DeepPotModelDevi::compute_relative_std_f (C++ function), 350
 deepmd::DeepPotModelDevi::compute_std (C++ function), 350
 deepmd::DeepPotModelDevi::compute_std_e (C++ function), 350
 deepmd::DeepPotModelDevi::compute_std_f (C++ function), 350
 deepmd::DeepPotModelDevi::cutoff (C++ function), 349
 deepmd::DeepPotModelDevi::DeepPotModelDevi (C++ function), 347
 deepmd::DeepPotModelDevi::dim_aparam (C++ function), 349
 deepmd::DeepPotModelDevi::dim_fparam (C++ function), 349
 deepmd::DeepPotModelDevi::init (C++ function), 348
 deepmd::DeepPotModelDevi::numb_types (C++ function), 349
 deepmd::DeepTensor (C++ class), 351
 deepmd::DeepTensor::~DeepTensor (C++ function), 351
 deepmd::DeepTensor::compute (C++ function), 351--353
 deepmd::DeepTensor::cutoff (C++ function), 354
 deepmd::DeepTensor::DeepTensor (C++ function), 351
 deepmd::DeepTensor::init (C++ function), 351
 deepmd::DeepTensor::numb_types (C++ function), 354
 deepmd::DeepTensor::output_dim (C++ function),

354
 deepmd::DeepTensor::print_summary (C++ function), 351
 deepmd::DeepTensor::sel_types (C++ function), 354
 deepmd::delete_device_memory (C++ function), 385
 deepmd::DipoleChargeModifier (C++ class), 354
 deepmd::DipoleChargeModifier::~DipoleChargeModifier (C++ function), 354
 deepmd::DipoleChargeModifier::compute (C++ function), 354
 deepmd::DipoleChargeModifier::cutoff (C++ function), 354
 deepmd::DipoleChargeModifier::DipoleChargeModifier (C++ function), 354
 deepmd::DipoleChargeModifier::init (C++ function), 354
 deepmd::DipoleChargeModifier::numb_types (C++ function), 354
 deepmd::DipoleChargeModifier::print_summary (C++ function), 354
 deepmd::DipoleChargeModifier::sel_types (C++ function), 354
 deepmd::dot1 (C++ function), 385
 deepmd::dot2 (C++ function), 385
 deepmd::dot3 (C++ function), 386
 deepmd::dot4 (C++ function), 386
 deepmd::dotmv3 (C++ function), 386
 deepmd::DPGetDeviceCount (C++ function), 386
 deepmd::DPSetDevice (C++ function), 386
 deepmd::ElectrostaticConversion (C++ member), 413
 deepmd::ENERGYTYPE (C++ type), 360
 deepmd::env_mat_a_cpu (C++ function), 387
 deepmd::env_mat_nbor_update (C++ function), 387
 deepmd::env_mat_r_cpu (C++ function), 388
 deepmd::ewald_rec (C++ function), 388
 deepmd::EwaldParameters (C++ struct), 368
 deepmd::EwaldParameters::beta (C++ member), 368
 deepmd::EwaldParameters::rcut (C++ member), 368
 deepmd::EwaldParameters::spacing (C++ member), 368
 deepmd::format_nbor_list_gpu_cuda (C++ function), 388
 deepmd::format_nlist_cpu (C++ function), 389
 deepmd::free_nlist_gpu_device (C++ function), 389
 deepmd::gelu_cpu (C++ function), 389
 deepmd::gelu_gpu_cuda (C++ function), 389
 deepmd::gelu_grad_cpu (C++ function), 390
 deepmd::gelu_grad_gpu_cuda (C++ function), 390
 deepmd::gelu_grad_grad_cpu (C++ function), 390
 deepmd::gelu_grad_grad_gpu_cuda (C++ function), 390
 deepmd::get_env_nthreads (C++ function), 355
 deepmd::init_region_cpu (C++ function), 391
 deepmd::InputNlist (C++ struct), 368
 deepmd::InputNlist::~InputNlist (C++ function), 369
 deepmd::InputNlist::firstneigh (C++ member), 369
 deepmd::InputNlist::ilist (C++ member), 369
 deepmd::InputNlist::InputNlist (C++ function), 369
 deepmd::InputNlist::inum (C++ member), 369
 deepmd::InputNlist::numneigh (C++ member), 369
 deepmd::invsqrt (C++ function), 391
 deepmd::invsqrt<double> (C++ function), 391
 deepmd::invsqrt<float> (C++ function), 391
 deepmd::load_op_library (C++ function), 356
 deepmd::malloc_device_memory (C++ function), 392
 deepmd::malloc_device_memory_sync (C++ function), 392, 393
 deepmd::map_aparam_cpu (C++ function), 393
 deepmd::max_numneigh (C++ function), 393
 deepmd::memcpy_device_to_host (C++ function), 394
 deepmd::memcpy_host_to_device (C++ function), 394, 395
 deepmd::memset_device_memory (C++ function), 395
 deepmd::model_compatible (C++ function), 356
 deepmd::name_prefix (C++ function), 356
 deepmd::NeighborListData (C++ struct), 342
 deepmd::NeighborListData::copy_from_nlist (C++ function), 342
 deepmd::NeighborListData::firstneigh (C++ member), 342
 deepmd::NeighborListData::ilist (C++ member), 342
 deepmd::NeighborListData::jlist (C++ member), 342
 deepmd::NeighborListData::make_inlist (C++ function), 342
 deepmd::NeighborListData::numneigh (C++ member), 342
 deepmd::NeighborListData::shuffle (C++ function), 342
 deepmd::NeighborListData::shuffle_exclude_empty (C++ function), 342
 deepmd::normalize_coord_cpu (C++ function), 395

deepmd::normalize_coord_gpu (C++ function), 396
 deepmd::pair_tab_cpu (C++ function), 396
 deepmd::prod_env_mat_a_cpu (C++ function), 396
 deepmd::prod_env_mat_a_gpu_cuda (C++ function), 396
 deepmd::prod_env_mat_r_cpu (C++ function), 397
 deepmd::prod_env_mat_r_gpu_cuda (C++ function), 397
 deepmd::prod_force_a_cpu (C++ function), 398
 deepmd::prod_force_a_gpu_cuda (C++ function), 398
 deepmd::prod_force_grad_a_cpu (C++ function), 398
 deepmd::prod_force_grad_a_gpu_cuda (C++ function), 398
 deepmd::prod_force_grad_r_cpu (C++ function), 399
 deepmd::prod_force_grad_r_gpu_cuda (C++ function), 399
 deepmd::prod_force_r_cpu (C++ function), 399
 deepmd::prod_force_r_gpu_cuda (C++ function), 399
 deepmd::prod_virial_a_cpu (C++ function), 400
 deepmd::prod_virial_a_gpu_cuda (C++ function), 400
 deepmd::prod_virial_grad_a_cpu (C++ function), 400
 deepmd::prod_virial_grad_a_gpu_cuda (C++ function), 400
 deepmd::prod_virial_grad_r_cpu (C++ function), 401
 deepmd::prod_virial_grad_r_gpu_cuda (C++ function), 401
 deepmd::prod_virial_r_cpu (C++ function), 401
 deepmd::prod_virial_r_gpu_cuda (C++ function), 401
 deepmd::read_file_to_string (C++ function), 356
 deepmd::Region (C++ struct), 369
 deepmd::Region::~Region (C++ function), 369
 deepmd::Region::boxt (C++ member), 369
 deepmd::Region::rec_boxt (C++ member), 369
 deepmd::Region::Region (C++ function), 369
 deepmd::select_by_type (C++ function), 357
 deepmd::select_map (C++ function), 357
 deepmd::select_map_inv (C++ function), 358
 deepmd::select_real_atoms (C++ function), 358
 deepmd::session_get_scalar (C++ function), 358
 deepmd::session_get_vector (C++ function), 359
 deepmd::session_input_tensors (C++ function), 359
 deepmd::soft_min_switch_cpu (C++ function), 402
 deepmd::soft_min_switch_force_cpu (C++ function), 402
 deepmd::soft_min_switch_force_grad_cpu (C++ function), 402
 deepmd::soft_min_switch_virial_cpu (C++ function), 402
 deepmd::soft_min_switch_virial_grad_cpu (C++ function), 403
 deepmd::spline3_switch (C++ function), 403
 deepmd::spline5_switch (C++ function), 403
 deepmd::STRINGTYPE (C++ type), 360
 deepmd::tabulate_fusion_se_a_cpu (C++ function), 403
 deepmd::tabulate_fusion_se_a_gpu_cuda (C++ function), 404
 deepmd::tabulate_fusion_se_a_grad_cpu (C++ function), 404
 deepmd::tabulate_fusion_se_a_grad_gpu_cuda (C++ function), 404
 deepmd::tabulate_fusion_se_a_grad_grad_cpu (C++ function), 405
 deepmd::tabulate_fusion_se_a_grad_grad_gpu_cuda (C++ function), 405
 deepmd::tabulate_fusion_se_r_cpu (C++ function), 405
 deepmd::tabulate_fusion_se_r_gpu_cuda (C++ function), 405
 deepmd::tabulate_fusion_se_r_grad_cpu (C++ function), 406
 deepmd::tabulate_fusion_se_r_grad_gpu_cuda (C++ function), 406
 deepmd::tabulate_fusion_se_r_grad_grad_cpu (C++ function), 406
 deepmd::tabulate_fusion_se_r_grad_grad_gpu_cuda (C++ function), 407
 deepmd::tabulate_fusion_se_t_cpu (C++ function), 407
 deepmd::tabulate_fusion_se_t_gpu_cuda (C++ function), 407
 deepmd::tabulate_fusion_se_t_grad_cpu (C++ function), 407
 deepmd::tabulate_fusion_se_t_grad_gpu_cuda (C++ function), 408
 deepmd::tabulate_fusion_se_t_grad_grad_cpu (C++ function), 408
 deepmd::tabulate_fusion_se_t_grad_grad_gpu_cuda (C++ function), 408
 deepmd::test_encoding_decoding_nbor_info_gpu_cuda (C++ function), 409
 deepmd::tf_exception (C++ struct), 343
 deepmd::tf_exception::tf_exception (C++ function), 343
 deepmd::use_nei_info_cpu (C++ function), 409
 deepmd::use_nei_info_gpu (C++ function), 409

deepmd::use_nlist_map (C++ function), 409
 deepmd::VALUETYPE (C++ type), 360
 deepmd::volume_cpu (C++ function), 410
 deepmd::volume_gpu (C++ function), 410
 DeepmdData (class in deepmd.utils.data), 259
 DeepmdDataSystem (class in deepmd.utils.data_system), 263
 DeepPolar (class in deepmd.infer), 197
 DeepPolar (class in deepmd.infer.deep_polar), 210
 DeepPot (class in deepmd.infer), 198
 DeepPot (class in deepmd.infer.deep_pot), 212
 DeepPotential() (in module deepmd), 140
 DeepPotential() (in module deepmd.infer), 201
 DeepTensor (class in deepmd.infer.deep_tensor), 215
 DeepWFC (class in deepmd.infer), 201
 DeepWFC (class in deepmd.infer.deep_wfc), 217
 deprecate_numb_test() (in module deepmd.utils.compat), 256
 Descriptor (class in deepmd.descriptor.descriptor), 144
 descriptor:
 model/descriptor (Argument), 52
 Descript() (in module deepmd.env.op_module), 291
 descript() (in module deepmd.env.op_module), 311
 descript2r4() (in module deepmd.nvnmd.descriptor.se_a), 232
 descript_hybrid_args() (in module deepmd.utils.argcheck), 253
 descript_local_frame_args() (in module deepmd.utils.argcheck), 253
 descript_norot() (in module deepmd.env.op_module), 312
 descript_se_a() (in module deepmd.env.op_module), 312
 descript_se_a_args() (in module deepmd.utils.argcheck), 253
 descript_se_a_ef() (in module deepmd.env.op_module), 313
 descript_se_a_ef_para() (in module deepmd.env.op_module), 313
 descript_se_a_ef_vert() (in module deepmd.env.op_module), 314
 descript_se_a_tpe_args() (in module deepmd.utils.argcheck), 253
 descript_se_atten_args() (in module deepmd.utils.argcheck), 253
 descript_se_r() (in module deepmd.env.op_module), 315
 descript_se_r_args() (in module deepmd.utils.argcheck), 253
 descript_se_t_args() (in module deepmd.utils.argcheck), 253
 descript_variant_type_args() (in module deepmd.utils.argcheck), 253
 DescriptHybrid (class in deepmd.descriptor.hybrid), 149
 DescriptLocFrame (class in deepmd.descriptor.loc_frame), 153
 DescriptNorot() (in module deepmd.env.op_module), 291
 DescriptSe (class in deepmd.descriptor.se), 156
 DescriptSeA (class in deepmd.descriptor.se_a), 158
 DescriptSeA() (in module deepmd.env.op_module), 292
 DescriptSeAEbd (class in deepmd.descriptor.se_a_ebd), 162
 DescriptSeAEf (class in deepmd.descriptor.se_a_ef), 164
 DescriptSeAEf() (in module deepmd.env.op_module), 293
 DescriptSeAEfLower (class in deepmd.descriptor.se_a_ef), 167
 DescriptSeAEfPara() (in module deepmd.env.op_module), 293
 DescriptSeAEfVert() (in module deepmd.env.op_module), 294
 DescriptSeAtten (class in deepmd.descriptor.se_atten), 168
 DescriptSeR (class in deepmd.descriptor.se_r), 171
 DescriptSeR() (in module deepmd.env.op_module), 294
 DescriptSeRGPUExecuteFunctor (C++ struct), 370
 DescriptSeRGPUExecuteFunctor::operator() (C++ function), 370
 DescriptSeT (class in deepmd.descriptor.se_t), 174
 DipoleChargeModifier (class in deepmd), 141
 DipoleChargeModifier (class in deepmd.infer), 202
 DipoleChargeModifier (class in deepmd.infer.data_modifier), 205
 DipoleFittingSeA (class in deepmd.fit.dipole), 184
 DipoleModel (class in deepmd.model.tensor), 228
 disp_file:
 training/disp_file (Argument), 75
 disp_freq:
 training/disp_freq (Argument), 75
 disp_message() (deepmd.nvnmd.utils.config.NvnmdConfig method), 243
 disp_training:
 training/disp_training (Argument), 75
 doc_train_input() (in module deepmd.entrypoints), 178
 doc_train_input() (in module deepmd.entrypoints.doc), 181
 DP (class in deepmd.calculator), 282
 DPAssert (C++ function), 410
 DPErrcheck (C macro), 413
 DPH5Path (class in deepmd.utils.path), 274
 DPOSPath (class in deepmd.utils.path), 275

