DeePMD-kit Manual

Contents

1 About DeePMD-kit

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.

1.1 Highlighted features

- **interfaced with TensorFlow**, one of the most popular deep learning frameworks, making the training process highly automatic and efficient.
- **interfaced with high-performance classical MD and quantum (path-integral) MD packages**, i.e., LAMMPS and i-PI, respectively.
- **implements the Deep Potential series models**, which have been successfully applied to finite and extended systems including organic molecules, metals, semiconductors, and insulators, etc.
- **implements MPI and GPU supports**, makes it highly efficient for high performance parallel and distributed computing.
- **highly modularized**, easy to adapt to different descriptors for deep learning based potential energy models.

1.2 Code structure

The code is organized as follows:

- `data/raw`: tools manipulating the raw data files.
- `examples`: example json parameter files.
- `source/3rdparty`: third-party packages used by DeePMD-kit.
- `source/cmake`: cmake scripts for building.
- `source/ipi`: source code of i-PI client.
- `source/lib`: source code of DeePMD-kit library.
- `source/lmp`: source code of Lammps module.
- `source/md`: source code of native MD.
- `source/op`: tensorflow op implementation. working with library.
- `source/scripts`: Python script for model freezing.
- `source/train`: Python modules and scripts for training and testing.

1.3 License and credits

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.`

1.4 Deep Potential in a nutshell

The goal of Deep Potential is to employ deep learning techniques and realize an inter-atomic potential energy model that is general, accurate, computationally efficient and scalable. The key component is to respect the extensive and symmetry-invariant properties of a potential energy model by assigning a local reference frame and a local environment to each atom. Each environment contains a finite number of atoms, whose local coordinates are arranged in a symmetry preserving way. These local coordinates are then transformed, through a sub-network, to a so-called *atomic energy*. Summing up all the atomic energies gives the potential energy of the system.

The initial proof of concept is in the Deep Potential paper, which employed an approach that was devised to train the neural network model with the potential energy only. With typical *ab initio* molecular dynamics (AIMD) datasets this is insufficient to reproduce the trajectories. The Deep Potential Molecular Dynamics (DeePMD) model overcomes this limitation. In addition, the learning process in DeePMD improves significantly over the Deep Potential method thanks to the introduction of a flexible family of loss functions. The NN potential constructed in this way reproduces accurately the AIMD trajectories, both classical and quantum (path integral), in extended and finite systems, at a cost that scales linearly with system size and is always several orders of magnitude lower than that of equivalent AIMD simulations.

Although being highly efficient, the original Deep Potential model satisfies the extensive and symmetry-invariant properties of a potential energy model at the price of introducing discontinuities in the model. This has negligible influence on a trajectory from canonical sampling but might not be sufficient for calculations of dynamical and mechanical properties. These points motivated us to develop the Deep Potential-Smooth Edition (DeepPot-SE) model, which replaces the non-smooth local frame with a smooth and adaptive embedding network. DeepPot-SE shows great ability in modelling many kinds of systems that are of interests in the fields of physics, chemistry, biology, and materials science.

In addition to building up potential energy models, DeePMD-kit can also be used to build up coarse-grained models. In these models, the quantity that we want to parametrize is the free energy, or the coarse-grained potential, of the coarse-grained particles. See the DeePCG paper for more details.

2 Download and install

Please follow our github webpage to see the latest released version and development version. ## Easy installation methods A docker for installing the DeePMD-kit on CentOS 7 is available here. We are currently working on installation methods using the `conda` package management system and `pip` tools. Hope these will come out soon.

2.1 Install DeePMD-kit from scratch

Installing DeePMD-kit from scratch is lengthy, but do not be panic. Just follow step by step. Wish you good luck..

2.1.1 Install tensorflow's Python interface

There are two ways of installing the Python interface of tensorflow, either using google's binary, or installing from sources. When you are using google's binary, do not forget to add the option `-DTF_GOOGLE_BIN=true` when building DeePMD-kit.

