Hands-on tutorial for DeePMD-kit

This tutorial will introduce you to the basic usage of DeePMD-kit, including data preparation, training, testing and running molecular dynamics simulations with lammps.

Typically the DeePMD-kit workflow contains three parts: data preparation, training/freezing/testing and molecular dynamics.

The folder structure of this tutorial is like this:

```
$ ls
00.data  01.train  02.lmp  readme.md
```

where the folder `00.data` contains the data, the folder `01.train` contains an example input script to train a model with DeePMD-kit and the folder `02.lmp` contains lammps example script for molecular dynamics simulation.

Before starting with the tutorial, we want you to notice that the unit system in the DeePMD-kit is:

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

Data preparation

The training data of DeePMD-kit contains the atom type, the simulation box, the atom coordinate, the atom force, the system energy and the virial. A snapshot of a molecular system that has these information is called a **frame**. A **system of data** or in short a **system** includes a number of frames that share the same number of atoms and atom types. For example, a molecular dynamics trajectory can be converted in to a system of data, with each timestep corresponding to a frame in the system.

The DeePMD-kit adopts a compressed data format. All training data should be firstly converted into this format and then can be used by DeePMD-kit. The data format is explained in detail in the DeePMD-kit manual that can be found in the DeePMD-kit official github site http://www.github.com/deepmodeling/deepmd-kit.

We provide a convenient tool named `dpdata` for converting the data produced by VASP, Gaussian, Quantum-Espresso and Lammps into the compressed format of DeePMD-kit. As an example, go to the data folder:

```
$ cd data
$ ls
OUTCAR
```

The OUTCAR that was produced by an ab-initio molecular dynamics (AIMD) simulation of a gas phase methane molecule using VASP. Now start an interactive python environment, for example

```
$ python
```

then execute the following commands:

```
import dpdata
import numpy as np
sys = dpdata.LabeledSystem('OUTCAR', fmt = 'vasp/outcar')
print('# the system contains %d frames' % sys.get_nframes())
sys.to_deepmd_npy('system_0', set_size = 20, prec = np.float32)
```

The commands import a system of data from the OUTCAR (with format `vasp/outcar`), and then dump it in to the compressed format (numpy compressed arrays). The output folder `system_0`.

```
$ ls system_0
set.000  set.001  set.002  set.003  set.004  set.005  set.006  set.007  set.008  set.009
```

The data system that has 200 frames is split into 10 sets, each of which has 20 frames. The DeePMD-kit assumes that the last set, i.e. `set.009`, is the testing set, while the other sets are the training sets. The parameter `set_size` specifies the set size. The parameter `prec` specifies the precision of the floating point numbers.

The file `type.raw` gives the type of atoms in the system. Since all frames in the system has the same atom types and atom numbers, we only need to specify the type information once for the whole system

```
$ cat system_0/type.raw
0 0 0 0 1
```

where the atom `H` is given type `0`, and atom `C` is given type `1`.

Training

Prepare input script

Once the data preparation is done, we can go on with training. Now go to the training directory

```
$ cd ../01.train
$ ls
input.json
```

where input.json gives you an example training script. The options are explained in detail in the DeePMD-kit manual, so they are not comprehensively explained. We want to draw your attention to a few of important parameters that matters the training accuracy. In the following are the parameters that specify the neural network architecture.

```
{
    "rcut_smth":        5.80,
    "rcut":             6.00,
    "sel_a":            [4, 1],
    "filter_neuron":    [10, 20, 40],
    "axis_neuron":      4,
    "fitting_neuron":   [100, 100, 100],
}
```

- `rcut` gives the cut-off radius, while the key `rcut_smth` gives where the inverse distance start to be smoothed. That is to say the distance dependency 1/r decays to 0 smoothly from `rcut_smth` to `rcut`.
- `sel_a` gives the (estimated) maximum number of atoms in the cut-off radius. This variable is a list, and `sel_a[i]` gives the maximum number of type i atoms in the cut-off radius. In the case of gas phase methane molecule `CH4`, a natural choice of `sel_a` is `[4,1]`, given `H` is of type 0 and `C` is of type 1.
- `filter_neuron` gives the size of embedding neural network.
- `fitting_neuron` gives the size of fitting neural neural.
- `axis_neuron` gives the number of axis neurons, i.e. the size of matrix G_1. This number should be smaller than or equal to `filter_neuron[-1]`.

