DeePMD-kit

Deep Potential

May 26, 2021

USER GUIDE

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.

CHAPTER

ONE

INSTALLATION

1.1 Easy installation methods

There 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 run LAMMPS in parallel.

1.1.1 Install off-line packages

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

1.1.2 Install with conda

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

To install the CPU version:

```
conda install deepmd-kit=*=*cpu lammps-dp=*=*cpu -c deepmodeling
```

To install the GPU version containing CUDA 10.1:

```
conda install deepmd-kit=*=*gpu lammps-dp=*=*gpu -c deepmodeling
```

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:2.0.0_cpu
```

To pull the GPU version:

```
docker pull ghcr.io/deepmodeling/deepmd-kit:2.0.0_cuda10.1_gpu
```

1.2 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`
```

- *Install the python interaction*
  - *Install the Tensorflow's python interface*
  - *Install the DeePMD-kit's python interface*
- *Install the C++ interface*
  - *Install the Tensorflow's C++ interface*
  - *Install the DeePMD-kit's C++ interface*
- *Install LAMMPS's DeePMD-kit module*

1.2.1 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 the tensorflow's Python interface. The full instruction can be found on the tensorflow's official website. 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==2.3.0
```

It is notice 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==2.3.0
```

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.

Install the DeePMD-kit's python interface

Execute

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

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:
```

(continues on next page)

(continued from previous page)

```
  {train,freeze,test}
    train               train a model
    freeze              freeze the model
    test                test the model
```

1.2.2 Install the C++ interface

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

Install the Tensorflow's C++ interface

Check the compiler version on your machine

```
gcc --version
```

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 >= 4.9.

First the C++ interface of Tensorflow should be installed. It is noted that the version of Tensorflow should be in consistent with the python interface. You may follow the instruction to install the corresponding C++ interface.

Install the DeePMD-kit's C++ interface

Now goto 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. The DeePMD-kit will automatically detect if a CUDA tool-kit is available on your machine and build the GPU support accordingly. If you want to force the cmake to find CUDA tool-kit, you can speicify the key `USE_CUDA_TOOLKIT`,

```
cmake -DUSE_CUDA_TOOLKIT=true -DTENSORFLOW_ROOT=$tensorflow_root -DCMAKE_INSTALL_
↪PREFIX=$deepmd_root ..
```

and you may further asked to provide `CUDA_TOOLKIT_ROOT_DIR`. If the cmake has executed successfully, then

```
make
make install
```

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
$ ls $deepmd_root/lib
libdeepmd_ipi.so  libdeepmd_op.so  libdeepmd.so
```

1.2.3 Install LAMMPS's DeePMD-kit module

DeePMD-kit provide 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-DEEPMD` in the `build` directory. Now download the LAMMPS code (`29Oct2020` or later), and uncompress it:

```
cd /some/workspace
wget https://github.com/lammps/lammps/archive/stable_29Oct2020.tar.gz
tar xf stable_29Oct2020.tar.gz
```

The source code of LAMMPS is stored in directory `lammps-stable_29Oct2020`. Now go into the LAMMPS code and copy the DeePMD-kit module like this

```
cd lammps-stable_29Oct2020/src/
cp -r $deepmd_source_dir/source/build/USER-DEEPMD .
```

Now build LAMMPS

```
make yes-kspace
make yes-user-deepmd
make mpi -j4
```

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 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
```

CHAPTER

TWO

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. *Prepare data*
2. *Train a model*
   - *Write the input script*
   - *Training*
   - *Training analysis with Tensorboard*
3. *Freeze a model*
4. *Test a model*
5. *Compress a model*
6. *Model inference*
7. *Run MD*
   - *Run MD with LAMMPS*
   - *Run path-integral MD with i-PI*
   - *Use deep potential with ASE*
8. *Known limitations*

2.1 Prepare data

One needs to provide the following information to train a model: the atom type, the simulation box, the atom coordinate, the atom force, system energy and virial. A snapshot of a system that contains these information is called a **frame**. We use the following convention of units:

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

The frames of the system are stored in two formats. A raw file is a plain text file with each information item written in one file and one frame written on one line. The default files that provide box, coordinate, force, energy and virial are `box.raw`, `coord.raw`, `force.raw`, `energy.raw` and `virial.raw`, respectively. *We recommend you use these file names*. 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. The coordinate file `coord.raw` is organized similarly. In `box.raw`, the 9 components of the box vectors should be provided on each line. In `virial.raw`, the 9 components of the virial tensor should be provided on each line in the order `XX XY XZ YX YY YZ ZX ZY ZZ`. The number of lines of all raw files should be identical.

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
O H
```

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

The second format is the data sets of `numpy` binary data that are directly used by the training program. User can use the script `$deepmd_source_dir/data/raw/raw_to_set.sh` to convert the prepared raw files to data sets. 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. One do not need to take care of the binary data files in each of the `set.*` directories. The path containing `set.*` and `type.raw` is called a *system*.

2.1.1 Data preparation 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. One may follow the *example* of using `dpdata` to find out how to use it.

2.2 Train a model

2.2.1 Write the input script

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.

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`: Concate a list of descriptors to form a new descriptor.

The fitting of the following physical properties are supported

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

2.2.2 Training

The training can be invoked by

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

where `input.json` is the name of the input script. See the example for more details.

During the training, checkpoints will be written to files with prefix `save_ckpt` every `save_freq` training steps.

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:
```

(continues on next page)

(continued from previous page)

```
  -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-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`.