DPPPath (class in deepmd.utils.path), 276
 DPTabulate (class in deepmd.utils.tabulate), 279
 DPTrainer (class in deepmd.train.trainer), 251

E

embed_atom_type() (in module deepmd.utils.type_embed), 281
 embedding_net() (in module deepmd.utils.network), 271
 embedding_net_rand_seed_shift() (in module deepmd.utils.network), 272
 enable:
 nvnmd/enable (Argument), 76
 enable_atom_ener_coeff:
 loss[ener]/enable_atom_ener_coeff (Argument), 71
 enable_compression()
 (deepmd.descriptor.descriptor.Descriptor method), 146
 enable_compression()
 (deepmd.descriptor.hybrid.DescriptHybrid method), 151
 enable_compression()
 (deepmd.descriptor.se_a.DescriptSeA method), 160
 enable_compression()
 (deepmd.descriptor.se_r.DescriptSeR method), 173
 enable_compression()
 (deepmd.descriptor.se_t.DescriptSeT method), 176
 enable_compression()
 (deepmd.fit.ener.EnerFitting method), 188
 enable_mixed_precision()
 (deepmd.descriptor.descriptor.Descriptor method), 146
 enable_mixed_precision()
 (deepmd.descriptor.hybrid.DescriptHybrid method), 151
 enable_mixed_precision()
 (deepmd.descriptor.se_a.DescriptSeA method), 161
 enable_mixed_precision()
 (deepmd.fit.dipole.DipoleFittingSeA method), 185
 enable_mixed_precision()
 (deepmd.fit.ener.EnerFitting method), 188
 enable_mixed_precision()
 (deepmd.fit.polar.GlobalPolarFittingSeA method), 190
 enable_mixed_precision()
 (deepmd.fit.polar.PolarFittingSeA method), 192
 enable_profiler:
 training/enable_profiler (Argument), 75
 Encode (class in deepmd.nvnmd.utils), 239
 Encode (class in deepmd.nvnmd.utils.encode), 244
 EnerDipoleLoss (class in deepmd.loss.ener), 222
 EnerFitting (class in deepmd.fit.ener), 186
 EnerModel (class in deepmd.model.ener), 226
 EnerStdLoss (class in deepmd.loss.ener), 223
 env_mat_a (C++ function), 411
 env_mat_r (C++ function), 411
 eval() (deepmd.DipoleChargeModifier method), 142
 eval() (deepmd.infer.data_modifier.DipoleChargeModifier method), 206
 eval() (deepmd.infer.deep_polar.DeepGlobalPolar method), 210
 eval() (deepmd.infer.deep_pot.DeepPot method), 213
 eval() (deepmd.infer.deep_tensor.DeepTensor method), 215
 eval() (deepmd.infer.DeepGlobalPolar method), 196
 eval() (deepmd.infer.DeepPot method), 199
 eval() (deepmd.infer.DipoleChargeModifier method), 203
 eval() (deepmd.infer.ewald_rec.EwaldRecp method), 218
 eval() (deepmd.infer.EwaldRecp method), 204
 eval() (deepmd.loss.ener.EnerDipoleLoss method), 222
 eval() (deepmd.loss.ener.EnerStdLoss method), 223
 eval() (deepmd.loss.loss.Loss method), 224
 eval() (deepmd.loss.tensor.TensorLoss method), 225
 eval_descriptor() (deepmd.infer.deep_pot.DeepPot method), 214
 eval_descriptor() (deepmd.infer.DeepPot method), 200
 eval_full() (deepmd.infer.deep_tensor.DeepTensor method), 216
 ewald_beta:
 model/modifier[dipole_charge]/ewald_beta (Argument), 67
 ewald_h:
 model/modifier[dipole_charge]/ewald_h (Argument), 67
 ewald_rec() (in module deepmd.env.op_module), 315
 EwaldRecp (class in deepmd.infer), 203
 EwaldRecp (class in deepmd.infer.ewald_rec), 218
 EwaldRecp() (in module deepmd.env.op_module), 295
 exclude_types:
 model/descriptor[se_a_tpe]/exclude_types (Argument), 58
 model/descriptor[se_atten]/exclude_types

(Argument), 62
 model/descriptor[se_e2_a]/exclude_types
 (Argument), 55
 model/descriptor[se_e2_r]/exclude_types
 (Argument), 60
 execute() (deepmd.utils.batch_size.AutoBatchSize
 method), 255
 execute_all() (deepmd.utils.batch_size.AutoBatchSize
 method), 255
 exits() (deepmd.nvnmd.utils.fio.Fio method), 247
 expand_sys_str() (in module deepmd.common),
 286
 extend_bin() (deepmd.nvnmd.utils.Encode
 method), 239
 extend_bin() (deepmd.nvnmd.utils.encode.Encode
 method), 245
 extend_hex() (deepmd.nvnmd.utils.Encode
 method), 240
 extend_hex() (deepmd.nvnmd.utils.encode.Encode
 method), 245
 extend_list() (deepmd.nvnmd.utils.Encode
 method), 240
 extend_list() (deepmd.nvnmd.utils.encode.Encode
 method), 245
F
 filter_GR2D() (in module
 deepmd.nvnmd.descriptor.se_a), 232
 filter_lower_R42GR() (in module
 deepmd.nvnmd.descriptor.se_a), 232
 filter_tensorVariableList() (in module
 deepmd.nvnmd.entrypoints.freeze), 235
 Fio (class in deepmd.nvnmd.utils.fio), 246
 FioBin (class in deepmd.nvnmd.utils), 240
 FioBin (class in deepmd.nvnmd.utils.fio), 247
 FioDic (class in deepmd.nvnmd.utils), 241
 FioDic (class in deepmd.nvnmd.utils.fio), 247
 FioJsonDic (class in deepmd.nvnmd.utils.fio), 247
 FioNpyDic (class in deepmd.nvnmd.utils.fio), 248
 FioTxt (class in deepmd.nvnmd.utils), 241
 FioTxt (class in deepmd.nvnmd.utils.fio), 248
 fit_diag:
 model/fitting_net[polar]/fit_diag
 (Argument), 66
 Fitting (class in deepmd.fit.fitting), 188
 fitting_dipole() (in module
 deepmd.utils.argcheck), 253
 fitting_ener() (in module deepmd.utils.argcheck),
 253
 fitting_net:
 model/fitting_net (Argument), 63
 fitting_polar() (in module
 deepmd.utils.argcheck), 253
 fitting_variant_type_args() (in module
 deepmd.utils.argcheck), 253
 format_name_length()
 (deepmd.utils.data_system.DataSystem
 method), 262
 format_nlist_i_cpu (C++ function), 411
 format_nlist_i_fill_a (C++ function), 411
 freeze() (in module deepmd.entrypoints), 178
 freeze() (in module deepmd.entrypoints.freeze),
 181
G
 gelu() (in module deepmd.common), 286
 Gelu() (in module deepmd.env.op_module), 295
 gelu() (in module deepmd.env.op_module), 315
 gelu_custom() (in module deepmd.env.op_module),
 316
 gelu_grad() (in module deepmd.env.op_module),
 316
 gelu_grad_custom() (in module
 deepmd.env.op_module), 316
 gelu_grad_grad() (in module
 deepmd.env.op_module), 316
 gelu_grad_grad_custom() (in module
 deepmd.env.op_module), 316
 gelu_tf() (in module deepmd.common), 286
 GeluCustom() (in module deepmd.env.op_module),
 295
 GeluGPUExecuteFunctor (C++ struct), 370
 GeluGPUExecuteFunctor::operator() (C++ func-
 tion), 370
 GeluGrad() (in module deepmd.env.op_module), 296
 GeluGradCustom() (in module
 deepmd.env.op_module), 296
 GeluGradGPUExecuteFunctor (C++ struct), 370
 GeluGradGPUExecuteFunctor::operator() (C++
 function), 371
 GeluGradGrad() (in module
 deepmd.env.op_module), 296
 GeluGradGradCustom() (in module
 deepmd.env.op_module), 296
 GeluGradGradGPUExecuteFunctor (C++ struct), 371
 GeluGradGradGPUExecuteFunctor::operator()
 (C++ function), 371
 gen_doc() (in module deepmd.utils.argcheck), 253
 gen_json() (in module deepmd.utils.argcheck), 253
 generate() (deepmd.utils.parallel_op.ParallelOp
 method), 273
 get() (deepmd.nvnmd.utils.fio.FioDic method), 247
 get() (deepmd.nvnmd.utils.FioDic method), 241
 get() (deepmd.utils.pair_tab.PairTab method), 272
 get_activation_func() (in module
 deepmd.common), 287

`get_all_argument()` (deepmd.utils.argcheck.ArgsPlugin method), 252
`get_atom_type()` (deepmd.utils.data.DeepmdData method), 261
`get_attention_layer_nodes_from_graph_def()` (in module deepmd.utils.graph), 265
`get_attention_layer_variables_from_graph_def()` (in module deepmd.utils.graph), 265
`get_batch()` (deepmd.utils.data.DataSets method), 258
`get_batch()` (deepmd.utils.data.DeepmdData method), 261
`get_batch()` (deepmd.utils.data_system.DataSystem method), 262
`get_batch()` (deepmd.utils.data_system.DeepmdDataSystem method), 264
`get_batch_size()` (deepmd.utils.data_system.DataSystem method), 262
`get_batch_size()` (deepmd.utils.data_system.DeepmdDataSystem method), 264
`get_constant_initializer()` (in module deepmd.nvnmd.utils.weight), 249
`get_data_dict()` (deepmd.utils.data.DeepmdData method), 261
`get_data_dict()` (deepmd.utils.data_system.DeepmdDataSystem method), 264
`get_deepmd_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 243
`get_dict()` (deepmd.common.ClassArg method), 285
`get_dim_aparam()` (deepmd.infer.deep_dipole.DeepDipole method), 207
`get_dim_aparam()` (deepmd.infer.deep_polar.DeepGlobalPolar method), 210
`get_dim_aparam()` (deepmd.infer.deep_polar.DeepPolar method), 211
`get_dim_aparam()` (deepmd.infer.deep_pot.DeepPot method), 214
`get_dim_aparam()` (deepmd.infer.deep_tensor.DeepTensor method), 216
`get_dim_aparam()` (deepmd.infer.deep_wfc.DeepWFC method), 218
`get_dim_aparam()` (deepmd.infer.DeepDipole method), 194
`get_dim_aparam()` (deepmd.infer.DeepGlobalPolar method), 197
`get_dim_aparam()` (deepmd.infer.DeepPolar method), 198
`get_dim_aparam()` (deepmd.infer.DeepPot method), 200
`get_dim_aparam()` (deepmd.infer.DeepWFC method), 202
`get_dim_out()` (deepmd.descriptor.descriptor.Descriptor method), 146
`get_dim_out()` (deepmd.descriptor.hybrid.DescriptHybrid method), 151
`get_dim_out()` (deepmd.descriptor.loc_frame.DescriptLocFrame method), 155
`get_dim_out()` (deepmd.descriptor.se_a.DescriptSeA method), 161
`get_dim_out()` (deepmd.descriptor.se_a_ef.DescriptSeAEf method), 166
`get_dim_out()` (deepmd.descriptor.se_r.DescriptSeR method), 173
`get_dim_out()` (deepmd.descriptor.se_t.DescriptSeT method), 176
`get_dim_rot_mat_1()` (deepmd.descriptor.descriptor.Descriptor method), 147
`get_dim_rot_mat_1()` (deepmd.descriptor.se_a.DescriptSeA method), 161
`get_dim_rot_mat_1()` (deepmd.descriptor.se_a_ef.DescriptSeAEf method), 166
`get_dscp_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 243
`get_embedding_net_nodes()` (in module deepmd.utils.graph), 265
`get_embedding_net_nodes_from_graph_def()` (in module deepmd.utils.graph), 266
`get_embedding_net_variables()` (in module deepmd.utils.graph), 266
`get_embedding_net_variables_from_graph_def()` (in module deepmd.utils.graph), 266
`get_ener()` (deepmd.utils.data.DataSets method),