2.1.2 Install tensorflow's C++ interface

Firstly get the source code of the tensorflow

```
cd /some/workspace
git clone https://github.com/tensorflow/tensorflow tensorflow
```

The DeePMD-kit works with tensorflow r1.4 and later versions. Now taking r1.8 for example:

```
cd tensorflow
git checkout r1.8
```

Please make sure you have the Bazel higher than version 0.5.4, otherwise, please install it.

DeePMD-kit is compiled by cmake, so we need to compile and integrate tensorflow with cmake projects. The rest of this section basically follows the instruction provided by Tuatini. Now execute

```
./configure
```

You will answer a list of questions that help configure the building of tensorflow. It is recommended to build for Python3. You may want to answer the question like this:

```
Please specify the location of python. [Default is /usr/bin/python]: /usr/bin/python3
```

The library path for Python should be set accordingly.

Now build the shared library of tensorflow:

```
bazel build -c opt --verbose_failures //tensorflow:libtensorflow_cc.so
```

You may want to add options `--copt=-msse4.2`, `--copt=-mavx`, `--copt=-mavx2` and `--copt=-mfma` to enable SSE4.2, AVX, AVX2 and FMA SIMD accelerations, respectively. It is noted that these options should be chosen according to the CPU architecture. If the RAM becomes an issue of your machine, you may limit the RAM usage by using `--local_resources 2048,.5,1.0`.

Now I assume you want to install tensorflow in directory `$tensorflow_root`. Create the directory if it does not exists

```
mkdir -p $tensorflow_root
```

Before moving on, we need to compile the dependencies of tensorflow, including Protobuf, Eigen and nsync. Firstly, protobuf

```
mkdir /tmp/proto
tensorflow/contrib/makefile/download_dependencies.sh
cd tensorflow/contrib/makefile/downloads/protobuf/
./autogen.sh
./configure --prefix=/tmp/proto/
make
make install
```

Then Eigen

```
mkdir /tmp/eigen
cd ../eigen
mkdir build_dir
cd build_dir
cmake -DCMAKE_INSTALL_PREFIX=/tmp/eigen/ ../
make install
```

And nsync

```
mkdir /tmp/nsync
cd ../../nsync
mkdir build_dir
cd build_dir
cmake -DCMAKE_INSTALL_PREFIX=/tmp/nsync/ ../
make
make install
cd ../../../../../..
```

Now, copy the libraries to the tensorflow's installation directory:

```
mkdir $tensorflow_root/lib
cp bazel-bin/tensorflow/libtensorflow_cc.so $tensorflow_root/lib/
cp bazel-bin/tensorflow/libtensorflow_framework.so $tensorflow_root/lib/
cp /tmp/proto/lib/libprotobuf.a $tensorflow_root/lib/
cp /tmp/nsync/lib/libnsync.a $tensorflow_root/lib/
```

Then copy the headers

```
mkdir -p $tensorflow_root/include/tensorflow
cp -r bazel-genfiles/* $tensorflow_root/include/
cp -r tensorflow/cc $tensorflow_root/include/tensorflow
cp -r tensorflow/core $tensorflow_root/include/tensorflow
cp -r third_party $tensorflow_root/include
cp -r /tmp/proto/include/* $tensorflow_root/include
cp -r /tmp/eigen/include/eigen3/* $tensorflow_root/include
cp -r /tmp/nsync/include/*h $tensorflow_root/include
```

Now clean up the source files in the header directories:

```
cd $tensorflow_root/include
find . -name "*.cc" -type f -delete
```

The temporary installation directories for the dependencies can be removed:

```
rm -fr /tmp/proto /tmp/eigen /tmp/nsync
```

2.1.3 Install xdrfile

xdrfile is a lib that reads, compresses and writes the MD trajectories. Firstly get the source:

```
cd /some/workspace
wget ftp://ftp.gromacs.org/pub/contrib/xdrfile-1.1.4.tar.gz
```

I assume you want to install it in `$xdrfile_root`, then you will probably do

```
tar xvf xdrfile-1.1.4.tar.gz
cd xdrfile-1.1.4
./configure --prefix=$xdrfile_root
make
make install
```

2.1.4 Install DeePMD-kit

The DeePMD-kit was tested with compiler gcc >= 4.9.