On some resources limited machines, one may want to 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=6
export TF_INTRA_OP_PARALLELISM_THREADS=3
export TF_INTER_OP_PARALLELISM_THREADS=2
dp train input.json
```

2.2.3 Training analysis with Tensorboard

If enbled in json/yaml input file DeePMD-kit will create log files which can be used to analyze training procedure with Tensorboard. For a short tutorial please read this *document*.

2.3 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`.

2.4 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
```

2.5 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 input.json -i graph.pb -o graph-compress.pb
```

where input.json denotes the original training input script, `-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] [-i INPUT] [-o OUTPUT] [-e EXTRAPOLATE] [-s STRIDE]
                   [-f FREQUENCY] [-d FOLDER]
                   INPUT

positional arguments:
  INPUT                 The input parameter file in json or yaml format, which
                        should be consistent with the original model parameter
                        file

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        The original frozen model, which will be compressed by
                        the deepmd-kit
  -o OUTPUT, --output OUTPUT
                        The compressed model
```

(continues on next page)

(continued from previous page)

```
  -e EXTRAPOLATE, --extrapolate EXTRAPOLATE
                        The scale of model extrapolation
  -s STRIDE, --stride STRIDE
                        The uniform stride of tabulation's first table, the
                        second table will use 10 * stride as it's uniform
                        stride
  -f FREQUENCY, --frequency FREQUENCY
                        The frequency of tabulation overflow check(If the
                        input environment matrix overflow the first or second
                        table range). By default do not check the overflow
  -d FOLDER, --folder FOLDER
                        path to checkpoint folder
```

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. 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. 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 method requires that the version of DeePMD-kit used in original model generation should be 1.3 or above. If one has a frozen 1.2 model, one can first use the convenient conversion interface of DeePMD-kit-v1.2.4 to get a 1.3 executable model.(eg: `dp convert-to-1.3 -i frozen_1.2.pb -o frozen_1.3.pb`)

2.6 Model inference

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

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

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

2.7 Run MD

2.7.1 Run MD with LAMMPS

Include deepmd in the pair_style

Syntax

```
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 the model deviation will be computed
- 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 evalulate the consistency of the force predictions from multiple models. By default, only the maximal, minimal and averge 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

```
           |Df_i|
Ef_i = -------------
       |f_i| + level
```

where `Df_i` is the absolute model deviation of the force on atom `i`, `|f_i|` is the norm of the the force and `level` is provided as the parameter of the keyword `relative`.

Restrictions

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

Long-range interaction

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

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

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

2.7.2 Run path-integral MD with i-PI

The i-PI works in a client-server model. The i-PI provides the server for integrating the replica positions of atoms, while the DeePMD-kit provides a client named dp_ipi that computes the interactions (including energy, force and virial). The server and client communicates via the Unix domain socket or the Internet socket. The client can be started by

```
$ 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 **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**.

2.7.3 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())
```

2.8 Known limitations

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.

CHAPTER

THREE

TENSORBOARD USAGE

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

3.1 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 viriables**

3.2 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" true in training subsection will enable the tensorboard data analysis. eg. **water_se_a.json**.

```
    "training" : {
        "systems":          ["../data/"],
        "set_prefix":          "set",
        "stop_batch":          1000000,
        "batch_size":          1,

        "seed":                  1,

        "_comment": " display and restart",
        "_comment": " frequencies counted in batch",
        "disp_file":         "lcurve.out",
        "disp_freq":         100,
        "numb_test":         10,
        "save_freq":         1000,
        "save_ckpt":         "model.ckpt",
        "load_ckpt":         "model.ckpt",
        "disp_training":true,
        "time_training":true,
        "tensorboard":         true,
        "tensorboard_log_dir":"log",
        "profiling":         false,
```

(continues on next page)

(continued from previous page)

