Neural Network Potential Tutorial

Lukáš Kývala, Alexander Gorfer

July 18th 2022

1 Introduction to n2p2

In this tutorial, you will learn how to train a neural network potential (NNP) and use it for energy and force predictions. We will also cover how to accelerate molecular dynamics with NNPs. For this purpose we will use n2p2 (Neural Network Potential Package), a package of different tools written in C++. The code is freely available under `https://github.com/CompPhysVienna/n2p2` and documented on `https://compphysvienna.github.io/n2p2/index.html`

Three input files are required to train a neural network potential:

- input.nn
  - contains the setup of neural networks and symmetry functions
  - `https://compphysvienna.github.io/n2p2/topics/keywords.html`
- scaling.data
  - contains symmetry function statistics for normalizing symmetry functions
  - can be generated via the nnp-scaling tool
- input.data
  - stores configurations in a simple ASCII format
  - this file is used both in training as well as in predictions
  - `https://compphysvienna.github.io/n2p2/topics/cfg_file.html`

To use a NNP for predicting energies (and forces), we also need to prepare element-specific files containing weights and biases. These files are called weights.***.data, where *** is replaced by the atomic number of the element. For example, we must provide the files weights.001.data and weights.008.data when predicting energies of water configurations.

The n2p2 package is controlled via a command-line interface. Always execute each line individually. If you use Docker, start the first container for running n2p2 by (for Windows):
docker run -v <fullpath>\Day06_July18:/home/Day06_July18 -p 8888:8888 -it summer-school /bin/bash

For Mac/Linux:

docker run -v <fullpath>/Day06_July18:/home/Day06_July18 -p 8888:8888 -it summer-school /bin/bash

We also need the second container for using jupyter notebooks. Open a new window/tab of PowerShell/Terminal and start a new container with a different port. For Windows:

docker run -v <fullpath>\Day06_July18:/home/Day06_July18 -p 8889:8889 -it summer-school /bin/bash

or for Mac/Linux

docker run -v <fullpath>/Day06_July18:/home/Day06_July18 -p 8889:8889 -it summer-school /bin/bash

Then open the jupyter notebook in the second Docker container by:

```
jupyter-notebook --ip 0.0.0.0 --no-browser --allow-root --port=8889
```

A lot will be printed to the terminal. At the end, there will be a URL like `http://127.0.0.1:8889/?token=07c5e38...` Copy the full address including the token into a browser and you should see a jupyter notebook page. Open the `Learning_curves.ipynb` notebook and you should be ready to go.

If you have installed n2p2 on your own, you can speed up most of the work with mpi.

2 Training a Water Neural Network Potential

The first part of the tutorial is dedicated to training an NNP of water. Change your directory using the `cd` command to /home/Day06_July18/H2O (in the first Docker container):

```
cd /home/Day06_July18/H2O
```

Two files (input.nn and input.data) are already prepared in this folder. You will use them repeatedly, so always modify only copies and keep the originals unchanged. This can be achieved by creating a new folder for each task and copying input.data and input.nn into the new folder:

```
mkdir folder_name
cp input.nn folder_name/
cp input.data folder_name/
```

2.1 Training procedure

You have learned in Section 1 that we need three files for training. Two of them have already been prepared for you, but do not worry, you will get in touch with them during the exercises. For now, keep these files unchanged and go through the training procedure:

1. Prepare the dataset (input.data has been prepared).

2. Prepare a settings file (input.nn file has been prepared).

3. Compute symmetry function statistics.

   - Among the best practices for training a neural network is to normalize the input layer (in our case symmetry functions) to obtain a mean close to 0. Normalizing the input layer generally speeds up learning and leads to a faster convergence. In order to normalize symmetry functions, we need statistical values like their maximum, minimum, mean, and sigma for each symmetry function.
   - Utilize the nnp-scaling tool to compute all symmetry functions for all atoms and store these statistics in a file (scaling.data). This tool requires the input.data and input.nn files.

     ```
     /home/n2p2/bin/nnp-scaling 100
     ```

   - If you are successful, a scaling.data file containing symmetry function statistics will have been created. These statistics will be used during training.
   - Open scaling.data and see how the statistical values of different symmetry functions differ from each other.