Firstly clone the DeePMD-kit source code

```
cd /some/workspace
git clone https://github.com/deepmodeling/deepmd-kit.git deepmd-kit
```

If one downloads the .zip file from the github, then the default folder of source code would be `deepmd-kit-master` rather than `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`
```

Then goto the source code directory and make a build directory.

```
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 -DXDRFILE_ROOT=$xdrfile_root -DTENSORFLOW_ROOT=$tensorflow_root \
-DCMAKE_INSTALL_PREFIX=$deepmd_root ..
```

If you are using google binary for tensorflow python interface, then you need to specify

```
cmake -DXDRFILE_ROOT=$xdrfile_root -DTENSORFLOW_ROOT=$tensorflow_root \
-DCMAKE_INSTALL_PREFIX=$deepmd_root -DTF_GOOGLE_BIN=true ..
```

If the cmake has executed successfully, then

```
make
make install
```

If everything works fine, you will have the following executables installed in `$deepmd_root/bin`

```
$ ls $deepmd_root/bin
dp_frz  dp_ipi  dp_mdnn  dp_test  dp_train
```

2.1.5 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 your favorite LAMMPS code, and uncompress it (I assume that you have downloaded the tar `lammps-stable.tar.gz`)

```
cd /some/workspace
tar xf lammps-stable.tar.gz
```

The source code of LAMMPS is stored in directory, for example `lammps-31Mar17`. Now go into the LAMMPS code and copy the DeePMD-kit module like this

```
cd lammps-31Mar17/src/
cp -r $deepmd_source_dir/source/build/USER-DEEPMD .
```

Now build LAMMPS

```
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`.

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

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

2.1.6 Build DeePMD-kit with GPU support

If your system has a NVIDIA GPU, you can build TensorFlow with GPU support, which will be inherited by DeePMD-kit and LAMMPS. To achieve this, please carefully check the webpage Install TensorFlow from Source and look for the GPU version. In particular, you have to make sure that the required NVIDIA softwares, namely CUDA Toolkit, GPU drivers, and cuDNN SDK, must be installed on your system.

To install TensorFlow with GPU support, all the installation steps will be the same as the non-GPU version, except that one may allow the GPU option when doing `configure`, e.g.,

```
Do you wish to build TensorFlow with CUDA support? [y/N] Y
CUDA support will be enabled for TensorFlow
Do you want to use clang as CUDA compiler? [y/N]
nvcc will be used as CUDA compiler
Please specify the CUDA SDK version you want to use. [Leave empty to default to CUDA 9.0]: 9.0
Please specify the location where CUDA 9.0 toolkit is installed. Refer to README.md for more det
Please specify which gcc should be used by nvcc as the host compiler. [Default is /usr/bin/gcc]:
```

```
Please specify the cuDNN version you want to use. [Leave empty to default to cuDNN 7.0]: 7
Please specify the location where cuDNN 7 library is installed. Refer to README.md for more deta
Please specify a list of comma-separated CUDA compute capabilities you want to build with.
You can find the compute capability of your device at: https://developer.nvidia.com/cuda-gpus.
Please note that each additional compute capability significantly increases your build time and
```

After sucessfully installing TensorFlow with GPU support, you should install DeePMD, LAMMPS, etc., in the same way of the non-GPU version. Sometimes you may need to explicitly tell the compiler the place of the CUDA Toolkit and cuDNN libraries, i.e.,

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/cuda_toolkit/lib64
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/cudnn/lib64
```

3 Use DeePMD-kit

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
3. Freeze the model
4. MD runs with the model (Native MD code or LAMMPS)

3.1 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
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. 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.

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. The last set (`set.002`) is used as testing set, while the rest sets (`set.000` and `set.001`) are used as training sets. 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*.

3.2 Train a model

3.2.1 The DeePMD model

The method of training is explained in our DeePMD paper. With the source code we provide a small training dataset taken from 400 frames generated by NVT ab-initio water MD trajectory with 300 frames for training and 100 for testing. An example training parameter file is provided. One can try with the training by