```
        "profiling_file":"timeline.json",
        "_comment":         "that's all"
    }
```

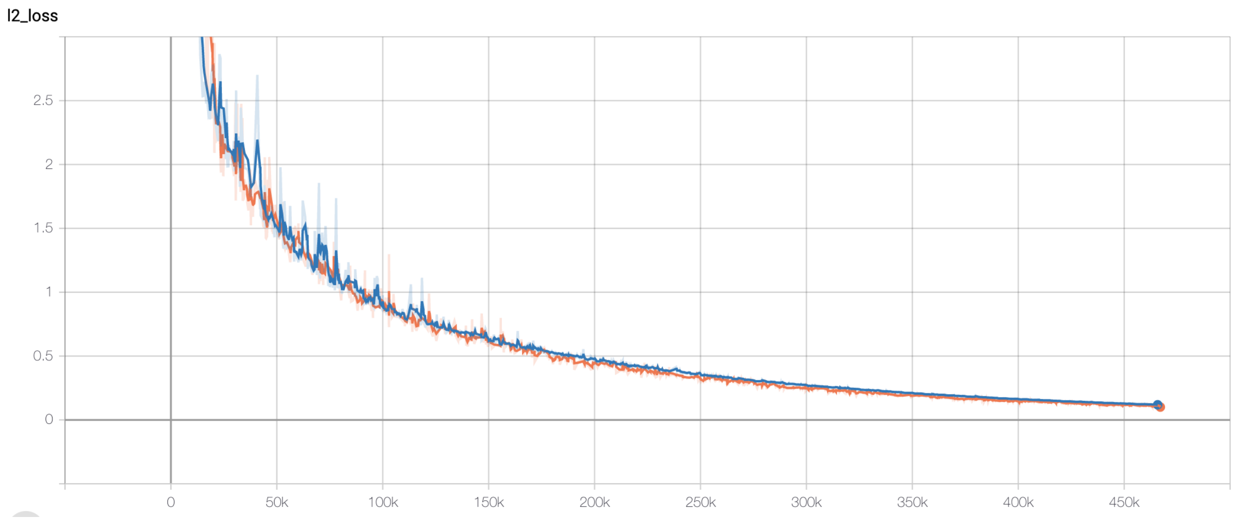
Once you have event files, run TensorBoard and provide the log directory. This should print that TensorBoard has started. Next, connect to http://tensorboard_server_ip:6006.

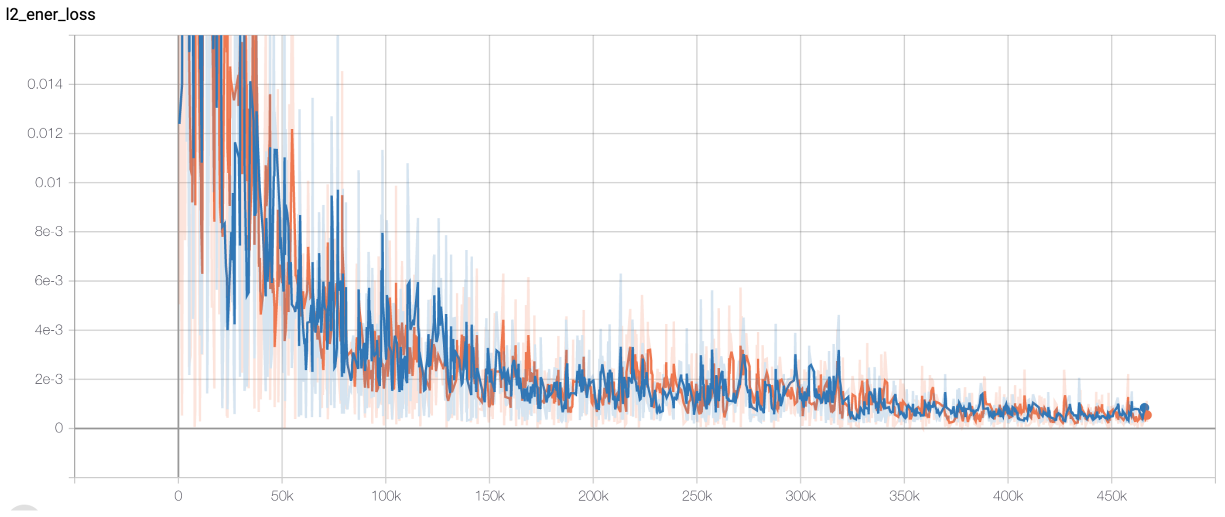
TensorBoard requires a logdir to read logs from. For info on configuring TensorBoard, run tensorboard –help. One can easily change the log name with "tensorboard_log_dir".

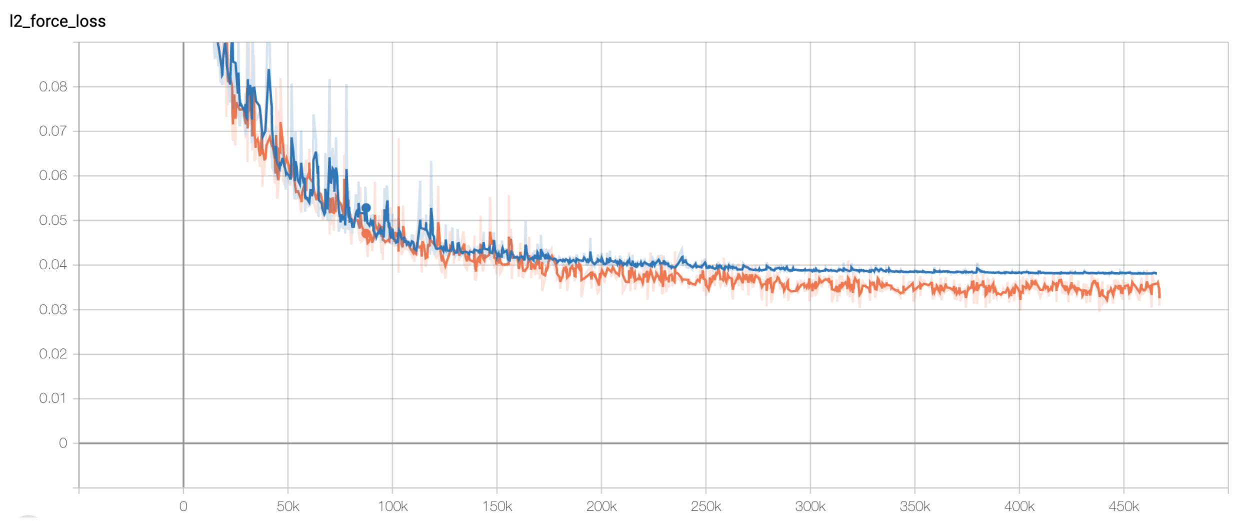
```
tensorboard --logdir path/to/logs
```

3.3 Examples

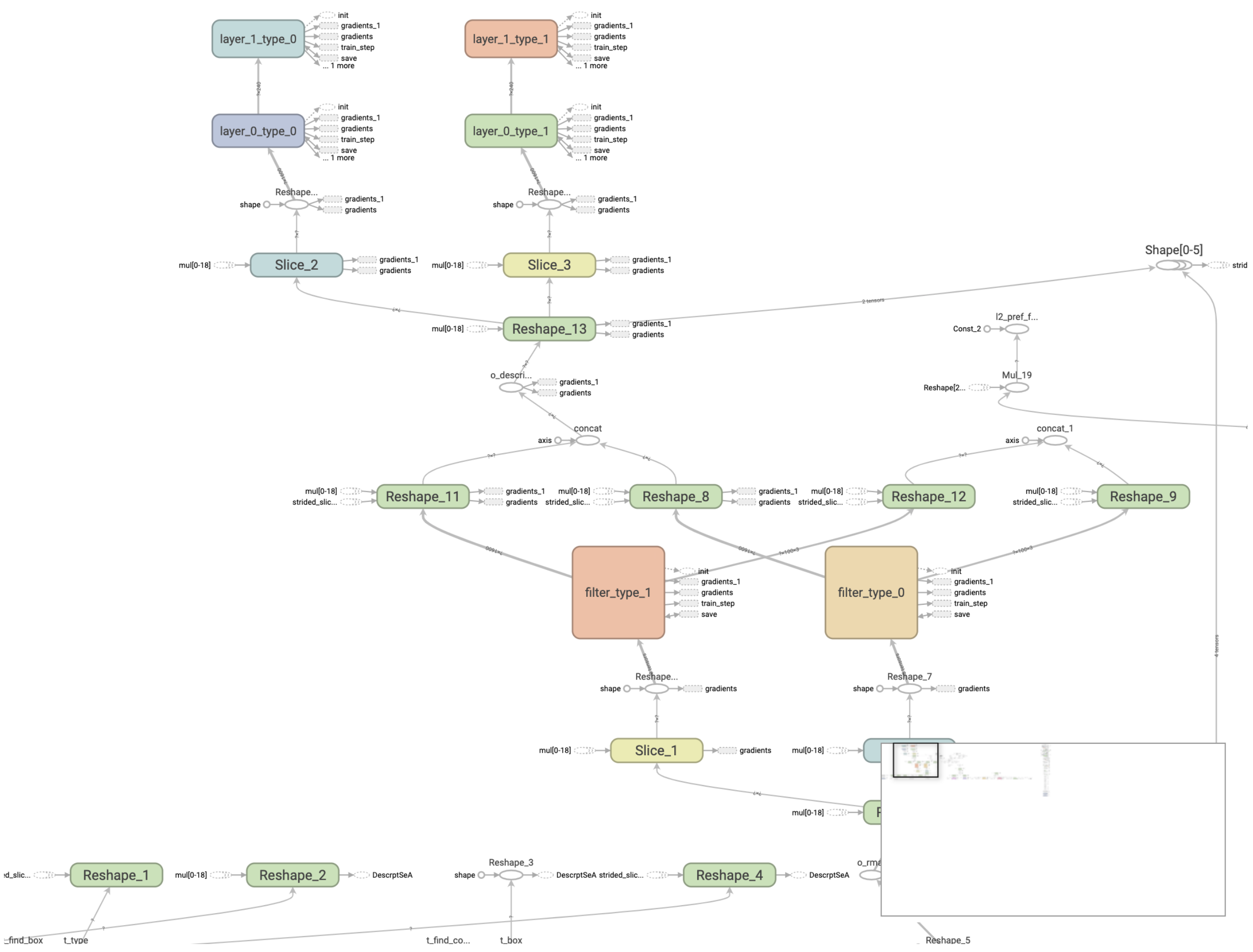
3.3.1 Tracking and visualizing loss metrics(red:train, blue:test)



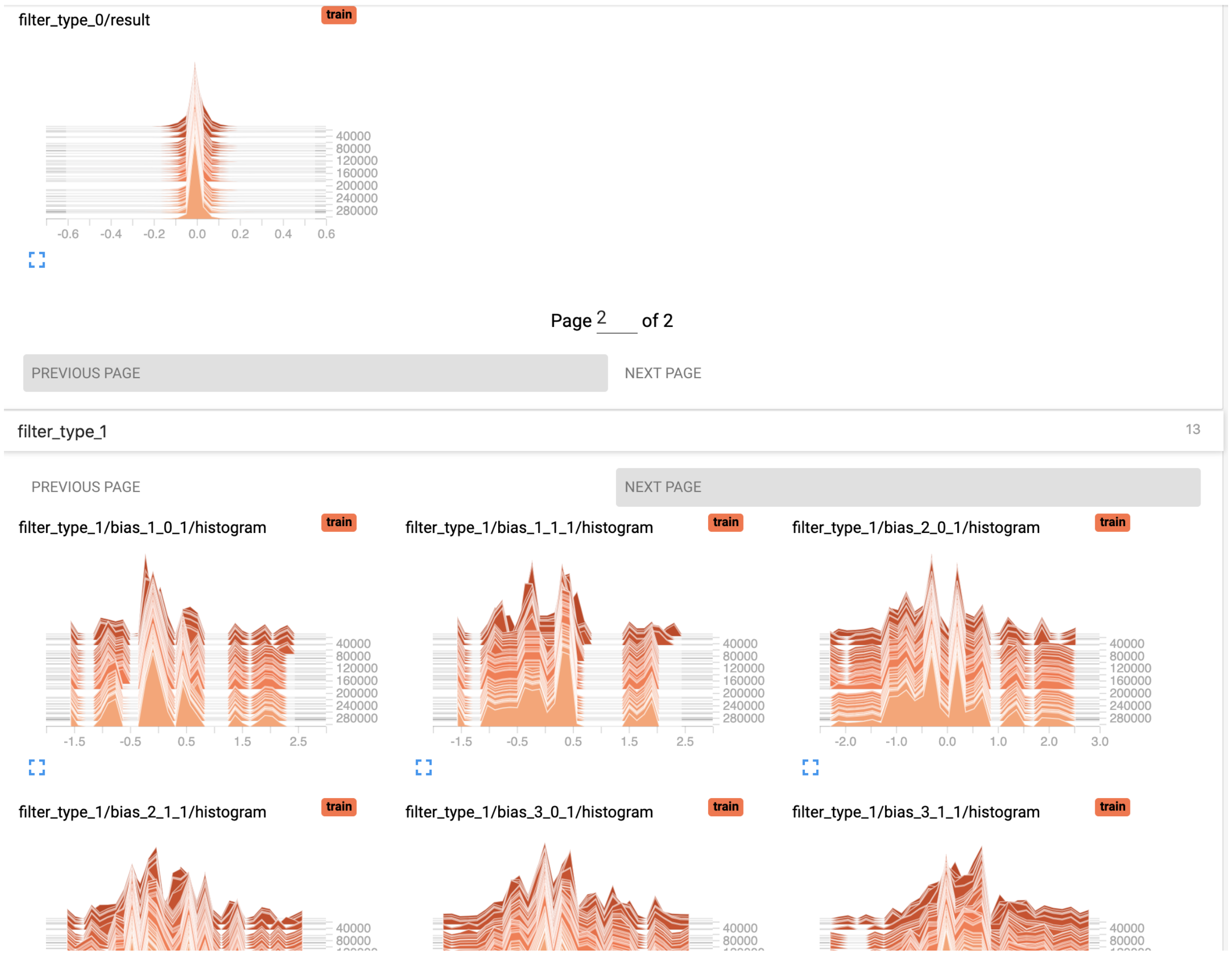