258
`get_evaluation_results()` (deepmd.train.trainer.DPTrainer method), 251
`get_feed_dict()` (deepmd.descriptor.descriptor.Descriptor method), 147
`get_feed_dict()` (deepmd.train.trainer.DPTrainer method), 251
`get_file_list()` (deepmd.nvnmd.utils.fio.Fio method), 247
`get_filter_weight()` (in module deepmd.nvnmd.utils), 241
`get_filter_weight()` (in module deepmd.nvnmd.utils.weight), 249
`get_fitn_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 243
`get_fitnet_weight()` (in module deepmd.nvnmd.utils), 242
`get_fitnet_weight()` (in module deepmd.nvnmd.utils.weight), 249
`get_fitting_net_nodes()` (in module deepmd.utils.graph), 266
`get_fitting_net_nodes_from_graph_def()` (in module deepmd.utils.graph), 266
`get_fitting_net_variables()` (in module deepmd.utils.graph), 267
`get_fitting_net_variables_from_graph_def()` (in module deepmd.utils.graph), 267
`get_global_step()` (deepmd.train.trainer.DPTrainer method), 251
`get_gpus()` (in module deepmd.cluster.local), 143
`get_learning_rate_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 243
`get_ll()` (in module deepmd.entrypoints.main), 182
`get_loss_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 243
`get_model_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 243
`get_natoms()` (deepmd.utils.data.DataSets method), 258
`get_natoms()` (deepmd.utils.data.DeepmdData method), 261
`get_natoms_2()` (deepmd.utils.data.DataSets method), 258
`get_natoms_vec()` (deepmd.utils.data.DataSets method), 259
`get_natoms_vec()` (deepmd.utils.data.DeepmdData method), 261
`get_nbatches()` (deepmd.utils.data_system.DataSystem method), 262
`get_nbatches()` (deepmd.utils.data_system.DeepmdDataSystem method), 264
`get_nlist()` (deepmd.descriptor.descriptor.Descriptor method), 147
`get_nlist()` (deepmd.descriptor.loc_frame.DescriptLocFrame method), 155
`get_nlist()` (deepmd.descriptor.se_a.DescriptSeA method), 161
`get_nlist()` (deepmd.descriptor.se_a_ef.DescriptSeAEf method), 166
`get_nlist()` (deepmd.descriptor.se_r.DescriptSeR method), 173
`get_nlist()` (deepmd.descriptor.se_t.DescriptSeT method), 176
`get_nlist_i()` (deepmd.descriptor.hybrid.DescriptHybrid method), 151
`get_normalize()` (in module deepmd.nvnmd.utils.weight), 249
`get_np_precision()` (in module deepmd.common), 287
`get_nsystems()` (deepmd.utils.data_system.DataSystem method), 262
`get_nsystems()` (deepmd.utils.data_system.DeepmdDataSystem method), 264
`get_ntypes()` (deepmd.descriptor.descriptor.Descriptor method), 147
`get_ntypes()` (deepmd.descriptor.hybrid.DescriptHybrid method), 152
`get_ntypes()` (deepmd.descriptor.loc_frame.DescriptLocFrame method), 155
`get_ntypes()` (deepmd.descriptor.se_a.DescriptSeA method), 161
`get_ntypes()` (deepmd.descriptor.se_a_ef.DescriptSeAEf method), 166
`get_ntypes()` (deepmd.descriptor.se_r.DescriptSeR method), 173
`get_ntypes()` (deepmd.descriptor.se_t.DescriptSeT method), 176
`get_ntypes()` (deepmd.infer.deep_pot.DeepPot method), 214
`get_ntypes()` (deepmd.infer.deep_tensor.DeepTensor method), 216
`get_ntypes()` (deepmd.infer.DeepPot method), 200
`get_ntypes()` (deepmd.model.ener.EnerModel method), 227
`get_ntypes()` (deepmd.model.tensor.TensorModel method), 230
`get_ntypes()` (deepmd.utils.data.DeepmdData method), 261
`get_ntypes()` (deepmd.utils.data_system.DataSystem method), 262
`get_ntypes()` (deepmd.utils.data_system.DeepmdDataSystem method), 264
`get_numb_aparam()` (deepmd.fit.ener.EnerFitting method), 188
`get_numb_batch()` (deepmd.utils.data.DeepmdData method), 261

`get_numb_fparam()` (deepmd.fit.ener.EnerFitting method), 188
`get_numb_set()` (deepmd.utils.data.DataSets method), 259
`get_numb_set()` (deepmd.utils.data.DeepmdData method), 261
`get_nvnmdata_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 166
`get_out_size()` (deepmd.fit.dipole.DipoleFittingSeA method), 185
`get_out_size()` (deepmd.fit.polar.GlobalPolarFittingSeA method), 190
`get_out_size()` (deepmd.fit.polar.PolarFittingLocFrame method), 191
`get_out_size()` (deepmd.fit.polar.PolarFittingSeA method), 192
`get_out_size()` (deepmd.fit.wfc.WFCFitting method), 193
`get_out_size()` (deepmd.model.tensor.TensorModel method), 230
`get_pattern_nodes_from_graph_def()` (in module deepmd.utils.graph), 267
`get_plugin()` (deepmd.utils.plugin.Plugin method), 277
`get_precision()` (in module deepmd.common), 287
`get_rcut()` (deepmd.descriptor.descriptor.Descriptor method), 148
`get_rcut()` (deepmd.descriptor.hybrid.DescriptHybrid method), 152
`get_rcut()` (deepmd.descriptor.loc_frame.DescriptLocFrame method), 155
`get_rcut()` (deepmd.descriptor.se_a.DescriptSeA method), 161
`get_rcut()` (deepmd.descriptor.se_a_ef.DescriptSeAEf method), 166
`get_rcut()` (deepmd.descriptor.se_r.DescriptSeR method), 173
`get_rcut()` (deepmd.descriptor.se_t.DescriptSeT method), 176
`get_rcut()` (deepmd.infer.deep_pot.DeepPot method), 214
`get_rcut()` (deepmd.infer.deep_tensor.DeepTensor method), 216
`get_rcut()` (deepmd.infer.DeepPot method), 200
`get_rcut()` (deepmd.model.ener.EnerModel method), 227
`get_rcut()` (deepmd.model.tensor.TensorModel method), 230
`get_resource()` (in module deepmd.cluster), 143
`get_resource()` (in module deepmd.cluster.local), 143
`get_resource()` (in module deepmd.cluster.slurm), 143
`get_rng_s()` (in module deepmd.nvnmd.utils.weight), 250
`get_rot_mat()` (deepmd.descriptor.loc_frame.DescriptLocFrame method), 155
`get_rot_mat()` (deepmd.descriptor.se_a.DescriptSeA method), 161
`get_rot_mat()` (deepmd.descriptor.se_a_ef.DescriptSeAEf method), 166
`get_sel_type()` (deepmd.fit.dipole.DipoleFittingSeA method), 185
`get_sel_type()` (deepmd.fit.polar.GlobalPolarFittingSeA method), 190
`get_sel_type()` (deepmd.fit.polar.PolarFittingLocFrame method), 191
`get_sel_type()` (deepmd.fit.polar.PolarFittingSeA method), 192
`get_sel_type()` (deepmd.fit.wfc.WFCFitting method), 193
`get_sel_type()` (deepmd.infer.deep_pot.DeepPot method), 214
`get_sel_type()` (deepmd.infer.deep_tensor.DeepTensor method), 217
`get_sel_type()` (deepmd.infer.DeepPot method), 201
`get_sel_type()` (deepmd.model.tensor.TensorModel method), 230
`get_sess()` (in module deepmd.nvnmd.utils.network), 248
`get_set()` (deepmd.utils.data.DataSets method), 259
`get_stat()` (deepmd.utils.neighbor_stat.NeighborStat method), 270
`get_sys()` (deepmd.utils.data_system.DataSystem method), 262
`get_sys()` (deepmd.utils.data_system.DeepmdDataSystem method), 264
`get_sys_nctest()` (deepmd.utils.data_system.DeepmdDataSystem method), 264
`get_sys_num_batch()` (deepmd.utils.data.DataSets method), 259
`get_sys_num_batch()` (deepmd.utils.data.DeepmdData method), 261
`get_tensor_by_name()` (in module deepmd.utils.graph), 267
`get_tensor_by_name_from_graph()` (in module deepmd.utils.graph), 268
`get_tensor_by_type()` (in module deepmd.utils.graph), 268
`get_tensor_names()` (deepmd.descriptor.descriptor.Descriptor method), 148
`get_tensor_names()` (deepmd.descriptor.hybrid.DescriptHybrid method), 152
`get_tensor_names()` (deepmd.descriptor.se.DescriptSe method), 157

`get_test()` (deepmd.utils.data.DataSets method), 259
`get_test()` (deepmd.utils.data.DeepmdData method), 261
`get_test()` (deepmd.utils.data_system.DataSystem method), 262
`get_test()` (deepmd.utils.data_system.DeepmdDataSystem method), 264
`get_training_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 243
`get_type_embedding_net_nodes_from_graph_def()` (in module deepmd.utils.graph), 268
`get_type_embedding_net_variables_from_graph_def()` (in module deepmd.utils.graph), 268
`get_type_map()` (deepmd.infer.deep_pot.DeepPot method), 214
`get_type_map()` (deepmd.infer.deep_tensor.DeepTensor method), 217
`get_type_map()` (deepmd.infer.DeepPot method), 201
`get_type_map()` (deepmd.model.ener.EnerModel method), 227
`get_type_map()` (deepmd.model.tensor.TensorModel method), 230
`get_type_map()` (deepmd.utils.data.DataSets method), 259
`get_type_map()` (deepmd.utils.data.DeepmdData method), 261
`get_type_map()` (deepmd.utils.data_system.DataSystem method), 262
`get_type_map()` (deepmd.utils.data_system.DeepmdDataSystem method), 264
`get_weight()` (in module deepmd.nvnmd.utils.weight), 250
`get_wfc_numb()` (deepmd.fit.wfc.WFCFitting method), 193
`glob()` (deepmd.utils.path.DPH5Path method), 274
`glob()` (deepmd.utils.path.DPOSPath method), 275
`glob()` (deepmd.utils.path.DPPPath method), 276
`global_cvt_2_ener_float()` (in module deepmd.env), 289
`global_cvt_2_tf_float()` (in module deepmd.env), 289
`GLOBAL_ENER_FLOAT_PRECISION` (in module deepmd.env), 289
`GLOBAL_NP_FLOAT_PRECISION` (in module deepmd.env), 289
`GlobalPolarFittingSeA` (class in deepmd.fit.polar), 189
`GlobalPolarModel` (class in deepmd.model.tensor), 228
`GPU_MAX_NBOR_SIZE` (C macro), 413
`gpus` (deepmd.train.run_options.RunOptions attribute), 251
`GraphTooLargeError`, 265
`GraphWithoutTensorError`, 265
H
`hex2bin()` (deepmd.nvnmd.utils.Encode method), 240
`hex2bin()` (deepmd.nvnmd.utils.encode.Encode method), 246
`hex2bin_str()` (deepmd.nvnmd.utils.Encode method), 240
`hex2bin_str()` (deepmd.nvnmd.utils.encode.Encode method), 246
I
`implemented_properties` (deepmd.calculator.DP attribute), 284
`import_ops()` (in module deepmd.op), 250
`init_ctrl()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 244
`init_dscp()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 244
`init_fitn()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 244
`init_from_config()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 244
`init_from_deepmd_input()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 244
`init_from_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 244
`init_net()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 244
`init_net_size()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 244
`init_size()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 244
`init_train_mode()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 244
`init_value()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 244
`init_variables()` (deepmd.descriptor.descriptor.Descriptor method), 148
`init_variables()` (deepmd.descriptor.hybrid.DescriptHybrid method), 152
`init_variables()` (deepmd.descriptor.loc_frame.DescriptLocFrame method), 155
`init_variables()` (deepmd.descriptor.se.DescriptSe method), 157
`init_variables()` (deepmd.descriptor.se_a.DescriptSeA method), 161
`init_variables()` (deepmd.descriptor.se_atten.DescriptSeAtten method), 170