```
$ cd $deepmd_source_dir/examples/train/
$ $deepmd_root/bin/dp_train water.json
```

`$deepmd_root/bin/dp_train` is the training program, and `water.json` is the `json` format parameter file that controls the training. The components of the `water.json` are

```
{
    "_comment": " model parameters",
    "use_smooth":   false,
    "sel_a":        [16, 32],
    "sel_r":        [30, 60],
    "rcut":     6.00,
```

```
    "axis_rule":    [0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0],
    "_comment": " default rule: []",
    "_comment": " user defined rule: for each type provides two axes, ",
    "_comment": "                     for each axis: (a_or_r, type, idx)",
    "_comment": "                     if type < 0, exclude type -(type+1)",
    "_comment": "                     for water (O:0, H:1) it can be",
    "_comment": "                     [0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0]",
    "n_neuron":     [240, 120, 60, 30, 10],

    "_comment": " training controls",
    "systems":      ["../data/water/"],
    "set_prefix":   "set",
    "stop_batch":   1000000,
    "batch_size":   4,
    "start_lr":     0.001,
    "decay_steps":  5000,
    "decay_rate":   0.95,

    "start_pref_e": 0.02,
    "limit_pref_e": 8,
    "start_pref_f": 1000,
    "limit_pref_f": 1,
    "start_pref_v": 0,
    "limit_pref_v": 0,

    "seed":     1,

    "_comment": " display and restart",
    "_comment": " frequencies counted in batch",
    "disp_file":    "lcurve.out",
    "disp_freq":    100,
    "numb_test":    100,
    "save_freq":    100,
    "save_ckpt":    "model.ckpt",
    "load_ckpt":    "model.ckpt",
    "disp_training":    true,
    "time_training":    true,

    "_comment":     "that's all"
}
```

The option `rcut` is the cut-off radius for neighbor searching. The `sel_a` and `sel_r` are the maximum selected numbers of fully-local-coordinate and radial-only-coordinate atoms from the neighbor list, respectively. `sel_a + sel_r` should be larger than the maximum possible number of neighbors in the cut-off radius. `sel_a` and `sel_r` are vectors, the length of the vectors are same as the number of atom types in the system. `sel_a[i]` and `sel_r[i]` denote the selected number of neighbors of type `i`.

The option `axis_rule` specifies how to make the axis for the local coordinate of each atom. For

each atom type, 6 integers should be provided. The first three for the first axis, while the last three for the second axis. Within the three integers, the first one specifies if the axis atom is fully-local-coordinated (`0`) or radial-only-coordinated (`1`). The second integer specifies the type of the axis atom. If this number is less than 0, saying `t < 0`, then this axis exclude atom of type `-(t+1)`. If the third integer is, saying `s`, then the axis atom is the `s`th nearest neighbor satisfying the previous two conditions.

The option `n_neuron` is an integer vector that determines the shape the neural network. The size of the vector is identical to the number of hidden layers of the network. From left to right the members denote the sizes of each hidden layers from input end to the output end, respectively.

The option `systems` provide location of the systems (path to `set.*` and `type.raw`). It is a vector, thus DeePMD-kit allows you to provide multiple systems. DeePMD-kit will train the model with the systems in the vector one by one in a cyclic manner.

The option `batch_size` specifies the number of frames in each batch. The option `stop_batch` specifies the total number of batches will be used in the training. The option `start_lr`, `decay_rate` and `decay_steps` specify how the learning rate changes. For example, the `t`th batch will be trained with learning rate:

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

The options `start_pref_e`, `limit_pref_e`, `start_pref_f`, `limit_pref_f`, `start_pref_v` and `limit_pref_v` determine how the prefactors of energy error, force error and virial error changes in the loss function (see the appendix of the DeePMD paper for details). Taking the prefactor of force error for example, the prefactor at batch `t` is

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

Since we do not have virial data, the virial prefactors `start_pref_v` and `limit_pref_v` are set to 0.

The option `seed` specifies the random seed for neural network initialization. If not provided, the `seed` will be initialized with `None`.

During the training, the error of the model is tested every `disp_freq` batches with `numb_test` frames from the last set in the `systems` directory on the fly, and the results are output to `disp_file`.

Checkpoints will be written to files with prefix `save_ckpt` every `save_freq` batches. If `restart` is set to `true`, then the training will start from the checkpoint named `load_ckpt`, rather than from scratch.

Several command line options can be passed to `dp_train`, which can be checked with