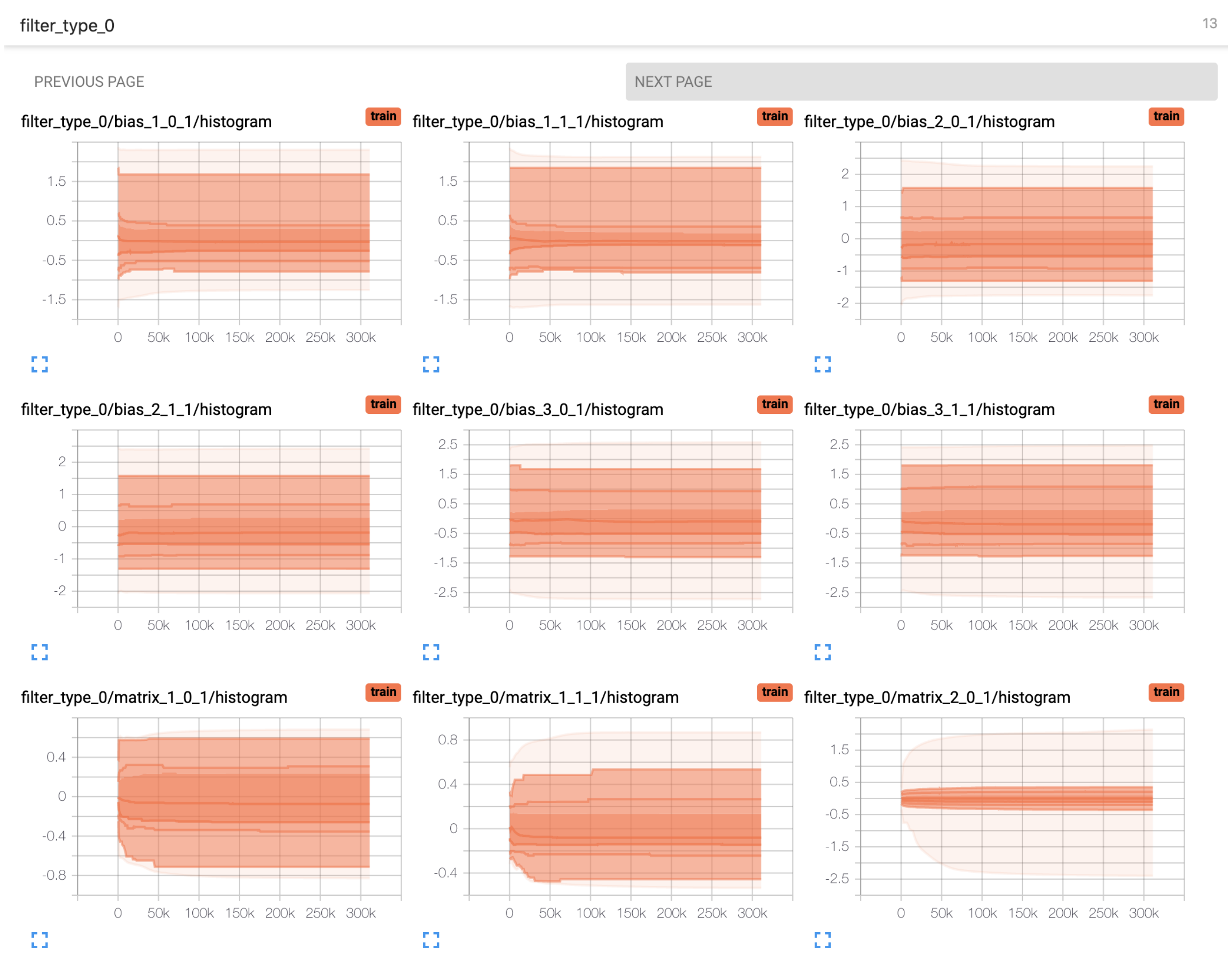


3.3.2 Visualizing deepmd-kit model graph



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


filter_type_0/result
train
40000
80000
120000
160000
200000
240000
280000
-0.6
-0.4
-0.2
0.0
0.2
0.4
0.6
Page 2 of 2
PREVIOUS PAGE
NEXT PAGE
filter_type_1
13
PREVIOUS PAGE
NEXT PAGE
filter_type_1/bias_1_0_1/histogram
filter_type_1/bias_1_1_1/histogram
filter_type_1/bias_2_0_1/histogram
filter_type_1/bias_2_1_1/histogram
filter_type_1/bias_3_0_1/histogram
filter_type_1/bias_3_1_1/histogram
-1.5
-0.5
0.5
1.5
2.5
-2.0
-1.0
0.0
1.0
2.0
3.0


filter_type_0

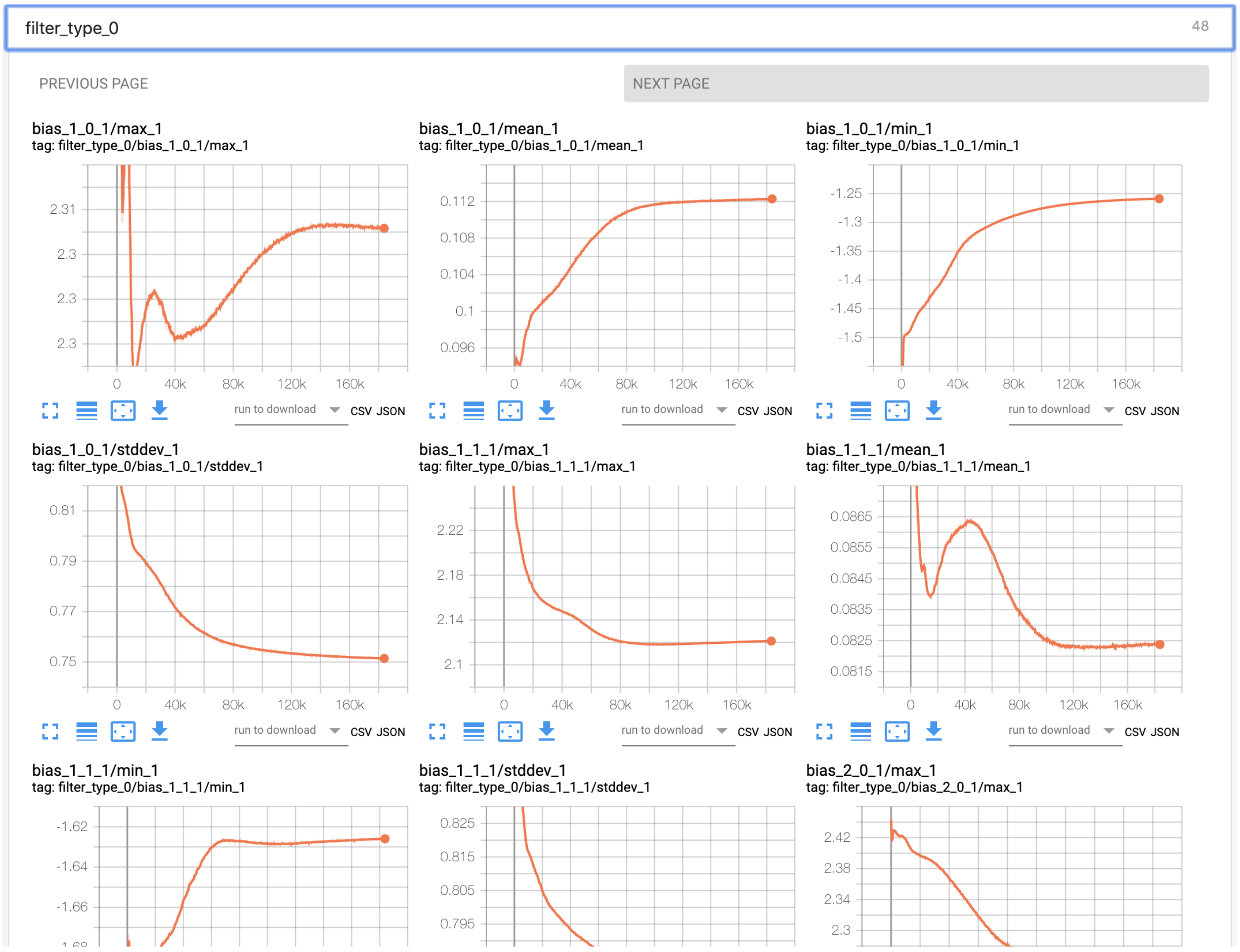
13

PREVIOUS PAGE

NEXT PAGE

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

3.3.4 Viewing summaries of trainable variables


filter_type_0
48
PREVIOUS PAGE