`init_variables()` (deepmd.fit.dipole.DipoleFittingSeaA method), 185
`init_variables()` (deepmd.fit.ener.EnerFittingSeaA method), 188
`init_variables()` (deepmd.fit.fitting.FittingSeaA method), 189
`init_variables()` (deepmd.fit.polar.GlobalPolarFittingSeaA method), 190
`init_variables()` (deepmd.fit.polar.PolarFittingSeaA method), 192
`init_variables()` (deepmd.model.ener.EnerModel method), 227
`init_variables()` (deepmd.model.model.Model method), 227
`init_variables()` (deepmd.model.tensor.TensorModel method), 230
`init_variables()` (deepmd.utils.type_embed.TypeEmbedder method), 281
`int_64` (C++ type), 415
`is_chief` (deepmd.train.run_options.RunOptions property), 251
`is_dir()` (deepmd.utils.path.DPH5Path method), 274
`is_dir()` (deepmd.utils.path.DPOSPath method), 275
`is_dir()` (deepmd.utils.path.DPPath method), 276
`is_file()` (deepmd.nvnmd.utils.fio.Fio method), 247
`is_file()` (deepmd.utils.path.DPH5Path method), 274
`is_file()` (deepmd.utils.path.DPOSPath method), 275
`is_file()` (deepmd.utils.path.DPPath method), 276
`is_path()` (deepmd.nvnmd.utils.fio.Fio method), 247

J
`j_loader()` (in module deepmd.common), 287
`j_must_have()` (in module deepmd.common), 287

L
`learning_rate` (Argument)
 `learning_rate`:, 68
`learning_rate/scale_by_worker` (Argument)
 `scale_by_worker`:, 68
`learning_rate/type` (Argument)
 `type`:, 69
`learning_rate`:
 `learning_rate` (Argument), 68
`learning_rate_args()` (in module deepmd.utils.argcheck), 253
`learning_rate_exp()` (in module deepmd.utils.argcheck), 253

`learning_rate_variant_type_args()` (in module deepmd.utils.argcheck), 253
`learning_rate[exp]/decay_steps` (Argument)
 `decay_steps`:, 69
`learning_rate[exp]/start_lr` (Argument)
 `start_lr`:, 69
`learning_rate[exp]/stop_lr` (Argument)
 `stop_lr`:, 69
`LearningRateExp` (class in module deepmd.utils.learning_rate), 269
`limit_pref()` (in module deepmd.utils.argcheck), 253
`limit_pref_ae`:
 `loss[ener]/limit_pref_ae` (Argument), 70
`limit_pref_e`:
 `loss[ener]/limit_pref_e` (Argument), 69
`limit_pref_f`:
 `loss[ener]/limit_pref_f` (Argument), 70
`limit_pref_pf`:
 `loss[ener]/limit_pref_pf` (Argument), 71
`limit_pref_v`:
 `loss[ener]/limit_pref_v` (Argument), 70
`list`:
 `model/descriptor[hybrid]/list` (Argument), 61
`list_to_doc()` (in module deepmd.utils.argcheck), 253
`load()` (deepmd.nvnmd.utils.fio.FioBin method), 247
`load()` (deepmd.nvnmd.utils.fio.FioDic method), 247
`load()` (deepmd.nvnmd.utils.fio.FioJsonDic method), 248
`load()` (deepmd.nvnmd.utils.fio.FioNpyDic method), 248
`load()` (deepmd.nvnmd.utils.fio.FioTxt method), 248
`load()` (deepmd.nvnmd.utils.FioBin method), 241
`load()` (deepmd.nvnmd.utils.FioDic method), 241
`load()` (deepmd.nvnmd.utils.FioTxt method), 241
`load_batch_set()` (deepmd.utils.data.DataSets method), 259
`load_data()` (deepmd.utils.data.DataSets method), 259
`load_energy()` (deepmd.utils.data.DataSets method), 259
`load_graph_def()` (in module deepmd.utils.graph), 269
`load_numpy()` (deepmd.utils.path.DPH5Path method), 274
`load_numpy()` (deepmd.utils.path.DPOSPath method), 275
`load_numpy()` (deepmd.utils.path.DPPath method), 276
`load_prefix` (deepmd.DeepEval attribute), 139

`load_prefix` (deepmd.DipoleChargeModifier attribute), 142
`load_prefix` (deepmd.infer.data_modifier.DipoleChargeModifierdeepmd.utils.argcheck), 253
 attribute), 206
`load_prefix` (deepmd.infer.deep_dipole.DeepDipole attribute), 207
`load_prefix` (deepmd.infer.deep_eval.DeepEval attribute), 208
`load_prefix` (deepmd.infer.deep_polar.DeepGlobalPolar attribute), 210
`load_prefix` (deepmd.infer.deep_polar.DeepPolar attribute), 211
`load_prefix` (deepmd.infer.deep_pot.DeepPot attribute), 214
`load_prefix` (deepmd.infer.deep_tensor.DeepTensor attribute), 217
`load_prefix` (deepmd.infer.deep_wfc.DeepWFC attribute), 218
`load_prefix` (deepmd.infer.DeepDipole attribute), 194
`load_prefix` (deepmd.infer.DeepEval attribute), 195
`load_prefix` (deepmd.infer.DeepGlobalPolar attribute), 197
`load_prefix` (deepmd.infer.DeepPolar attribute), 198
`load_prefix` (deepmd.infer.DeepPot attribute), 201
`load_prefix` (deepmd.infer.DeepWFC attribute), 202
`load_prefix` (deepmd.infer.DipoleChargeModifier attribute), 203
`load_set` () (deepmd.utils.data.DataSets method), 259
`load_test_set` () (deepmd.utils.data.DataSets method), 259
`load_txt` () (deepmd.utils.path.DPH5Path method), 274
`load_txt` () (deepmd.utils.path.DPOSPath method), 275
`load_txt` () (deepmd.utils.path.DPPath method), 276
`load_type` () (deepmd.utils.data.DataSets method), 259
`load_type_map` () (deepmd.utils.data.DataSets method), 259
`loss` (Argument)
 loss:, 69
`Loss` (class in deepmd.loss.loss), 224
`loss/type` (Argument)
 type:, 69
`loss:`
 loss (Argument), 69
`loss_args` () (in module deepmd.utils.argcheck), 253
`loss_ener` () (in module deepmd.utils.argcheck), 253
`loss_tensor` () (in module deepmd.utils.argcheck), 253
`loss_variant_type_args` () (in module deepmd.utils.argcheck), 253
`loss[ener]/enable_atom_ener_coeff` (Argument)
 enable_atom_ener_coeff:, 71
`loss[ener]/limit_pref_ae` (Argument)
 limit_pref_ae:, 70
`loss[ener]/limit_pref_e` (Argument)
 limit_pref_e:, 69
`loss[ener]/limit_pref_f` (Argument)
 limit_pref_f:, 70
`loss[ener]/limit_pref_pf` (Argument)
 limit_pref_pf:, 71
`loss[ener]/limit_pref_v` (Argument)
 limit_pref_v:, 70
`loss[ener]/relative_f` (Argument)
 relative_f:, 71
`loss[ener]/start_pref_ae` (Argument)
 start_pref_ae:, 70
`loss[ener]/start_pref_e` (Argument)
 start_pref_e:, 69
`loss[ener]/start_pref_f` (Argument)
 start_pref_f:, 70
`loss[ener]/start_pref_pf` (Argument)
 start_pref_pf:, 71
`loss[ener]/start_pref_v` (Argument)
 start_pref_v:, 70
`loss[tensor]/pref` (Argument)
 pref:, 71
`loss[tensor]/pref_atomic` (Argument)
 pref_atomic:, 71

M

`main` () (in module deepmd.entrypoints.main), 182
`main_parser` () (in module deepmd.entrypoints.main), 182
`make_default_mesh` () (in module deepmd.common), 288
`make_index` () (in module deepmd.utils.argcheck), 253
`make_link` () (in module deepmd.utils.argcheck), 253
`make_model_devi` () (in module deepmd.entrypoints), 178
`make_model_devi` () (in module deepmd.infer.model_devi), 219
`make_natoms_vec` () (deepmd.DeepEval method), 139
`make_natoms_vec` () (deepmd.infer.deep_eval.DeepEval method), 208
`make_natoms_vec` () (deepmd.infer.DeepEval method), 195
`make_stat_input` () (in module deepmd.model.model_stat), 228