```
$ $deepmd_root/bin/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
  -t INTER_THREADS, --inter-threads INTER_THREADS
                        With default value 0. Setting the "inter_op_parallelism_threads" key for
  --init-model INIT_MODEL
                        Initialize a model by the provided checkpoint
  --restart RESTART     Restart the training from the provided checkpoint
```

The keys `intra_op_parallelism_threads` and `inter_op_parallelism_threads` are Tensorflow configurations for multithreading, which are explained here. Skipping `-t` and `OMP_NUM_THREADS` leads to the default setting of these keys in the Tensorflow.

`--init-model model.ckpt`, for example, 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`.

3.2.2 The DeepPot-SE model

The smooth version of DeePMD, or the DeepPot-SE model, can also be trained by DeePMD-kit. An example training parameter file is provided. One can try with the training by

```
$ cd $deepmd_source_dir/examples/train/
$ $deepmd_root/bin/dp_train water_smth.json
```

The difference between the standard and smooth DeePMD models lies in the model parameters:

```
"use_smooth":   true,
"sel_a":        [46, 92],
"rcut_smth":    5.80,
"rcut":     6.00,
"filter_neuron":    [25, 50, 100],
"filter_resnet_dt": false,
"n_axis_neuron":    16,
"n_neuron":     [240, 240, 240],
"resnet_dt":    true,
```

The `sel_r` option is skipped by the smooth version and the model use fully-local-coordinate for all neighboring atoms. The `sel_a` should larger than the maximum possible number of neighbors in the cut-off radius `rcut`.

The descriptors will decay smoothly from `rcut_smth` to the cutoff radius `rcut`.

`filter_neuron` provides the size of the filter network (also called local-embedding network). If the size of the next layer is the same or twice as the previous layer, then a skip connection is build (ResNet). `filter_resnet_dt` tells if a timestep is used in the skip connection. By default it is `false`. `n_axis_neuron` specifies the number of axis filter, which should be much smaller than the size of the last layer of the filter network.

`n_neuron` specifies the fitting network. If the size of the next layer is the same as the previous layer, then a skip connection is build (ResNet). `resnet_dt` tells if a timestep is used in the skip connection. By default it is `true`.

3.3 Freeze and test a model

The trained neural network is extracted from a checkpoint and dumped into a database. This process is called "freeze" a model. Typically one does

```
$ $deepmd_root/bin/dp_frz -o graph.pb
```

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

The frozen model can be used in many ways. The most straightforward test can be performed using `dp_test`. Several command line options can be passed to `dp_test`, which can be checked with

```
$ $deepmd_root/bin/dp_test --help
```

An explanation will be provided

```
usage: dp_test [-h] [-m MODEL] [-s SYSTEM] [-S SET_PREFIX] [-n NUMB_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
  -d DETAIL_FILE, --detail-file DETAIL_FILE
                        The file containing details of energy force and virial
                        accuracy
```

The files `dp_frz` and `dp_test` may also serve as a python template for further analyses and more user-specific applications.

3.4 Run MD with LAMMPS

3.4.1 Include deepmd in the pair style

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. The `pair_coeff` should be left blank. 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.

3.4.2 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.

3.5 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`.

3.6 Run MD with native code

DeePMD-kit provides a simple MD implementation that runs under either NVE or NVT ensemble. One needs to provide the following input files

```
$ ls
conf.gro  graph.pb  water.json
```

`conf.gro` is the file that provides the initial coordinates and/or velocities of all atoms in the system. It is of Gromacs `gro` format. Details of this format can be find in this website. It should be notice that the length unit of the `gro` format is **nm** rather than A.

`graph.pb` is the frozen model.

`water.json` is the parameter file that specifies how the MD runs. An example parameter file for water NVT simulation is provided.