4. NNP training

   - Copy the newly created scaling.data to your newly created folder, which already contains input.data and input.nn:

     ```
     cp scaling.data folder_name/
     ```

     If all three files are present, you can train a model using the nnp-train tool:

     ```
     cd folder_name
     /home/n2p2/bin/nnp-train
     ```

   - After each epoch, the RMSEs (Root Mean Squared Error) for energies and forces are printed. You can also check the newly created learning-curve.out file, which contains various error metrics for all epochs. We are particularly interested in columns 2, 3, 10, and 11 (they are in atomic units). Plot the training error versus epochs using the jupyter notebook from the second docker container. Change the folder variable in jupyter notebook to H2O/folder_name. You can see the last epoch errors are quite different from the penultimate one. This is an indication that the number of epochs is insufficient and that you can get better accuracy with more epochs. Create a new folder and copy input.nn, input.data and scaling.data again.

```
cd ..
mkdir folder_name2
cp input.nn folder_name2/
cp input.data folder_name2/
cp scaling.data folder_name2/
cd folder_name2/
```

- In the new folder, open the input.nn file with your favorite text editor (vim, nano, NotePad, ...) and change the keyword called epochs from 3 to a higher number (10 is enough) and train the model again. If you have increased the number of epochs sufficiently, errors will not decrease significantly at later epochs.

5. Collect weight files

   - In the last step, select the best model. Identify the model $e that has the best score (the lowest error) for training energies and forces. Then copy the chosen epoch weights into the prediction format.

     ```
     cp weights.001.00000$e.out weights.001.data
     cp weights.008.00000$e.out weights.008.data
     ```

   - Congratulations. You have successfully trained a neural network potential.

2.2 Train/validation/test dataset

1. You may have noticed that your model, while having the lowest training error, does not simultaneously possess the lowest test error. Choosing a model based on training error is a bad habit since neural networks tend to overtrain. To avoid this, we will use a validation set and early stopping. First, we need to separate the test set from the training and validation set.

   Change your working directory to /home/Day06_July18/H2O. Then use the nnp-select tool to select around 10% of structures from the original dataset (input.data). Our dataset has 100 structures, so we want to select around 10 structures. We want a random selection from this dataset and we achieve this by using the argument "random". It makes the selective process stochastic so there is no guarantee that 10 structures will be selected initially. Change $random_seed to a positive integer until you get 10 structures selected (9 or 11 structures are also acceptable).

   ```
   /home/n2p2/bin/nnp-select random 0.1 $random_seed
   ```

   Selected structures are saved into output.data, the rest into reject.data. Rename output.data to test.data and reject.data to train-validation.data:

```
mv reject.data train-validation.data
mv output.data test.data
```

Create a new folder and copy input.nn and scaling.data inside. However, instead of copying input.data as usual, this time copy train-validation.data as input.data:

```
cp train-validation.data folder_name5/input.data
```

The new input.data should contain about 90% of the original dataset. Train the model as before. But this time, select the best model not only based on the training errors, but also on validation errors by taking into account E_test , F_test in the output or the plot in the notebook (n2p2 incorrectly calls the validation-set test-set). The neural network is not trained on the validation set, but we select the model based on the errors encountered in the validation set. You will probably choose a model that does not have the lowest error in the training set, precisely because we now consider the validation set as well. This is called early stopping and it prevents overtraining.

Create a new folder inside the current one and copy input.nn, scaling.data, weights.001.data, weights.008.data (as in Section 2.1.5) and test.data from /home/Day06_July18/H2O folder as input.data.

```
mkdir test
cp input.nn test/
cp scaling.data test/
cp weights.001.data test/
cp weights.008.data test/
cp ../test.data test/input.data
cd test
```

As everything is ready, it is time to find out the RMSE for the test dataset. Utilize the nnp-dataset tool for this task. We redirect its output to the test-error.txt file so that we can check test errors repeatedly:

```
/home/n2p2/bin/nnp-dataset 1 >test-error.txt
```

RMSEs for the test dataset are printed at the end of the test-error.txt file. If we are interested in predicting energy and forces for only one structure, it is more convenient to use the nnp-predict tool.

In this exercise we encountered a train, test and validation set. Only the train set is used for training. The validation and test sets contain structures that have not been used for training. We calculate the errors for train and validation set every epoch. The test set stands separately and its error is evaluated only once.

2. Although only one file with atomic configurations (input.data) has been provided, we got training RMSEs as well as validation RMSEs. The n2p2 package automatically splits the structures from input.data into a test and validation dataset. The ratio is controlled by the test_fraction keyword in input.nn. Create a new folder and copy into it all required files. Then increase the test_fraction to 0.9. It means that only 10% of the dataset is used for training and the remaining 90% of the dataset is used for validation. Train a model and evaluate testing errors. Do you expect an increase or decrease in validation and test errors?

3. Training a neural network is stochastic. We can achieve different results by simply changing the random seed, which is set by the random_seed keyword in input.nn. By changing the random seed we will get a different split of the train/validation dataset, different initialization of weights and biases and a few more advanced things. Therefore, we should never rely on just one model. If computational resources allow us to, we should always repeat the training with a different random seed. Create a new folder and change the random seed to any positive integer number. Then compare the errors with the model from the previous random seed.