map_aparam() (in module deepmd.env.op_module), 317	(Argument)
map_file:	table_config:, 68
nvnmnd/map_file (Argument), 76	model/data_stat_nbatch (Argument)
map_nvnmnd() (in module deepmd.env.op_module), 317	data_stat_nbatch:, 51
map_nvnmnd() (in module deepmd.nvnmnd.utils), 242	model/data_stat_protect (Argument)
map_nvnmnd() (in module deepmd.nvnmnd.utils.op), 249	data_stat_protect:, 51
MapAparam() (in module deepmd.env.op_module), 296	model/descriptor (Argument)
MapNvnmnd() (in module deepmd.env.op_module), 297	descriptor:, 52
mapt() (in module deepmd.nvnmnd.entrypoints.mapt), 237	model/descriptor/type (Argument)
MapTable (class in deepmd.nvnmnd.entrypoints), 232	type:, 52
MapTable (class in deepmd.nvnmnd.entrypoints.mapt), 235	model/descriptor[hybrid]/list (Argument)
matmul2_qq() (in module deepmd.nvnmnd.utils.network), 248	list:, 61
matmul3_qq() (in module deepmd.nvnmnd.utils.network), 248	model/descriptor[loc_frame]/axis_rule
matmul_nvnmnd() (in module deepmd.env.op_module), 317	(Argument)
MatmulNvnmnd() (in module deepmd.env.op_module), 297	axis_rule:, 53
merge_bin() (deepmd.nvnmnd.utils.Encode method), 240	model/descriptor[loc_frame]/rcut (Argument)
merge_bin() (deepmd.nvnmnd.utils.encode.Encode method), 246	rcut:, 53
merge_sys_stat() (in module deepmd.model.model_stat), 228	model/descriptor[loc_frame]/sel_a (Argument)
min_nbor_dist:	sel_a:, 53
model/compress[se_e2_a]/min_nbor_dist	model/descriptor[loc_frame]/sel_r (Argument)
(Argument), 68	sel_r:, 53
mixed_precision:	model/descriptor[se_a_tpe]/activation_function
training/mixed_precision (Argument), 74	(Argument)
mixed_precision_args() (in module deepmd.utils.argcheck), 253	activation_function:, 58
mkdir() (deepmd.nvnmnd.utils.fio.Fio method), 247	model/descriptor[se_a_tpe]/axis_neuron
MOASPDIM (C macro), 414	(Argument)
model (Argument)	axis_neuron:, 58
model:, 50	model/descriptor[se_a_tpe]/exclude_types
Model (class in deepmd.model.model), 227	(Argument)
model/compress (Argument)	exclude_types:, 58
compress:, 68	model/descriptor[se_a_tpe]/neuron (Argument)
model/compress/type (Argument)	neuron:, 57
type:, 68	model/descriptor[se_a_tpe]/numb_aparam
model/compress[se_e2_a]/min_nbor_dist	(Argument)
(Argument)	numb_aparam:, 59
min_nbor_dist:, 68	model/descriptor[se_a_tpe]/precision
model/compress[se_e2_a]/model_file	(Argument)
(Argument)	precision:, 58
model_file:, 68	model/descriptor[se_a_tpe]/rcut (Argument)
model/compress[se_e2_a]/table_config	rcut:, 57
	model/descriptor[se_a_tpe]/rcut_smth
	(Argument)
	rcut_smth:, 57
	model/descriptor[se_a_tpe]/resnet_dt
	(Argument)
	resnet_dt:, 58
	model/descriptor[se_a_tpe]/seed (Argument)
	seed:, 58
	model/descriptor[se_a_tpe]/sel (Argument)
	sel:, 57
	model/descriptor[se_a_tpe]/set_davg_zero
	(Argument)
	set_davg_zero:, 58

```

model/descriptor[se_a_tpe]/trainable
    (Argument)
    trainable:, 58
model/descriptor[se_a_tpe]/type_nchanl
    (Argument)
    type_nchanl:, 59
model/descriptor[se_a_tpe]/type_nlayer
    (Argument)
    type_nlayer:, 59
model/descriptor[se_a_tpe]/type_one_side
    (Argument)
    type_one_side:, 58
model/descriptor[se_atten]/activation_function
    (Argument)
    activation_function:, 62
model/descriptor[se_atten]/attn (Argument)
    attn:, 63
model/descriptor[se_atten]/attn_dotr
    (Argument)
    attn_dotr:, 63
model/descriptor[se_atten]/attn_layer
    (Argument)
    attn_layer:, 63
model/descriptor[se_atten]/attn_mask
    (Argument)
    attn_mask:, 63
model/descriptor[se_atten]/axis_neuron
    (Argument)
    axis_neuron:, 62
model/descriptor[se_atten]/exclude_types
    (Argument)
    exclude_types:, 62
model/descriptor[se_atten]/neuron (Argument)
    neuron:, 61
model/descriptor[se_atten]/precision
    (Argument)
    precision:, 62
model/descriptor[se_atten]/rcut (Argument)
    rcut:, 61
model/descriptor[se_atten]/rcut_smth
    (Argument)
    rcut_smth:, 61
model/descriptor[se_atten]/resnet_dt
    (Argument)
    resnet_dt:, 62
model/descriptor[se_atten]/seed (Argument)
    seed:, 62
model/descriptor[se_atten]/sel (Argument)
    sel:, 61
model/descriptor[se_atten]/set_davg_zero
    (Argument)
    set_davg_zero:, 62
model/descriptor[se_atten]/trainable
    (Argument)
    trainable:, 62
model/descriptor[se_atten]/type_one_side
    (Argument)
    type_one_side:, 62
model/descriptor[se_e2_a]/activation_function
    (Argument)
    activation_function:, 54
model/descriptor[se_e2_a]/axis_neuron
    (Argument)
    axis_neuron:, 54
model/descriptor[se_e2_a]/exclude_types
    (Argument)
    exclude_types:, 55
model/descriptor[se_e2_a]/neuron (Argument)
    neuron:, 54
model/descriptor[se_e2_a]/precision
    (Argument)
    precision:, 55
model/descriptor[se_e2_a]/rcut (Argument)
    rcut:, 54
model/descriptor[se_e2_a]/rcut_smth
    (Argument)
    rcut_smth:, 54
model/descriptor[se_e2_a]/resnet_dt
    (Argument)
    resnet_dt:, 55
model/descriptor[se_e2_a]/seed (Argument)
    seed:, 55
model/descriptor[se_e2_a]/sel (Argument)
    sel:, 54
model/descriptor[se_e2_a]/set_davg_zero
    (Argument)
    set_davg_zero:, 55
model/descriptor[se_e2_a]/trainable
    (Argument)
    trainable:, 55
model/descriptor[se_e2_a]/type_one_side
    (Argument)
    type_one_side:, 55
model/descriptor[se_e2_r]/activation_function
    (Argument)
    activation_function:, 60
model/descriptor[se_e2_r]/exclude_types
    (Argument)
    exclude_types:, 60
model/descriptor[se_e2_r]/neuron (Argument)
    neuron:, 59
model/descriptor[se_e2_r]/precision
    (Argument)
    precision:, 60
model/descriptor[se_e2_r]/rcut (Argument)
    rcut:, 59
model/descriptor[se_e2_r]/rcut_smth
    (Argument)

```

```

    rcut_smth:, 59
model/descriptor[se_e2_r]/resnet_dt
    (Argument)
    resnet_dt:, 60
model/descriptor[se_e2_r]/seed (Argument)
    seed:, 60
model/descriptor[se_e2_r]/sel (Argument)
    sel:, 59
model/descriptor[se_e2_r]/set_davg_zero
    (Argument)
    set_davg_zero:, 60
model/descriptor[se_e2_r]/trainable
    (Argument)
    trainable:, 60
model/descriptor[se_e2_r]/type_one_side
    (Argument)
    type_one_side:, 60
model/descriptor[se_e3]/activation_function
    (Argument)
    activation_function:, 56
model/descriptor[se_e3]/neuron (Argument)
    neuron:, 56
model/descriptor[se_e3]/precision (Argument)
    precision:, 56
model/descriptor[se_e3]/rcut (Argument)
    rcut:, 56
model/descriptor[se_e3]/rcut_smth (Argument)
    rcut_smth:, 56
model/descriptor[se_e3]/resnet_dt (Argument)
    resnet_dt:, 56
model/descriptor[se_e3]/seed (Argument)
    seed:, 57
model/descriptor[se_e3]/sel (Argument)
    sel:, 55
model/descriptor[se_e3]/set_davg_zero
    (Argument)
    set_davg_zero:, 57
model/descriptor[se_e3]/trainable (Argument)
    trainable:, 57
model/fitting_net (Argument)
    fitting_net:, 63
model/fitting_net/type (Argument)
    type:, 63
model/fitting_net[dipole]/activation_function
    (Argument)
    activation_function:, 65
model/fitting_net[dipole]/neuron (Argument)
    neuron:, 65
model/fitting_net[dipole]/precision
    (Argument)
    precision:, 65
model/fitting_net[dipole]/resnet_dt
    (Argument)
    resnet_dt:, 65
model/fitting_net[dipole]/seed (Argument)
    seed:, 65
model/fitting_net[dipole]/sel_type
    (Argument)
    sel_type:, 65
model/fitting_net[ener]/activation_function
    (Argument)
    activation_function:, 64
model/fitting_net[ener]/atom_ener (Argument)
    atom_ener:, 65
model/fitting_net[ener]/neuron (Argument)
    neuron:, 64
model/fitting_net[ener]/numb_aparam
    (Argument)
    numb_aparam:, 64
model/fitting_net[ener]/numb_fparam
    (Argument)
    numb_fparam:, 63
model/fitting_net[ener]/precision (Argument)
    precision:, 64
model/fitting_net[ener]/rcond (Argument)
    rcond:, 64
model/fitting_net[ener]/resnet_dt (Argument)
    resnet_dt:, 64
model/fitting_net[ener]/seed (Argument)
    seed:, 65
model/fitting_net[ener]/trainable (Argument)
    trainable:, 64
model/fitting_net[polar]/activation_function
    (Argument)
    activation_function:, 66
model/fitting_net[polar]/fit_diag (Argument)
    fit_diag:, 66
model/fitting_net[polar]/neuron (Argument)
    neuron:, 66
model/fitting_net[polar]/precision
    (Argument)
    precision:, 66
model/fitting_net[polar]/resnet_dt
    (Argument)
    resnet_dt:, 66
model/fitting_net[polar]/scale (Argument)
    scale:, 66
model/fitting_net[polar]/seed (Argument)
    seed:, 67
model/fitting_net[polar]/sel_type (Argument)
    sel_type:, 66
model/fitting_net[polar]/shift_diag
    (Argument)
    shift_diag:, 66
model/modifier (Argument)
    modifier:, 67
model/modifier/type (Argument)
    type:, 67

```

```

model/modifier[dipole_charge]/ewald_beta
    (Argument)
    ewald_beta:, 67
model/modifier[dipole_charge]/ewald_h
    (Argument)
    ewald_h:, 67
model/modifier[dipole_charge]/model_charge_map
    (Argument)
    model_charge_map:, 67
model/modifier[dipole_charge]/model_name
    (Argument)
    model_name:, 67
model/modifier[dipole_charge]/sys_charge_map
    (Argument)
    sys_charge_map:, 67
model/smin_alpha (Argument)
    smin_alpha:, 51
model/sw_rmax (Argument)
    sw_rmax:, 51
model/sw_rmin (Argument)
    sw_rmin:, 51
model/type_embedding (Argument)
    type_embedding:, 51
model/type_embedding/activation_function
    (Argument)
    activation_function:, 52
model/type_embedding/neuron (Argument)
    neuron:, 51
model/type_embedding/precision (Argument)
    precision:, 52
model/type_embedding/resnet_dt (Argument)
    resnet_dt:, 52
model/type_embedding/seed (Argument)
    seed:, 52
model/type_embedding/trainable (Argument)
    trainable:, 52
model/type_map (Argument)
    type_map:, 50
model/use_srtab (Argument)
    use_srtab:, 51
model:
    model (Argument), 50
model_args() (in module deepmd.utils.argcheck),
    253
model_charge_map:
    model/modifier[dipole_charge]/model_charge_map
        (Argument), 67
model_compression() (in module
    deepmd.utils.argcheck), 254
model_compression_type_args() (in module
    deepmd.utils.argcheck), 254
model_file:
    model/compress[se_e2_a]/model_file
        (Argument), 68
model_name:
    model/modifier[dipole_charge]/model_name
        (Argument), 67
model_type (deepmd.DeepEval property), 140
model_type (deepmd.infer.deep_eval.DeepEval
    property), 208
model_type (deepmd.infer.DeepEval property), 195
model_type (deepmd.model.ener.EnerModel at-
    tribute), 227
model_version (deepmd.DeepEval property), 140
model_version (deepmd.infer.deep_eval.DeepEval
    property), 208
model_version (deepmd.infer.DeepEval property),
    195
modifier:
    model/modifier (Argument), 67
modifier_dipole_charge() (in module
    deepmd.utils.argcheck), 254
modifier_variant_type_args() (in module
    deepmd.utils.argcheck), 254
modify_data() (deepmd.DipoleChargeModifier
    method), 142
modify_data() (deepmd.infer.data_modifier.DipoleChargeModifier
    method), 206
modify_data() (deepmd.infer.DipoleChargeModifier
    method), 203
module
    deepmd, 139
    deepmd.calculator, 282
    deepmd.cluster, 143
    deepmd.cluster.local, 143
    deepmd.cluster.slurm, 143
    deepmd.common, 284
    deepmd.descriptor, 144
    deepmd.descriptor.descriptor, 144
    deepmd.descriptor.hybrid, 149
    deepmd.descriptor.loc_frame, 153
    deepmd.descriptor.se, 156
    deepmd.descriptor.se_a, 158
    deepmd.descriptor.se_a_ebd, 162
    deepmd.descriptor.se_a_ef, 164
    deepmd.descriptor.se_atten, 168
    deepmd.descriptor.se_r, 171
    deepmd.descriptor.se_t, 174
    deepmd.entrypoints, 177
    deepmd.entrypoints.compress, 180
    deepmd.entrypoints.config, 181
    deepmd.entrypoints.convert, 181
    deepmd.entrypoints.doc, 181
    deepmd.entrypoints.freeze, 181
    deepmd.entrypoints.main, 182
    deepmd.entrypoints.neighbor_stat, 182
    deepmd.entrypoints.test, 183
    deepmd.entrypoints.train, 183