```
{
    "conf_file":    "conf.gro",
    "conf_format":  "gro",
    "graph_file":   "graph.pb",
    "nsteps":       500000,
    "dt":       5e-4,
    "ener_freq":    20,
    "ener_file":    "energy.out",
    "xtc_freq":     20,
    "xtc_file":     "traj.xtc",
    "trr_freq":     20,
    "trr_file":     "traj.trr",
    "print_force":  false,
    "T":        300,
    "tau_T":        0.1,
    "rand_seed":    2017,
    "atom_type" : {
    "OW":       0,
    "HW1":      1,
    "HW2":      1
    },
    "atom_mass" : {
    "OW":       16,
    "HW1":      1,
    "HW2":      1
    }
}
```

The options `conf_file`, `conf_format` and `graph_file` are self-explanatory. It should be noticed, again, the length unit is nm in the `gro` format file.

The option `nsteps` specifies the number of time steps of the MD simulation. The option `dt` specifies the timestep of the simulation.

The options `ener_file` and `ener_freq` specify the energy output file and frequency.

The options `xtc_file`, `xtc_freq`, `trr_file` and `trr_freq` are similar options that specify the output files and frequencies of the xtc and trr trajectory, respectively. When the frequencies are set to 0, the corresponding file will not be output. The instructions of the xtc and trr formats can be found in xtc manual and trr manual. It is noticed that the length unit in the xtc and trr files is **nm**.

If the option `print_force` is set to `true`, then the atomic force will be output.

The option `T` specifies the temperature of the simulation, and the option `tau_T` specifies the timescale of the thermostat. We implement the Langevin thermostat for the NVT simulation. `rand_seed` set the random seed of the random generator in the thermostat.

The `atom_type` set the type for the atoms in the system. The names of the atoms are those provided in the `conf_file` file. The `atom_mass` set the mass for the atoms. Again, the name of the atoms are those provided in the `conf_file`.

4 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.

4.1 Installation: inadequate versions of gcc/g++

Sometimes you may use a gcc/g++ of version <4.9. If you have a gcc/g++ of version > 4.9, 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++
```

If, for any reason, for example, you only have a gcc/g++ of version 4.8.5, you can still compile all the parts of TensorFlow and most of the parts of DeePMD-kit. In this case, follow the following steps.

First, goto the source code directory, open the file `CMakeLists.txt`

```
cd $deepmd_source_dir/source
vi CMakeLists.txt
```

Next, comment the following 4 lines out:

```
# set (LIB_DEEPMD_NATIVE  "deepmd_native_md")
# set (LIB_DEEPMD_IPI     "deepmd_ipi")
# add_subdirectory (md/)
# add_subdirectory (ipi/)
```

Then you may continue with the installation procedure.

4.2 Installation: 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.

4.3 Training: TensorFlow abi binary cannot be found when doing training

If you confront such kind of error:

```
$deepmd_root/lib/deepmd/libop_abi.so: undefined symbol:
_ZN10tensorflow8internal21CheckOpMessageBuilder9NewStringB5cxx11Ev
```

you may set `-DTF_GOOGLE_BIN=true` in the process of `cmake`.

Another possible reason might be the large gap between the python version of TensorFlow and the TensorFlow c++ interface.

4.4 MD: cannot run LAMMPS after installing a new version of DeePMD-kit

This typically happens when you install a new version of DeePMD-kit and copy directly the generated `USER-DEEPMD` to a LAMMPS source code folder and re-install LAMMPS.

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

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

and then install the new `USER-DEEPMD`.

If this does not solve your problem, try to decompress the LAMMPS source tarball and install LAMMPS from scratch again, which typically should be very fast.