2.3 Hyperparameter tuning

As in any machine learning model, it is necessary to perform hyperparameter tuning. Neural network potentials are no exception. While cross-validation is the most proper way, we will not use cross-validation due to time constraints.

The n2p2 package uses predefined descriptors. Their proper tuning, together with a well-constructed structure dataset, are the most important elements for a good NNP. Generating descriptors is covered in Section 4 but dataset construction is beyond the scope of this tutorial. Another important element for a good neural network model is an adequate network topology. For NNPs we use relatively small neural networks with 1-3 hidden layers and a node count of less than 100. Small size is important if we plan to apply NNPs for many different structures, such as those encountered in a molecular dynamics simulation (Section 3). The topology of a neural network potential will also depends on the material in question and it is always necessary to perform this type of tuning.

1. Using the same train and test set from the previous task, try to slightly increase (5-10) the number of nodes in both layers. The number of nodes is set using the global_nodes_short keyword in input.nn.

2. Repeat the process, but this time reduce the number of nodes in both layers. Compare the evolution of the RMSE of the test set as a function of the number of nodes. For this task, you can use the second pre-prepared plot in the jupyter notebook. Enter the

number of nodes in x and test errors in y arrays. Also, find out how the actual number of parameters (weights + biases) changed compared to the model with the two layered 20 nodes one by checking the weights files.

3. In the next exercise, keep the number of nodes the same but change the activation functions. Activation functions are set via the global_activation_short keyword in input.nn. A list of activation functions and corresponding keywords is available under `https://compphysvienna.github.io/n2p2/topics/keywords.html#global-activation-short`

4. This time try the model with just one hidden layer. You need to modify the keyword global_hidden_layers_short to 1 and write only one integer after the global_nodes_short and only two letters in the global_activation_short.

There are many more hyperparameters that can be optimized. However, their influence is often noticeably smaller and the default values are sufficient for most cases.

3 Molecular Dynamics with an NNP

Your n2p2 installation comes with a LAMMPS molecular-dynamics suite with an interface to n2p2 for energy and force predictions. It allows driving molecular dynamics simulations with a neural network potential. Head on over to:

```
cd /home/Day06_July18/MolecularDynamics/H2O
```

In this folder, you should see an `nnp-data` folder containing a neural network potential just like the one you trained (but on a larger training set), a LAMMPS input file called `md.lmp` and a starting configuration `initial_configuration.data`.

3.1 Running molecular dynamics with LAMMPS

To run a simulation, LAMMPS needs a series of commands in its input file. They typically come in four categories:

```
# Initialization
# System definition
# Simulation settings
# Run
```

In our `md.lmp`-file, the `Initialization`-part looks like:

```
units metal
boundary p p p
atom_style atomic
```

```
timestep 0.0005
thermo 1
```

The first line defines the units used (Angstrom, eV, picoseconds ...). In the second line, we set periodic boundary conditions in all directions. Since our NNP does not use charges, we tell LAMMPS to not expect them with `atom_style atomic`. Then in the fourth line, we set our timestep to 0.0005 ps = 0.5 fs. Finally, we set `thermo 1` to save energy, temperature, and pressure at each timestep.
In the `System definition` part, we read in a water configuration from a local file:

```
read_data "initial_configuration.data"
```

The `Simulation`-part contains:

```
mass 1 1.00794
mass 2 15.9994

pair_style nnp dir nnp-data showew no showewsum 10 resetew no maxew 100 cflength
    1.8897261328 cfenergy 0.0367493254 emap "1:H,2:O"
pair_coeff * * 6.36

# INTEGRATOR
fix INT all nve
```

First, we define the masses of our particles. Then an interaction potential is defined; in our case a neural network potential. We need to supply this keyword with a few settings. `dir nnp-data` sets the folder containing the trained potential. The four keyword combinations `showew no showewsum 10 resetew yes maxew 100` tell LAMMPS how to handle extrapolation warnings. The `scaling.data` contains minimum and maximum values of the descriptors used to train the neural network potential in the training set. The behavior outside the training set - where neural networks suddenly need to extrapolate instead of interpolate - can quickly become unphysical and we could get bad predictions. A rough estimate of this state of extrapolation is when our descriptors are above/below their known max/min values encountered in the training set. The developer of the NNP needs to decide how to handle these cases. We have chosen to not show an immediate extrapolation warning once they appear but only show statistics on them every 10 steps. We do not sum up extrapolation warnings but reset at every step and if in a single step we notice 100 extrapolation warnings (something very bad is happening), we stop the simulation. Since the NNP was trained on atomic units we need to convert these to the LAMMPS metal units: `cflength 1.8897261328 cfenergy 0.0367493254`. In the next line, we set the cutoff of the potential to the one we trained on plus a little bit of leeway; `6.36` Å. And in the last line, an NVE integration scheme (velocity-Verlet without thermo- or barostats) is defined.
The simulation is finally launched using:

```
dump 1 all atom 5 traj.dump
run 10000
```