```

[deepmd.entripoints.transfer](#), 184
[deepmd.env](#), 289
[deepmd.env.op_grads_module](#), 332
[deepmd.env.op_module](#), 291
[deepmd.fit](#), 184
[deepmd.fit.dipole](#), 184
[deepmd.fit.ener](#), 186
[deepmd.fit.fitting](#), 188
[deepmd.fit.polar](#), 189
[deepmd.fit.wfc](#), 193
[deepmd.infer](#), 193
[deepmd.infer.data_modifier](#), 205
[deepmd.infer.deep_dipole](#), 207
[deepmd.infer.deep_eval](#), 208
[deepmd.infer.deep_polar](#), 209
[deepmd.infer.deep_pot](#), 212
[deepmd.infer.deep_tensor](#), 215
[deepmd.infer.deep_wfc](#), 217
[deepmd.infer.ewald_recip](#), 218
[deepmd.infer.model_devi](#), 219
[deepmd.loggers](#), 220
[deepmd.loggers.loggers](#), 221
[deepmd.loss](#), 222
[deepmd.loss.ener](#), 222
[deepmd.loss.loss](#), 224
[deepmd.loss.tensor](#), 225
[deepmd.model](#), 226
[deepmd.model.ener](#), 226
[deepmd.model.model](#), 227
[deepmd.model.model_stat](#), 228
[deepmd.model.tensor](#), 228
[deepmd.nvnmd](#), 231
[deepmd.nvnmd.data](#), 231
[deepmd.nvnmd.data.data](#), 232
[deepmd.nvnmd.descriptor](#), 232
[deepmd.nvnmd.descriptor.se_a](#), 232
[deepmd.nvnmd.entripoints](#), 232
[deepmd.nvnmd.entripoints.freeze](#), 235
[deepmd.nvnmd.entripoints.mapt](#), 235
[deepmd.nvnmd.entripoints.train](#), 237
[deepmd.nvnmd.entripoints.wrap](#), 237
[deepmd.nvnmd.fit](#), 238
[deepmd.nvnmd.fit.ener](#), 239
[deepmd.nvnmd.utils](#), 239
[deepmd.nvnmd.utils.argcheck](#), 242
[deepmd.nvnmd.utils.config](#), 242
[deepmd.nvnmd.utils.encode](#), 244
[deepmd.nvnmd.utils.fio](#), 246
[deepmd.nvnmd.utils.network](#), 248
[deepmd.nvnmd.utils.op](#), 249
[deepmd.nvnmd.utils.weight](#), 249
[deepmd.op](#), 250
[deepmd.train](#), 250
[deepmd.train.run_options](#), 250

[deepmd.train.trainer](#), 251
[deepmd.utils](#), 252
[deepmd.utils.argcheck](#), 252
[deepmd.utils.batch_size](#), 254
[deepmd.utils.compat](#), 255
[deepmd.utils.convert](#), 256
[deepmd.utils.data](#), 258
[deepmd.utils.data_system](#), 262
[deepmd.utils.errors](#), 265
[deepmd.utils.graph](#), 265
[deepmd.utils.learning_rate](#), 269
[deepmd.utils.neighbor_stat](#), 270
[deepmd.utils.network](#), 271
[deepmd.utils.pair_tab](#), 272
[deepmd.utils.parallel_op](#), 273
[deepmd.utils.path](#), 274
[deepmd.utils.plugin](#), 277
[deepmd.utils.random](#), 278
[deepmd.utils.sess](#), 279
[deepmd.utils.tabulate](#), 279
[deepmd.utils.type_embed](#), 280
[deepmd.utils.weight_avg](#), 282
[my_device](#) ([deepmd.train.run_options.RunOptions](#) attribute), 251
[my_rank](#) ([deepmd.train.run_options.RunOptions](#) attribute), 251

N

[name](#) ([deepmd.calculator.DP](#) attribute), 284
[nborAssert](#) (C++ function), 412
[nborErrcheck](#) (C macro), 414
[neighbor_stat\(\)](#) (in module [deepmd.entripoints](#)), 178
[neighbor_stat\(\)](#) (in module [deepmd.entripoints.neighbor_stat](#)), 182
[neighbor_stat\(\)](#) (in module [deepmd.env.op_module](#)), 318
[NeighborStat](#) (class in [deepmd.utils.neighbor_stat](#)), 270
[NeighborStat\(\)](#) (in module [deepmd.env.op_module](#)), 297
[net_size](#):
 [nvnmd/net_size](#) (Argument), 76
[neuron](#):
 [model/descriptor\[se_a_tpe\]/neuron](#) (Argument), 57
 [model/descriptor\[se_atten\]/neuron](#) (Argument), 61
 [model/descriptor\[se_e2_a\]/neuron](#) (Argument), 54
 [model/descriptor\[se_e2_r\]/neuron](#) (Argument), 59
 [model/descriptor\[se_e3\]/neuron](#) (Argument), 56

model/fitting_net[dipole]/neuron
 (Argument), 65
 model/fitting_net[ener]/neuron
 (Argument), 64
 model/fitting_net[polar]/neuron
 (Argument), 66
 model/type_embedding/neuron(Argument), 51
 modelist (deepmd.train.run_options.RunOptions
 attribute), 251
 nodename (deepmd.train.run_options.RunOptions
 attribute), 251
 normalize() (in module deepmd.utils.argcheck), 254
 normalize_hybrid_list() (in module
 deepmd.utils.argcheck), 254
 normalized_input() (in module
 deepmd.nvnmd.entrypoints.train), 237
 normalized_input_qnn() (in module
 deepmd.nvnmd.entrypoints.train), 237
 numb_aparam() (deepmd.utils.data.DataSets
 method), 259
 numb_aparam:
 model/descriptor[se_a_tpe]/numb_aparam
 (Argument), 59
 model/fitting_net[ener]/numb_aparam
 (Argument), 64
 numb_btch:
 training/validation_data/numb_btch
 (Argument), 74
 numb_fparam() (deepmd.utils.data.DataSets
 method), 259
 numb_fparam() (deepmd.utils.data_system.DataSystem
 method), 262
 numb_fparam:
 model/fitting_net[ener]/numb_fparam
 (Argument), 63
 numb_steps:
 training/numb_steps(Argument), 74
 nvnmd(Argument)
 nvnmd:, 76
 nvnmd/config_file(Argument)
 config_file:, 76
 nvnmd/enable(Argument)
 enable:, 76
 nvnmd/map_file(Argument)
 map_file:, 76
 nvnmd/net_size(Argument)
 net_size:, 76
 nvnmd/quantize_descriptor(Argument)
 quantize_descriptor:, 77
 nvnmd/quantize_fitting_net(Argument)
 quantize_fitting_net:, 77
 nvnmd/restore_descriptor(Argument)
 restore_descriptor:, 77
 nvnmd/restore_fitting_net(Argument)
 restore_fitting_net:, 77
 nvnmd/weight_file(Argument)
 weight_file:, 76
 nvnmd:
 nvnmd(Argument), 76
 nvnmd_args() (in module deepmd.nvnmd.utils), 242
 nvnmd_args() (in module
 deepmd.nvnmd.utils.argcheck), 242
 NvnmdConfig (class in deepmd.nvnmd.utils.config),
 242

O

omp_get_num_threads(C++ function), 412
 omp_get_thread_num(C++ function), 412
 one_layer() (in module deepmd.nvnmd.utils), 242
 one_layer() (in module
 deepmd.nvnmd.utils.network), 249
 one_layer() (in module deepmd.utils.network), 272
 one_layer_rand_seed_shift() (in module
 deepmd.utils.network), 272
 one_layer_wb() (in module
 deepmd.nvnmd.utils.network), 249
 OutOfMemoryError, 265
 output_prec:
 training/mixed_precision/output_prec
 (Argument), 74

P

pair_tab() (in module deepmd.env.op_module), 318
 PairTab (class in deepmd.utils.pair_tab), 272
 PairTab() (in module deepmd.env.op_module), 298
 parallel_prod_force_se_a() (in module
 deepmd.env.op_module), 318
 ParallelOp (class in deepmd.utils.parallel_op), 273
 ParallelProdForceSeA() (in module
 deepmd.env.op_module), 298
 parse() (deepmd.common.ClassArg method), 285
 parse_args() (in module
 deepmd.entrypoints.main), 182
 pass_tensors_from_frz_model()
 (deepmd.descriptor.descriptor.Descriptor
 method), 148
 pass_tensors_from_frz_model()
 (deepmd.descriptor.hybrid.DescrptHybrid
 method), 152
 pass_tensors_from_frz_model()
 (deepmd.descriptor.se.DescrptSe method),
 157
 Plugin (class in deepmd.utils.plugin), 277
 PluginVariant (class in deepmd.utils.plugin), 278
 PolarFittingLocFrame (class in deepmd.fit.polar),
 190
 PolarFittingSeA (class in deepmd.fit.polar), 191
 PolarModel (class in deepmd.model.tensor), 229

```

precision (deepmd.descriptor.se.DescriptSe property), 158
precision (deepmd.fit.fitting.Fitting property), 189
precision:
    model/descriptor[se_a_tpe]/precision (Argument), 58
    model/descriptor[se_atten]/precision (Argument), 62
    model/descriptor[se_e2_a]/precision (Argument), 55
    model/descriptor[se_e2_r]/precision (Argument), 60
    model/descriptor[se_e3]/precision (Argument), 56
    model/fitting_net[dipole]/precision (Argument), 65
    model/fitting_net[ener]/precision (Argument), 64
    model/fitting_net[polar]/precision (Argument), 66
    model/type_embedding/precision (Argument), 52
pref:
    loss[tensor]/pref (Argument), 71
pref_atomic:
    loss[tensor]/pref_atomic (Argument), 71
print_header() (deepmd.loss.ener.EnerDipoleLoss static method), 223
print_header() (deepmd.loss.ener.EnerStdLoss method), 224
print_header() (deepmd.loss.tensor.TensorLoss method), 226
print_header() (deepmd.train.trainer.DPTrainer static method), 251
print_on_training() (deepmd.loss.ener.EnerDipoleLoss method), 223
print_on_training() (deepmd.loss.ener.EnerStdLoss method), 224
print_on_training() (deepmd.loss.tensor.TensorLoss method), 226
print_on_training() (deepmd.train.trainer.DPTrainer static method), 252
print_resource_summary() (deepmd.train.run_options.RunOptions method), 251
print_summary() (deepmd.utils.data_system.DataSystem method), 262
print_summary() (deepmd.utils.data_system.DeepmdDataSystem method), 264
process_sys_weights() (deepmd.utils.data_system.DataSystem method), 263
prod_env_mat_a() (in module deepmd.env.op_module), 319
prod_env_mat_a_mix() (in module deepmd.env.op_module), 320
prod_env_mat_a_nvnmmd_quantize() (in module deepmd.env.op_module), 321
prod_env_mat_r() (in module deepmd.env.op_module), 321
prod_force() (in module deepmd.env.op_module), 322
prod_force_grad() (in module deepmd.env.op_grads_module), 335
prod_force_norot() (in module deepmd.env.op_module), 322
prod_force_se_a() (in module deepmd.env.op_module), 322
prod_force_se_a_grad() (in module deepmd.env.op_grads_module), 335
prod_force_se_r() (in module deepmd.env.op_module), 323
prod_force_se_r_grad() (in module deepmd.env.op_grads_module), 335
prod_force_virial() (deepmd.descriptor.descriptor.Descriptor method), 149
prod_force_virial() (deepmd.descriptor.hybrid.DescriptHybrid method), 152
prod_force_virial() (deepmd.descriptor.loc_frame.DescriptLocFrame method), 155
prod_force_virial() (deepmd.descriptor.se_a.DescriptSeA method), 161
prod_force_virial() (deepmd.descriptor.se_a_ef.DescriptSeAEf method), 166
prod_force_virial() (deepmd.descriptor.se_r.DescriptSeR method), 173
prod_force_virial() (deepmd.descriptor.se_t.DescriptSeT method), 177
prod_virial() (in module deepmd.env.op_module), 323
prod_virial_grad() (in module deepmd.env.op_grads_module), 336
prod_virial_norot() (in module deepmd.env.op_module), 323
prod_virial_se_a() (in module deepmd.env.op_module), 324
prod_virial_se_a_grad() (in module

```


deepmd.env.op_grads_module), 336
 prod_virial_se_r() (in module
 deepmd.env.op_module), 324
 prod_virial_se_r_grad() (in module
 deepmd.env.op_grads_module), 336
 ProdEnvMatA() (in module deepmd.env.op_module),
 298
 ProdEnvMatAMix() (in module
 deepmd.env.op_module), 300
 ProdEnvMatANvnmdQuantize() (in module
 deepmd.env.op_module), 300
 ProdEnvMatR() (in module deepmd.env.op_module),
 301
 ProdForce() (in module deepmd.env.op_module),
 301
 ProdForceGrad() (in module
 deepmd.env.op_grads_module), 332
 ProdForceNorot() (in module
 deepmd.env.op_module), 302
 ProdForceSeA() (in module
 deepmd.env.op_module), 302
 ProdForceSeAGPUExecuteFunctor (C++ struct), 371
 ProdForceSeAGPUExecuteFunctor::operator()
 (C++ function), 371
 ProdForceSeAGrad() (in module
 deepmd.env.op_grads_module), 332
 ProdForceSeR() (in module
 deepmd.env.op_module), 302
 ProdForceSeRGPUEExecuteFunctor (C++ struct), 372
 ProdForceSeRGPUEExecuteFunctor::operator()
 (C++ function), 372
 ProdForceSeRGrad() (in module
 deepmd.env.op_grads_module), 332
 ProdVirial() (in module deepmd.env.op_module),
 303
 ProdVirialGrad() (in module
 deepmd.env.op_grads_module), 333
 ProdVirialNorot() (in module
 deepmd.env.op_module), 303
 ProdVirialSeA() (in module
 deepmd.env.op_module), 303
 ProdVirialSeAGPUExecuteFunctor (C++ struct),
 372
 ProdVirialSeAGPUExecuteFunctor::operator()
 (C++ function), 372
 ProdVirialSeAGrad() (in module
 deepmd.env.op_grads_module), 333
 ProdVirialSeR() (in module
 deepmd.env.op_module), 304
 ProdVirialSeRGPUEExecuteFunctor (C++ struct),
 372
 ProdVirialSeRGPUEExecuteFunctor::operator()
 (C++ function), 372
 ProdVirialSeRGrad() (in module

deepmd.env.op_grads_module), 334
 profiling:
 training/profiling (Argument), 75
 profiling_file:
 training/profiling_file (Argument), 75