NEXT PAGE
bias_1_0_1/max_1
tag: filter_type_0/bias_1_0_1/max_1
bias_1_0_1/mean_1
tag: filter_type_0/bias_1_0_1/mean_1
bias_1_0_1/min_1
tag: filter_type_0/bias_1_0_1/min_1
bias_1_0_1/stddev_1
tag: filter_type_0/bias_1_0_1/stddev_1
bias_1_1_1/max_1
tag: filter_type_0/bias_1_1_1/max_1
bias_1_1_1/mean_1
tag: filter_type_0/bias_1_1_1/mean_1
bias_1_1_1/min_1
tag: filter_type_0/bias_1_1_1/min_1
bias_1_1_1/stddev_1
tag: filter_type_0/bias_1_1_1/stddev_1
bias_2_0_1/max_1
tag: filter_type_0/bias_2_0_1/max_1
run to download
CSV JSON

3.4 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.

CHAPTER FOUR

TROUBLESHOOTING

In consequence of various differences of computers or systems, problems may occur. Some common circumstances are listed as follows. If other unexpected problems occur, you're welcome to contact us for help.

- Installation
- The temperature undulates violently during early stages of MD
- MD: cannot run LAMMPS after installing a new version of DeePMD-kit
- Model compatability
- Do we need to set rcut < half boxsize ?

CHAPTER

FIVE

DATA

In this example we will convert the DFT labeled data stored in VASP `OUTCAR` format into the data format used by DeePMD-kit. The example `OUTCAR` can be found in the directory.

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

5.1 Definition

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 short 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.

5.2 Data conversion

It is conveninent to use dpdata to convert data generated by DFT packages to the data format used by DeePMD-kit.

To install one can execute