where we first tell LAMMPS to save a configuration containing all coordinates of each atom every 5 timesteps and then we run for 10000 timesteps in total.
To run this simulation use:

```
/home/n2p2/bin/lmp_mpi -in md.lmp
```

Machine learning force fields can run much faster than ab-initio methods but are in general still more expensive than classical force-fields. We can continue in a jupyter notebook while the simulation runs a bit in the background:

3.2 Visualization, Energy, Pair Correlation and Diffusion behaviour

While performing a molecular dynamics simulation it is important to do a few small sanity checks where inconsistencies can be caught. We want to visualize the trajectory by rendering its configurations, look at temperature and the conservation of total energy, determine the pair correlation function and look at the diffusion behavior of the particles under our force field.

During the simulation, every timestep LAMMPS writes the energy, pressure, and kinetic temperature into `log.lammps` and every 5th timestep saves the configuration to `result.traj`. For visualizing these files we have provided a jupyter notebook called `Analysis_scripts.ipynb`. You can open it in the jupyter notebook browser of the second docker container.

3.3 Additional things to do (optional)

Copy the best neural network potential (weight files) from Section 2 into the `nnp-data`-folder and perform a simulation with your own NNP! But be careful, your model is trained on a small dataset.

4 Training a Cu_2S Neural Network Potential

In this last section of the tutorial, you will learn how to prepare an input.nn file, including constructing descriptors - symmetry functions. Open /home/Day06_July18/Cu2S folder:

```
cd /home/Day06_July18/Cu2S
```

We do not have to start from scratch. Most of the parameters remain the same as in the previous NNP of water. The keywords.nn file contains all the necessary keywords for training, but the ones that may vary for each material are missing their arguments. Also the necessary symmetry functions are not prepared.

1. Your first task is to fill in the missing keyword arguments. Take inspiration from the water example, or look up their meaning on
   `https://compphysvienna.github.io/n2p2/topics/keywords.html`

2. To verify that the arguments are filled in correctly, create an input.nn file by merging the modified keywords.nn and the reference.nn containing reference symmetry functions. This can be done very easily by using cat in the terminal or by merging files manually:

```
cat keywords.nn reference.nn >input.nn
```

3. Then repeat the procedure from 2.1 using input.data for the training/validation dataset and train a model. The train-validation.data file is again provided (energy in eV and forces in eV/Å). Your task is to get a working Cu_2S neural network potential (do not waste too much time on hyperparameter optimization). Find the model with the smallest errors, copy weights, copy test.data as input.data (provided in /home/-Day06_July18/Cu2S) and find out its testing RMSEs using the nnp-dataset tool.

```
/home/n2p2/bin/nnp-dataset 1
```

4. The supplied symmetry functions have been optimized for this material and will serve as a reference. Note the resulting errors for the test dataset which you got with nnp-dataset. Now create your own symmetry functions and compare the error on the same test dataset. If you are adventurous, you can try to create symmetry functions completely by yourself from scratch. Documentation can be found on
   `https://compphysvienna.github.io/n2p2/api/symmetry_function_types.html`.

   However, we do not recommend this approach due to its difficulty. To simplify this task for you, we have prepared a simple script that can generate a set of symmetry functions. You can achieve the accuracy of our reference symmetry functions with this generator. The generating script is called nnp-setup-poly.py and located in the /home/Day06_July18/Cu2S folder. It uses polynomial symmetry functions which are easier to read than those based on exponentials. The radial part has 5 arguments:

```
<element-central> 20 <element-neighbor> <rlow> <rcutoff> <subtype>

<element-central> .... element symbol of central atom
<element-neighbor> ... element symbol of neighbor atom
<rlow> ............... low radius boundary
<rcutoff> ............ high radius boundary
<subtype> ............ compact function specifier
```

   An example of a radial symmetry function is shown in Fig. 1 and defined by:

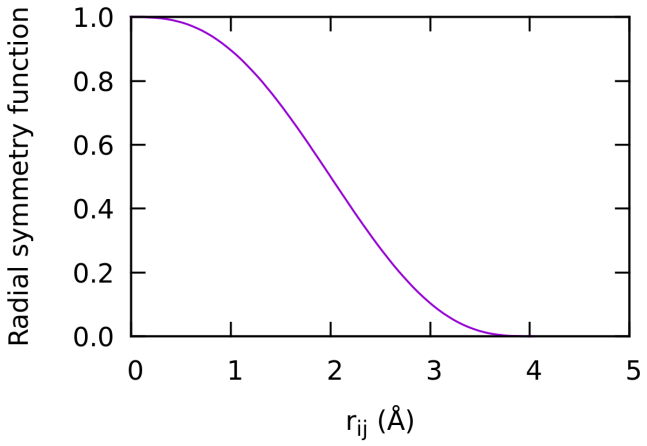

Figure 1: Example of a radial polynomial symmetry function

```
symfunction_short Cu 20 Cu -4.0 4.0 p2
```

Angular symmetry functions have 8 arguments since in addition to a radial part they also contain angle boundaries and a second neighbor element:

```
<element-central> 21 <element-neighbor1> <element-neighbor2> <rlow> <rcutoff
   > <left> <right> <subtype>

<element-central> ..... element symbol of central atom
<element-neighbor1> ... element symbol of neighbor atom 1
<element-neighbor2> ... element symbol of neighbor atom 2
<rlow> ................ low radius boundary
<rcutoff> ............. high radius boundary
<left> ................ left angle boundary
<right> ............... right angle boundary
<subtype> ............. compact function specifier
```

An example of an angular symmetry function is shown in Fig. 2 and defined by:

```
symfunction short Cu 22 S S -4.0 4.0 0 180 p2
```

The best way to understand the functionality of the nnp-setup-poly.py script is by using it. The script generates symmetry functions into a sym.nn file. The overview of its available arguments is shown by:

```
python3 /home/Day06_July18/Cu2S/nnp-setup-poly.py -h
```

A possible command may be for example:

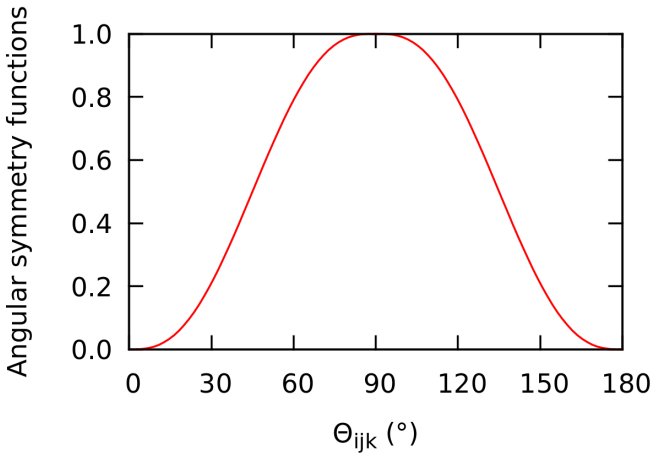

Figure 2: Example of an angular polynomial symmetry function

```
python3 /home/Day06_July18/Cu2S/nnp-setup-poly.py --ele Cu S --rcen 2 --rbeg
    3.0 --rend 6.0
```

After generating symmetry functions, do not forget to merge them with keywords.nn to input.nn by:

```
cat keywords.nn sym.nn >input.nn
```

5. An automatic generation has one drawback. The generated symmetry functions may use radial distances and angles that are never realized. For example, if the shortest interatomic distance is 3 Å and a radial symmetry function only samples distances up to 2 Å, the symmetry function will always evaluate to zero. The pruning of redundant symmetry functions can be easily achieved using the nnp-prune tool. First, you need to find the statistics of the symmetry functions using the nnp-scaling tool, then find the symmetry functions with zero values, remove these redundant functions, find the statistics of the new set of symmetry functions again, and then use this pruned set for training. This can be achieved for example by:

```
/home/n2p2/bin/nnp-scaling 100
/home/n2p2/bin/nnp-prune range 1E-04
cp output-prune-range.nn input.nn
/home/n2p2/bin/nnp-scaling 100
```

Where the second line prunes all symmetry functions whose values never reach 1E-04.

6. You should be able to construct your own symmetry functions now. Repeat the whole training process with different script arguments and try to beat the reference symmetry

functions. Changing the number of descriptors also changes the number of weights and biases. In practice, topology tuning should be performed for each new set of descriptors.

7. You can also prune symmetry functions after training by utilizing sensitive analysis. `https://compphysvienna.github.io/n2p2/tools/nnp-prune.html`

```
/home/n2p2/bin/nnp-prune sensitivity 0.5 max
```