Q

qc() (deepmd.nvnmd.utils.Encode method), 240
 qc() (deepmd.nvnmd.utils.encode.Encode method),
 246
 qf() (deepmd.nvnmd.utils.Encode method), 240
 qf() (deepmd.nvnmd.utils.encode.Encode method),
 246
 qf() (in module deepmd.nvnmd.utils.network), 249
 qq() (deepmd.nvnmd.entrypoints.maptable.MapTable
 method), 237
 qq() (deepmd.nvnmd.entrypoints.MapTable
 method), 234
 qr() (deepmd.nvnmd.utils.Encode method), 240
 qr() (deepmd.nvnmd.utils.encode.Encode method),
 246
 qr() (in module deepmd.nvnmd.utils.network), 249
 quantize_descriptor:
 nvnmd/quantize_descriptor (Argument), 77
 quantize_fitting_net:
 nvnmd/quantize_fitting_net (Argument), 77
 quantize_nvnmd() (in module
 deepmd.env.op_module), 325
 QuantizeNvnmd() (in module
 deepmd.env.op_module), 304

R

random() (in module deepmd.utils.random), 278
 rcond:
 model/fitting_net[ener]/rcond (Argument),
 64
 rcut:
 model/descriptor[loc_frame]/rcut
 (Argument), 53
 model/descriptor[se_a_tpe]/rcut
 (Argument), 57
 model/descriptor[se_atten]/rcut
 (Argument), 61
 model/descriptor[se_e2_a]/rcut
 (Argument), 54
 model/descriptor[se_e2_r]/rcut
 (Argument), 59
 model/descriptor[se_e3]/rcut (Argument),
 56
 rcut_smth:
 model/descriptor[se_a_tpe]/rcut_smth
 (Argument), 57
 model/descriptor[se_atten]/rcut_smth
 (Argument), 61

`model/descriptor[se_e2_a]/rcut_smth`
 (Argument), 54
`model/descriptor[se_e2_r]/rcut_smth`
 (Argument), 59
`model/descriptor[se_e3]/rcut_smth`
 (Argument), 56
`reduce()` (deepmd.utils.data.DeepmdData method),
 261
`reduce()` (deepmd.utils.data_system.DeepmdDataSystem
 method), 265
`register()` (deepmd.descriptor.descriptor.Descriptor
 static method), 149
`register()` (deepmd.utils.argcheck.ArgsPlugin
 method), 252
`register()` (deepmd.utils.plugin.Plugin method),
 277
`reinit()` (deepmd.utils.pair_tab.PairTab method),
 272
`relative_f:`
`loss[ener]/relative_f` (Argument), 71
`remove_decay_rate()` (in module
 deepmd.utils.compat), 256
`replace_path()` (in module
 deepmd.nvnmd.entrypoints.train), 237
`reset_default_tf_session_config()` (in module
 deepmd.env), 289
`reset_get_batch()` (deepmd.utils.data.DeepmdData
 method), 262
`reset_iter()` (deepmd.utils.data.DataSets method),
 259
`resnet_dt:`
`model/descriptor[se_a_tpe]/resnet_dt`
 (Argument), 58
`model/descriptor[se_atten]/resnet_dt`
 (Argument), 62
`model/descriptor[se_e2_a]/resnet_dt`
 (Argument), 55
`model/descriptor[se_e2_r]/resnet_dt`
 (Argument), 60
`model/descriptor[se_e3]/resnet_dt`
 (Argument), 56
`model/fitting_net[dipole]/resnet_dt`
 (Argument), 65
`model/fitting_net[ener]/resnet_dt`
 (Argument), 64
`model/fitting_net[polar]/resnet_dt`
 (Argument), 66
`model/type_embedding/resnet_dt`
 (Argument), 52
`restore_descriptor:`
`nvnmd/restore_descriptor` (Argument), 77
`restore_fitting_net:`
`nvnmd/restore_fitting_net` (Argument), 77
`reverse_bin()` (deepmd.nvnmd.utils.Encode
 method), 240
`reverse_bin()` (deepmd.nvnmd.utils.encode.Encode
 method), 246
`reverse_map()` (deepmd.DeepEval static method),
 140
`reverse_map()` (deepmd.infer.deep_eval.DeepEval
 static method), 209
`reverse_map()` (deepmd.infer.DeepEval static
 method), 195
`rglob()` (deepmd.utils.path.DPH5Path method), 274
`rglob()` (deepmd.utils.path.DPOSPath method), 275
`rglob()` (deepmd.utils.path.DPPPath method), 276
`run_s2G()` (deepmd.nvnmd.entrypoints.mapt.MapTable
 method), 237
`run_s2G()` (deepmd.nvnmd.entrypoints.MapTable
 method), 234
`run_sess()` (in module deepmd.utils.sess), 279
`run_u2s()` (deepmd.nvnmd.entrypoints.mapt.MapTable
 method), 237
`run_u2s()` (deepmd.nvnmd.entrypoints.MapTable
 method), 234
`RunOptions` (class in deepmd.train.run_options), 250

S

`safe_cast_tensor()` (in module deepmd.common),
 288
`save()` (deepmd.nvnmd.utils.config.NvnmdConfig
 method), 244
`save()` (deepmd.nvnmd.utils.fio.FioBin method), 247
`save()` (deepmd.nvnmd.utils.fio.FioDic method), 247
`save()` (deepmd.nvnmd.utils.fio.FioJsonDic
 method), 248
`save()` (deepmd.nvnmd.utils.fio.FioNpyDic
 method), 248
`save()` (deepmd.nvnmd.utils.fio.FioTxt method),
 248
`save()` (deepmd.nvnmd.utils.FioBin method), 241
`save()` (deepmd.nvnmd.utils.FioDic method), 241
`save()` (deepmd.nvnmd.utils.FioTxt method), 241
`save_checkpoint()` (deepmd.train.trainer.DPTrainer
 method), 252
`save_ckpt:`
`training/save_ckpt` (Argument), 75
`save_compressed()` (deepmd.train.trainer.DPTrainer
 method), 252
`save_freq:`
`training/save_freq` (Argument), 75
`save_weight()` (in module
 deepmd.nvnmd.entrypoints), 235
`save_weight()` (in module
 deepmd.nvnmd.entrypoints.freeze), 235
`scale:`
`model/fitting_net[polar]/scale`
 (Argument), 66

scale_by_worker:
 learning_rate/scale_by_worker (Argument), 68
 seed() (in module deepmd.utils.random), 278
 seed:
 model/descriptor[se_a_tpe]/seed (Argument), 58
 model/descriptor[se_atten]/seed (Argument), 62
 model/descriptor[se_e2_a]/seed (Argument), 55
 model/descriptor[se_e2_r]/seed (Argument), 60
 model/descriptor[se_e3]/seed (Argument), 57
 model/fitting_net[dipole]/seed (Argument), 65
 model/fitting_net[ener]/seed (Argument), 65
 model/fitting_net[polar]/seed (Argument), 67
 model/type_embedding/seed (Argument), 52
 training/seed (Argument), 74
 sel:
 model/descriptor[se_a_tpe]/sel (Argument), 57
 model/descriptor[se_atten]/sel (Argument), 61
 model/descriptor[se_e2_a]/sel (Argument), 54
 model/descriptor[se_e2_r]/sel (Argument), 59
 model/descriptor[se_e3]/sel (Argument), 55
 sel_a:
 model/descriptor[loc_frame]/sel_a (Argument), 53
 sel_r:
 model/descriptor[loc_frame]/sel_r (Argument), 53
 sel_type:
 model/fitting_net[dipole]/sel_type (Argument), 65
 model/fitting_net[polar]/sel_type (Argument), 66
 select_idx_map() (in module deepmd.common), 288
 sess (deepmd.DeepEval property), 140
 sess (deepmd.infer.deep_eval.DeepEval property), 209
 sess (deepmd.infer.DeepEval property), 195
 set_davg_zero:
 model/descriptor[se_a_tpe]/set_davg_zero (Argument), 58
 model/descriptor[se_atten]/set_davg_zero (Argument), 62
 model/descriptor[se_e2_a]/set_davg_zero (Argument), 55
 model/descriptor[se_e2_r]/set_davg_zero (Argument), 60
 model/descriptor[se_e3]/set_davg_zero (Argument), 57
 set_log_handles() (in module deepmd.loggers), 220
 set_log_handles() (in module deepmd.loggers.loggers), 221
 set_numb_batch() (deepmd.utils.data.DataSets method), 259
 set_prefix:
 training/training_data/set_prefix (Argument), 72
 training/validation_data/set_prefix (Argument), 73
 set_sys_probs() (deepmd.utils.data_system.DeepmdDataSystem method), 265
 shift_diag:
 model/fitting_net[polar]/shift_diag (Argument), 66
 shuffle() (in module deepmd.utils.random), 279
 SimulationRegion (C++ class), 374
 SimulationRegion::~SimulationRegion (C++ function), 374
 SimulationRegion::affineTransform (C++ function), 374
 SimulationRegion::backup (C++ function), 374
 SimulationRegion::compactIndex (C++ function), 375
 SimulationRegion::computeShiftVec (C++ function), 375
 SimulationRegion::DBOX_XX (C++ member), 376
 SimulationRegion::DBOX_YY (C++ member), 376
 SimulationRegion::DBOX_ZZ (C++ member), 376
 SimulationRegion::diffNearestNeighbor (C++ function), 375
 SimulationRegion::getBoxOrigin (C++ function), 374
 SimulationRegion::getBoxTensor (C++ function), 374
 SimulationRegion::getInterShiftVec (C++ function), 375
 SimulationRegion::getNullShiftIndex (C++ function), 375
 SimulationRegion::getNumShiftVec (C++ function), 375
 SimulationRegion::getRecBoxTensor (C++ function), 374
 SimulationRegion::getShiftIndex (C++ function), 375
 SimulationRegion::getShiftVec (C++ function), 374, 375