The training parameters are given in the following

```
{
    "use_smooth":   true,
    "systems":          ["../00.data/system_0/"],
    "set_prefix":       "set",
    "batch_size":       [8],
    "stop_batch":       1000000,
    "start_lr":         0.001,
    "decay_steps":      5000,
    "decay_rate":       0.95,
}
```

- `use_smooth` tells the DeePMD-kit that the smoothed deep potential, namely DeepPot-SE model, will be trained.
- `systems` is a list that provides location of the systems (in our case we have only one system). DeePMD-kit allows you to provide multiple systems. DeePMD-kit will train the model with the systems

randomly picked from the list.
- `batch_size` is a list that provides the batch size, i.e. number of frames used in one SGD step, of each system, respectively
- `stop_batch` specifies the total number of SGD step (equal to the number of batches) used in the training.
- `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 )
```

It is noted that the parameter `stop_batch` and `decay_steps` are usually increased in proportion, so that the learning rate at the end of the training does not change. A rule-of-thumb guide of the learning rate at the end of training is of order `10^{-8}` to `10^{-7}`.

Train a model

After the training script is prepared, we can start the training with DeePMD-kit by simply running

```
$ dp_train input.json
```

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

```
# DEEPMD: ---Summary of DataSystem-----------------------------------------
# DEEPMD: find 1 system(s):
# DEEPMD:                                     system  natoms  bch_sz  n_bch
# DEEPMD:                         ../00.data/system_0/       5       8     18
# DEEPMD: -----------------------------------------------------------------
```

and the starting and final learning rate of this training

```
# DEEPMD: start training at lr 1.00e-03 (== 1.00e-03), final lr will be 3.51e-08
```

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

```
# DEEPMD: batch    1000 training time 5.92 s, testing time 0.01 s
# DEEPMD: batch    2000 training time 5.20 s, testing time 0.01 s
# DEEPMD: batch    3000 training time 5.50 s, testing time 0.01 s
# DEEPMD: batch    4000 training time 5.54 s, testing time 0.01 s
# DEEPMD: batch    5000 training time 5.58 s, testing time 0.01 s
# DEEPMD: batch    6000 training time 5.59 s, testing time 0.01 s
# DEEPMD: batch    7000 training time 5.64 s, testing time 0.01 s
# DEEPMD: batch    8000 training time 5.64 s, testing time 0.01 s
# DEEPMD: batch    9000 training time 5.73 s, testing time 0.01 s
# DEEPMD: batch   10000 training time 5.71 s, testing time 0.01 s
# DEEPMD: saved checkpoint model.ckpt
```

They present the training and testing time counts. At the end of the 10000th batch, the model is saved in Tensorflow's checkpoint file `model.ckpt`. At the same time, the training and testing errors are presented in file `lcurve.out`.

```
$ head -n 2 lcurve.out
# batch      l2_tst    l2_trn    l2_e_tst  l2_e_trn    l2_f_tst  l2_f_trn         lr
      0    4.91e+01  5.55e+01    4.61e+00  4.59e+00    1.55e+00  1.75e+00    1.0e-03
```

and

```
$ tail -n 2 lcurve.out
 137000    1.87e+00  1.70e+00    9.64e-04  1.24e-03    1.18e-01  1.07e-01    2.5e-04
 138000    1.90e+00  1.72e+00    3.23e-03  2.86e-03    1.20e-01  1.09e-01    2.5e-04
 139000    1.93e+00  1.61e+00    1.58e-03  1.33e-03    1.21e-01  1.01e-01    2.5e-04
 140000    1.86e+00  1.68e+00    1.50e-03  1.66e-03    1.21e-01  1.09e-01    2.4e-04
 141000    1.83e+00  1.67e+00    9.39e-04  6.38e-04    1.19e-01  1.08e-01    2.4e-04
```

Volumes 4, 5 and 6, 7 presents the energy and force training and testing errors, respectively. It is demonstrated that after 140,000 steps of training, the energy testing error is around 1 meV and the force testing error is around 120 meV/Å. It is also observed that the force testing error is systematically (but slightly) larger than the training error, which implies a slight over-fitting to the rather small dataset.

Freeze a model

At the end of the training, the model parameters saved in tensorflow's checkpoint file should be frozen as a model file that is usually ended with extension `.pb`. Simply execute

```
$ dp_frz
Converted 46 variables to const ops.
922 ops in the final graph.
```

and it will output a model file named `frozen_model.pb` in the current directory.

Run molecular dynamics simulation with Lammps

Now let's switch to the lammps directory to check the necessary input files for running DeePMD with lammps.

```
$ cd ../02.lmp
```

Firstly, we soft-link the output model in the training directory to the current directory

```
$ ln -s ../01.train/frozen_model.pb .
```

Then we have three files

```
$ ls
conf.lmp  frozen_model.pb  in.lammps
```

where `conf.lmp` gives the initial configuration of a gas phase methane MD simulation, and the file `in.lammps` is the lammps input script. One may check `in.lammps` and finds that it is a rather standard lammps input file for a MD simulation, with only two exception lines:

```
pair_style      deepmd frozen_model.pb
pair_coeff
```

where the pair style `deepmd` is invokes and the model file `frozen_model.pb` is provided, which means the atomic interaction will be computed by the DeePMD model that is stored in the file `frozen_model.pb`.

One may execute lammps in the standard way

```
$ lmp_mpi -i in.lammps
```

After waiting for a while, the MD simulation finishes, and the `log.lammps` and `ch4.dump` files are generated. They store the thermodynamic information and the trajectory of the molecule, respectively. One may want to visualize the trajectory by, e.g. ovito

```
$ ovito ch4.dump
```

to check the evolution of the molecular configuration.