```
pip install dpdata
```

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

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

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

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

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

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

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

CHAPTER

SIX

TRAINING PARAMETERS

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.

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 columes. The first colume is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

smin_alpha:

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

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

sw_rmin:

type: `float`, optional

argument path: `model/sw_rmin`

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

sw_rmax:

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

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

type_embedding:

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

The type embedding.

neuron:

type: `list`, optional, default: `[2, 4, 8]`
argument path: `model/type_embedding/neuron`

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

activation_function:

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

The activation function in the embedding net. Supported activation functions are "relu", "relu6", "softplus", "sigmoid", "tanh", "gelu".

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: `float64`
argument path: `model/type_embedding/precision`

The precision of the embedding net parameters, supported options are "default", "float16", "float32", "float64".

trainable:

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

If the parameters in the embedding net are trainable

seed:

type: `int` | `NoneType`, optional
argument path: `model/type_embedding/seed`

Random seed for parameter initialization

descriptor:

type: `dict`
argument path: `model/descriptor`

The descriptor of atomic environment.

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

type:

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

The type of the descritpor. See explanation below.

- *loc_frame*: Defines a local frame at each atom, and the compute the descriptor as local coordinates under this frame.
- *se_e2_a*: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor.
- *se_e2_r*: Used by the smooth edition of Deep Potential. Only the distance between atoms is used to construct the descriptor.
- *se_e3*: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Three-body embedding will be used by this descriptor.
- *se_a_tpe*: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Type embedding will be used by this descriptor.
- *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`
argument path: `model/descriptor[se_e2_a]/sel`

A list of integers. 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.

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`
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”.

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: `float64`
argument path: `model/descriptor[se_e2_a]/precision`

The precision of the embedding net parameters, supported options are "default", "float16", "float32", "float64".

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 types

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_e2_r` (or its alias `se_r`):

sel:

type: `list`
argument path: `model/descriptor[se_e2_r]/sel`

A list of integers. 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.

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”.

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: `float64`
argument path: `model/descriptor[se_e2_r]/precision`

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

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 types

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 `se_e3` (or its aliases `se_at`, `se_a_3be`, `se_t`):

sel:

type: `list`
argument path: `model/descriptor[se_e3]/sel`

A list of integers. 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.

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”.

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: `float64`
argument path: `model/descriptor[se_e3]/precision`

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

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

A list of integers. 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.

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`
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”.

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: `float64`
argument path: `model/descriptor[se_a_tpe]/precision`

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

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 types

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 `hybrid`:

list:

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

A list of descriptor definitions

fitting_net:

type: `dict`
argument path: `model/fitting_net`

The fitting of physical properties.

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

type:

type: `str` (flag key), default: `ener`
argument path: `model/fitting_net/type`
possible choices: *`ener`*, *`dipole`*, *`polar`*, *`global_polar`*

The type of the fitting. See explanation below.

- *ener*: Fit an energy model (potential energy surface).
- *dipole*: Fit an atomic dipole model. Atomic dipole labels for all the selected atoms (see *sel_type*) should be provided by *dipole.npy* in each data system. The file has number of frames lines and 3 times of number of selected atoms columns.
- *polar*: Fit an atomic polarizability model. Atomic polarizability labels for all the selected atoms (see *sel_type*) should be provided by *polarizability.npy* in each data system. The file has number of frames lines and 9 times of number of selected atoms columns.
- *global_polar*: Fit a polarizability model. Polarizability labels should be provided by *polarizability.npy* in each data system. The file has number of frames lines and 9 columns.

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]`
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”.

precision:

type: `str`, optional, default: `float64`
argument path: `model/fitting_net[ener]/precision`

The precision of the fitting net parameters, supported options are “default”, “float16”, “float32”, “float64”.

resnet_dt:

type: `bool`, optional, default: `True`
argument path: `model/fitting_net[ener]/resnet_dt`

Whether to use a “Timestep” in the skip connection

trainable:

type: `bool` | `list`, optional, default: `True`
argument path: `model/fitting_net[ener]/trainable`

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

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

rcond:

type: `float`, optional, default: `0.001`
argument path: `model/fitting_net[ener]/rcond`

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

seed:

type: `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]`
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".

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: `float64`
argument path: `model/fitting_net[dipole]/precision`

The precision of the fitting net parameters, supported options are "default", "float16", "float32", "float64".

sel_type:

type: `int` | `NoneType` | `list`, optional
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]`
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".

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: `float64`
argument path: `model/fitting_net[polar]/precision`

The precision of the fitting net parameters, supported options are "default", "float16", "float32", "float64".

fit_diag:

type: `bool`, optional, default: `True`
argument path: `model/fitting_net[polar]/fit_diag`

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

scale:

type: `float` | `list`, optional, default: `1.0`
argument path: `model/fitting_net[polar]/scale`

The output of the fitting net (polarizability matrix) will be scaled by `scale`

diag_shift:

type: `float` | `list`, optional, default: `0.0`
argument path: `model/fitting_net[polar]/diag_shift`

The diagonal part of the polarizability matrix will be shifted by `diag_shift`. The shift operation is carried out after `scale`.

sel_type:

type: `int` | `NoneType` | `list`, optional
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

When type is set to `global_polar`:

neuron:

type: `list`, optional, default: `[120, 120, 120]`
argument path: `model/fitting_net[global_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[global_polar]/activation_function`

The activation function in the fitting net. Supported activation functions are "relu", "relu6", "softplus", "sigmoid", "tanh", "gelu".

resnet_dt:

type: `bool`, optional, default: `True`
argument path: `model/fitting_net[global_polar]/resnet_dt`

Whether to use a "Timestep" in the skip connection