SimulationRegion::getShiftVecTotalSize (C++ function), 375
 SimulationRegion::getVolume (C++ function), 374
 SimulationRegion::index3to1 (C++ function), 376
 SimulationRegion::inter2Phys (C++ function), 374
 SimulationRegion::inter_shift_vec (C++ member), 376
 SimulationRegion::isPeriodic (C++ function), 374
 SimulationRegion::NBOX_XX (C++ member), 376
 SimulationRegion::NBOX_YY (C++ member), 376
 SimulationRegion::NBOX_ZZ (C++ member), 376
 SimulationRegion::phys2Inter (C++ function), 374
 SimulationRegion::recover (C++ function), 374
 SimulationRegion::reinitBox (C++ function), 374
 SimulationRegion::reinitOrigin (C++ function), 374
 SimulationRegion::shift_info_size (C++ member), 376
 SimulationRegion::shift_vec (C++ member), 376
 SimulationRegion::shift_vec_size (C++ member), 376
 SimulationRegion::shiftCoord (C++ function), 375
 SimulationRegion::SimulationRegion (C++ function), 374
 SimulationRegion::SPACENDIM (C++ member), 376
 SimulationRegion::toFaceDistance (C++ function), 374
 smin_alpha:
 model/smin_alpha (Argument), 51
 soft_min_force() (in module deepmd.env.op_module), 325
 soft_min_force_grad() (in module deepmd.env.op_grads_module), 337
 soft_min_switch() (in module deepmd.env.op_module), 325
 soft_min_virial() (in module deepmd.env.op_module), 326
 soft_min_virial_grad() (in module deepmd.env.op_grads_module), 337
 SoftMinForce() (in module deepmd.env.op_module), 305
 SoftMinForceGrad() (in module deepmd.env.op_grads_module), 334
 SoftMinSwitch() (in module deepmd.env.op_module), 305
 SoftMinVirial() (in module deepmd.env.op_module), 305
 SoftMinVirialGrad() (in module deepmd.env.op_grads_module), 334
 sort_input() (deepmd.DeepEval static method), 140
 sort_input() (deepmd.infer.deep_eval.DeepEval static method), 209
 sort_input() (deepmd.infer.DeepEval static method), 195
 split_bin() (deepmd.nvnmd.utils.Encode method), 240
 split_bin() (deepmd.nvnmd.utils.encode.Encode method), 246
 SQRT_2_PI (C macro), 414
 start_lr() (deepmd.utils.learning_rate.LearningRateExp method), 270
 start_lr:
 learning_rate[exp]/start_lr (Argument), 69
 start_pref() (in module deepmd.utils.argcheck), 254
 start_pref_ae:
 loss[ener]/start_pref_ae (Argument), 70
 start_pref_e:
 loss[ener]/start_pref_e (Argument), 69
 start_pref_f:
 loss[ener]/start_pref_f (Argument), 70
 start_pref_pf:
 loss[ener]/start_pref_pf (Argument), 71
 start_pref_v:
 loss[ener]/start_pref_v (Argument), 70
 stats_energy() (deepmd.utils.data.DataSets method), 259
 stop_lr:
 learning_rate[exp]/stop_lr (Argument), 69
 sw_rmax:
 model/sw_rmax (Argument), 51
 sw_rmin:
 model/sw_rmin (Argument), 51
 sys_charge_map:
 model/modifier[dipole_charge]/sys_charge_map (Argument), 67
 sys_probs:
 training/training_data/sys_probs (Argument), 73
 training/validation_data/sys_probs (Argument), 74
 systems:
 training/training_data/systems (Argument), 72
 training/validation_data/systems (Argument), 73
T
 table_config:

model/compress[se_e2_a]/table_config (Argument), 68	TabulateFusionSeRGradGrad() (in module deepmd.env.op_module), 308
tabulate_fusion() (in module deepmd.env.op_module), 326	TabulateFusionSeT() (in module deepmd.env.op_module), 308
tabulate_fusion_grad() (in module deepmd.env.op_module), 326	TabulateFusionSeTGrad() (in module deepmd.env.op_module), 309
tabulate_fusion_grad_grad() (in module deepmd.env.op_module), 327	TabulateFusionSeTGradGrad() (in module deepmd.env.op_module), 309
tabulate_fusion_se_a() (in module deepmd.env.op_module), 327	tanh2() (in module deepmd.nvnmd.utils.network), 249
tabulate_fusion_se_a_grad() (in module deepmd.env.op_module), 327	tanh2_nvnmd() (in module deepmd.env.op_module), 330
tabulate_fusion_se_a_grad_grad() (in module deepmd.env.op_module), 328	Tanh2Nvnmd() (in module deepmd.env.op_module), 310
tabulate_fusion_se_r() (in module deepmd.env.op_module), 328	tanh4() (in module deepmd.nvnmd.utils.network), 249
tabulate_fusion_se_r_grad() (in module deepmd.env.op_module), 328	tanh4_nvnmd() (in module deepmd.env.op_module), 330
tabulate_fusion_se_r_grad_grad() (in module deepmd.env.op_module), 329	Tanh4Nvnmd() (in module deepmd.env.op_module), 310
tabulate_fusion_se_t() (in module deepmd.env.op_module), 329	tensorboard:
tabulate_fusion_se_t_grad() (in module deepmd.env.op_module), 329	training/tensorboard (Argument), 75
tabulate_fusion_se_t_grad_grad() (in module deepmd.env.op_module), 330	tensorboard_freq:
TabulateCheckerGPUExecuteFunctor (C++ struct), 373	training/tensorboard_freq (Argument), 76
TabulateCheckerGPUExecuteFunctor::operator() (C++ function), 373	tensorboard_log_dir:
TabulateFusion() (in module deepmd.env.op_module), 306	training/tensorboard_log_dir (Argument), 76
TabulateFusionGPUExecuteFunctor (C++ struct), 373	TensorLoss (class in deepmd.loss.tensor), 225
TabulateFusionGPUExecuteFunctor::operator() (C++ function), 373	TensorModel (class in deepmd.model.tensor), 229
TabulateFusionGrad() (in module deepmd.env.op_module), 306	tensors (deepmd.infer.deep_tensor.DeepTensor attribute), 217
TabulateFusionGradGPUExecuteFunctor (C++ struct), 373	test() (in module deepmd.entrypoints), 179
TabulateFusionGradGPUExecuteFunctor::operator() (C++ function), 374	test() (in module deepmd.entrypoints.test), 183
TabulateFusionGradGrad() (in module deepmd.env.op_module), 306	time_training:
TabulateFusionSeA() (in module deepmd.env.op_module), 307	training/time_training (Argument), 75
TabulateFusionSeAGrad() (in module deepmd.env.op_module), 307	TPB (C macro), 415
TabulateFusionSeAGradGrad() (in module deepmd.env.op_module), 307	train() (deepmd.train.trainer.DPTrainer method), 252
TabulateFusionSeR() (in module deepmd.env.op_module), 308	train() (in module deepmd.entrypoints.train), 183
TabulateFusionSeRGrad() (in module deepmd.env.op_module), 308	train_dp() (in module deepmd.entrypoints), 179
	train_nvnmd() (in module deepmd.nvnmd.entrypoints.train), 237
	trainable:
	model/descriptor[se_a_tpe]/trainable (Argument), 58
	model/descriptor[se_atten]/trainable (Argument), 62
	model/descriptor[se_e2_a]/trainable (Argument), 55
	model/descriptor[se_e2_r]/trainable (Argument), 60
	model/descriptor[se_e3]/trainable (Argument), 57
	model/fitting_net[ener]/trainable


```

        (Argument), 64
    model/type_embedding/trainable
        (Argument), 52
    training (Argument)
        training:, 71
    training/disp_file (Argument)
        disp_file:, 75
    training/disp_freq (Argument)
        disp_freq:, 75
    training/disp_training (Argument)
        disp_training:, 75
    training/enable_profiler (Argument)
        enable_profiler:, 75
    training/mixed_precision (Argument)
        mixed_precision:, 74
    training/mixed_precision/compute_prec
        (Argument)
        compute_prec:, 74
    training/mixed_precision/output_prec
        (Argument)
        output_prec:, 74
    training/numb_steps (Argument)
        numb_steps:, 74
    training/profiling (Argument)
        profiling:, 75
    training/profiling_file (Argument)
        profiling_file:, 75
    training/save_ckpt (Argument)
        save_ckpt:, 75
    training/save_freq (Argument)
        save_freq:, 75
    training/seed (Argument)
        seed:, 74
    training/tensorboard (Argument)
        tensorboard:, 75
    training/tensorboard_freq (Argument)
        tensorboard_freq:, 76
    training/tensorboard_log_dir (Argument)
        tensorboard_log_dir:, 76
    training/time_training (Argument)
        time_training:, 75
    training/training_data (Argument)
        training_data:, 72
    training/training_data/auto_prob (Argument)
        auto_prob:, 72
    training/training_data/batch_size (Argument)
        batch_size:, 72
    training/training_data/set_prefix (Argument)
        set_prefix:, 72
    training/training_data/sys_probs (Argument)
        sys_probs:, 73
    training/training_data/systems (Argument)
        systems:, 72
    training/validation_data (Argument)
        validation_data:, 73
    training/validation_data/auto_prob
        (Argument)
        auto_prob:, 73
    training/validation_data/batch_size
        (Argument)
        batch_size:, 73
    training/validation_data/numb_btch
        (Argument)
        numb_btch:, 74
    training/validation_data/set_prefix
        (Argument)
        set_prefix:, 73
    training/validation_data/sys_probs
        (Argument)
        sys_probs:, 74
    training/validation_data/systems (Argument)
        systems:, 73
    training:
        training (Argument), 71
    training_args() (in module
        deepmd.utils.argcheck), 254
    training_data:
        training/training_data (Argument), 72
    training_data_args() (in module
        deepmd.utils.argcheck), 254
    transfer() (in module deepmd.entrypoints), 179
    transfer() (in module
        deepmd.entrypoints.transfer), 184
    type:
        learning_rate/type (Argument), 69
        loss/type (Argument), 69
        model/compress/type (Argument), 68
        model/descriptor/type (Argument), 52
        model/fitting_net/type (Argument), 63
        model/modifier/type (Argument), 67
    type_embedding:
        model/type_embedding (Argument), 51
    type_embedding_args() (in module
        deepmd.utils.argcheck), 254
    type_map:
        model/type_map (Argument), 50
    type_nchanl:
        model/descriptor[se_a_tpe]/type_nchanl
            (Argument), 59
    type_nlayer:
        model/descriptor[se_a_tpe]/type_nlayer
            (Argument), 59
    type_one_side:
        model/descriptor[se_a_tpe]/type_one_side
            (Argument), 58
        model/descriptor[se_atten]/type_one_side
            (Argument), 62

```

model/descriptor[se_e2_a]/type_one_side (Argument), 55
 model/descriptor[se_e2_r]/type_one_side (Argument), 60
 TypeEmbedNet (class in deepmd.utils.type_embed), 280

U

uint_64 (C++ type), 415
 unaggregated_dy2_dx() (in module deepmd.env.op_module), 330
 unaggregated_dy2_dx_s() (in module deepmd.env.op_module), 331
 unaggregated_dy_dx() (in module deepmd.env.op_module), 331
 unaggregated_dy_dx_s() (in module deepmd.env.op_module), 331
 UnaggregatedDy2Dx() (in module deepmd.env.op_module), 310
 UnaggregatedDy2DxS() (in module deepmd.env.op_module), 310
 UnaggregatedDyDx() (in module deepmd.env.op_module), 311
 UnaggregatedDyDxS() (in module deepmd.env.op_module), 311
 update() (deepmd.nvnmd.utils.fio.FioDic method), 247
 update() (deepmd.nvnmd.utils.FioDic method), 241
 update_deepmd_input() (in module deepmd.utils.compat), 256
 use_srtab:
 model/use_srtab (Argument), 51

V

valid_on_the_fly() (deepmd.train.trainer.DPTrainer method), 252
 validation_data:
 training/validation_data (Argument), 73
 validation_data_args() (in module deepmd.utils.argcheck), 254
 value() (deepmd.utils.learning_rate.LearningRateExp method), 270
 variable_summaries() (in module deepmd.utils.network), 272
 VariantABCMeta (class in deepmd.utils.plugin), 278
 VariantMeta (class in deepmd.utils.plugin), 278

W

weight_file:
 nvnmd/weight_file (Argument), 76
 weighted_average() (in module deepmd.utils.weight_avg), 282
 WFCFitting (class in deepmd.fit.wfc), 193
 WFCModel (class in deepmd.model.tensor), 230
 world_size (deepmd.train.run_options.RunOptions attribute), 251
 Wrap (class in deepmd.nvnmd.entrypoints), 234
 Wrap (class in deepmd.nvnmd.entrypoints.wrap), 237
 wrap() (deepmd.nvnmd.entrypoints.Wrap method), 234
 wrap() (deepmd.nvnmd.entrypoints.wrap.Wrap method), 238
 wrap() (in module deepmd.nvnmd.entrypoints.wrap), 238
 wrap_bias() (deepmd.nvnmd.entrypoints.Wrap method), 234
 wrap_bias() (deepmd.nvnmd.entrypoints.wrap.Wrap method), 238
 wrap_dscp() (deepmd.nvnmd.entrypoints.Wrap method), 234
 wrap_dscp() (deepmd.nvnmd.entrypoints.wrap.Wrap method), 238
 wrap_fitn() (deepmd.nvnmd.entrypoints.Wrap method), 235
 wrap_fitn() (deepmd.nvnmd.entrypoints.wrap.Wrap method), 238
 wrap_head() (deepmd.nvnmd.entrypoints.Wrap method), 235
 wrap_head() (deepmd.nvnmd.entrypoints.wrap.Wrap method), 238
 wrap_map() (deepmd.nvnmd.entrypoints.Wrap method), 235
 wrap_map() (deepmd.nvnmd.entrypoints.wrap.Wrap method), 238
 wrap_weight() (deepmd.nvnmd.entrypoints.Wrap method), 235
 wrap_weight() (deepmd.nvnmd.entrypoints.wrap.Wrap method), 238
 write_model_devi_out() (in module deepmd.infer.model_devi), 220