precision:

type: `str`, optional, default: `float64`
argument path: `model/fitting_net[global_polar]/precision`

The precision of the fitting net parameters, supported options are "default", "float16", "float32", "float64".

fit_diag:

type: `bool`, optional, default: `True`
argument path: `model/fitting_net[global_polar]/fit_diag`

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

scale:

type: `float` | `list`, optional, default: `1.0`
argument path: `model/fitting_net[global_polar]/scale`

The output of the fitting net (polarizability matrix) will be scaled by `scale`

diag_shift:

type: `float` | `list`, optional, default: `0.0`
argument path: `model/fitting_net[global_polar]/diag_shift`

The diagonal part of the polarizability matrix will be shifted by `diag_shift`. The shift operation is carried out after `scale`.

sel_type:

type: `int` | `NoneType` | `list`, optional
argument path: `model/fitting_net[global_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[global_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 A^-1

ewald_h:

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

The grid spacing of the FFT grid. Unit is A

loss:

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

The definition of loss function. The loss type should be set to the fitting type 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`*, *`dipole`*, *`polar`*, *`global_polar`*

The type of the loss. The loss type should be set to the fitting type or left unset. .

When type is set to `ener`:

start_pref_e:

type: `float` | `int`, optional, default: `0.02`

argument path: `loss[ener]/start_pref_e`

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

limit_pref_e:

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

argument path: `loss[ener]/limit_pref_e`

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

start_pref_f:

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

argument path: `loss[ener]/start_pref_f`

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

limit_pref_f:

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

argument path: `loss[ener]/limit_pref_f`

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

start_pref_v:

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

argument path: `loss[ener]/start_pref_v`

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

limit_pref_v:

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

argument path: `loss[ener]/limit_pref_v`

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

start_pref_ae:

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

argument path: `loss[ener]/start_pref_ae`

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

limit_pref_ae:

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

argument path: `loss[ener]/limit_pref_ae`

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

relative_f:

type: `float` | `NoneType`, optional
argument path: `loss[ener]/relative_f`

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

When type is set to `dipole`:

pref_weight:

type: `float` | `NoneType` | `int`, optional, default: `None`
argument path: `loss[dipole]/pref_weight`

The prefactor of the weight of global loss. It should be larger than or equal to 0. If not provided, training will be atomic mode, i.e. atomic label should be provided.

pref_atomic_weight:

type: `float` | `NoneType` | `int`, optional, default: `None`
argument path: `loss[dipole]/pref_atomic_weight`

The prefactor of the weight of atomic loss. It should be larger than or equal to 0. If it's not provided and global weight is provided, training will be global mode, i.e. global label should be provided. If both global and atomic weight are not provided, training will be atomic mode, i.e. atomic label should be provided.

When type is set to `polar`:

pref_weight:

type: `float` | `NoneType` | `int`, optional, default: `None`
argument path: `loss[polar]/pref_weight`

The prefactor of the weight of global loss. It should be larger than or equal to 0. If not provided, training will be atomic mode, i.e. atomic label should be provided.

pref_atomic_weight:

type: `float` | `NoneType` | `int`, optional, default: `None`
argument path: `loss[polar]/pref_atomic_weight`

The prefactor of the weight of atomic loss. It should be larger than or equal to 0. If it's not provided and global weight is provided, training will be global mode, i.e. global label should be provided. If both global and atomic weight are not provided, training will be atomic mode, i.e. atomic label should be provided.

When type is set to `global_polar`:

pref_weight:

type: `float` | `NoneType` | `int`, optional, default: `None`
argument path: `loss[global_polar]/pref_weight`

The prefactor of the weight of global loss. It should be larger than or equal to 0. If it's not provided and atomic weight is provided, training will be atomic mode, i.e. atomic label should be provided. If both global and atomic weight are not provided, training will be global mode, i.e. global label should be provided.

pref_atomic_weight:

type: `float` | `NoneType` | `int`, optional, default: `None`
argument path: `loss[global_polar]/pref_atomic_weight`

The prefactor of the weight of atomic loss. It should be larger than or equal to 0. If not provided, training will be global mode, i.e. global label should be provided.

learning_rate:

type: `dict`
argument path: `learning_rate`

The definitio of learning rate

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.

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: `int` | `list` | `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 devided into blocks. A block is specified by *stt_idx:end_idx:weight*, where *stt_idx* is the starting index of the system, *end_idx* is then ending (not including) index of the system, the probabilities of the systems in this block sums up to *weight*, and the relatively probabilities within this block is proportional to the number of batches in the system.

sys_probs:

type: `NoneType` | `list`, optional, default: `None`, alias: *sys_weights*
argument path: `training/training_data/sys_probs`

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

validation_data:

type: `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: `int` | `list` | `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 devided into blocks. A block is specified by *stt_idx:end_idx:weight*, where *stt_idx* is the starting index of the system, *end_idx* is then ending (not including) index of the system, the probabilities of the systems in this block sums up to *weight*, and the relatively probabilities within this block is proportional to the number of batches in the system.

sys_probs:

type: `NoneType` | `list`, optional, default: `None`, alias: *sys_weights*
argument path: `training/validation_data/sys_probs`

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

numb_btch:

type: `int`, optional, default: `1`, alias: *numb_batch*
argument path: `training/validation_data/numb_btch`

An integer that specifies the number of systems to be sampled for each validation period.

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: `lcueve.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.

numb_test:

type: `int` | `list` | `str`, optional, default: `1`
argument path: `training/numb_test`

Number of frames used for the test during training.

save_freq:

type: `int`, optional, default: `1000`
argument path: `training/save_freq`

The frequency of saving check point.

save_ckpt:

type: `str`, optional, default: `model.ckpt`
argument path: `training/save_ckpt`

The file name of saving check point.

disp_training:

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

Displaying verbose information during training.

time_training:

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

Timing durining training.

profiling:

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

Profiling during training.

profiling_file:

type: `str`, optional, default: `timeline.json`
argument path: `training/profiling_file`

Output file for profiling.

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

CHAPTER

SEVEN

DEVELOPER GUIDE

CHAPTER EIGHT

LICENSE

The project DeePMD-kit is licensed under GNU LGPLv3.0.

CHAPTER

NINE

AUTHORS AND CREDITS

9.1 Package Contributors

- amacadmus
- AnguseZhang
- denghuilu
- bwang-ecnu
- frankhan91
- GeiduanLiu
- gzq942560379
- haidi-ustc
- hlyang1992
- hsulab
- iProzd
- JiabinYang
- marian-code
- njzjz
- tuoping
- wsyxbcl
- y1xiaoc
- YWolfeee
- zhouwei25
- ZiyaoLi

9.2 Other Credits

- Zhang ZiXuan for designing the Deepmodeling logo.
- Everyone on the *Deepmodeling mailing list* for contributing to many discussions and decisions!

(If you have contributed to the `deepmd-kit` core package and your name is missing, please send an email to the contributors, or open a pull request in the deepmd-kit repository)

- genindex
- modindex
- search