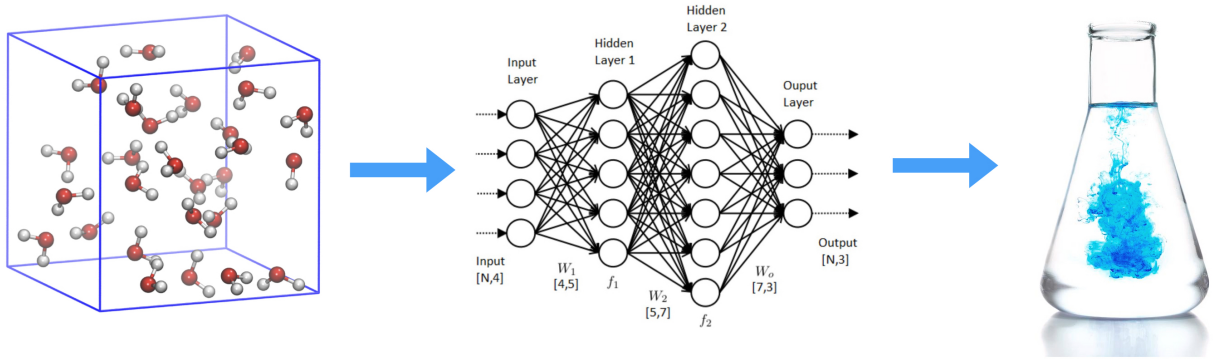
Molecular Dynamics meets Neural Networks: TUTORIAL



Veronika JURÁSKOVÁ
Frédéric CELERSE
Rubén LAPLAZA
Clémence CORMINBOEUF

Contents

1 Introduction

The aim of this tutorial is to provide a guidance for a building of the neural network-based potentials (NNPs) and their application in molecular dynamics (MD). As the traditional *ab initio* MD is too computationally demanding for the description of chemical reactions and processes in large systems, the MD with NNPs benefits from much lower cost of the simulation while the accuracy it the AIMD is conserved. In this regard, NNPs has been already used in the reactive simulations in gas phase and aqueous solutions (*e.g.*, in the description of proton transfer and urea hydrolysis).

Part 1 of this tutorial is dedicated to the generation of the Behler-Parrinello NNPs, which represent the most complicated task of this tutorial as the preparation of a large and robust data set is time consuming and the accuracy of the training needs to be carefully monitored.

Part 2 demonstrates how to use the NNPs using the i–Pi interface to LAMMPS and other codes. Finally, Part 3 introduces the Baselined NNPs and showcase its advantages compared to standard NNPs trained directly on DFT data.

2 Neural Networks-based potentials: Crash course

2.1 Feedforward Neural Networks

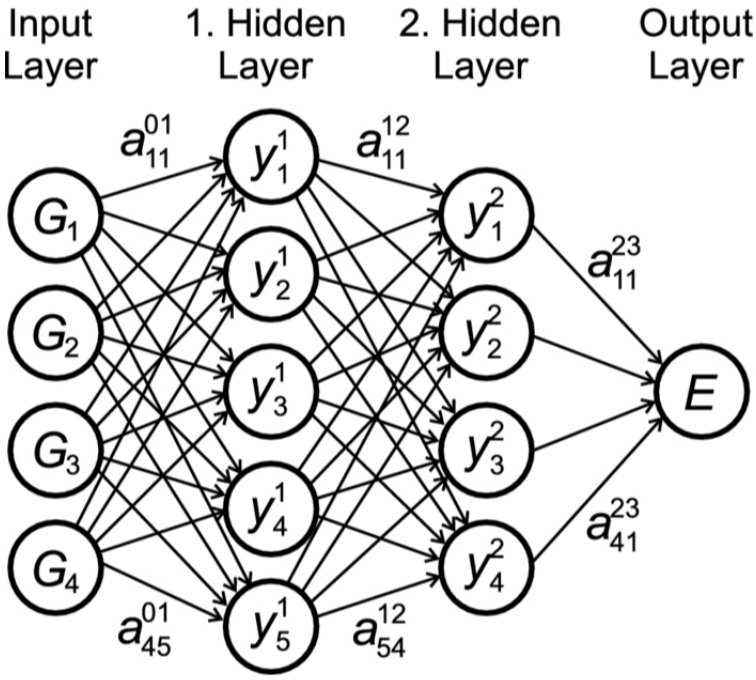


Figure 1: Schematic representation of a small feedforward NN. Nodes are interconnected in order to establish a functional relation between a set of coordinates G (input layer) and the potential energy of the structure E (output layer). Notation of this NN is 4–5–4–1.

Neural Networks are statistical learning models which are able to reproduce complex non-linear mathematical functions, *e.g.* the potential energy surface of molecular systems. Using the so-called **data set**, the aim is to train the Neural Network Potentials (NNPs), which are used to propagate atom dynamics by estimating the shape of the potential energy surface. The forces are computed analytically as a negative gradient of the potential energy. As depicted in Fig. 1, the feedforward NN is made from number of nodes organized in several interconnected layers. The potential energy E corresponds to the node in output layer and the atomic coordinates of a frame are provided in the input layer as a set of vector of input coordinates $G = \{G_i\}$. The input and output layers are connected via several so-called hidden layers with no physical meaning. The number of layers and the nodes defines the flexibility of the final NN. The example shows the NN with two layers containing five and four nodes, respectively. The short notation of such a NN is written as 4–5–4–1. The explanation of the notation is the following:

- 4: Four nodes represent the input layer (G_1, G_2, G_3 and G_4);
- 5: Fives nodes in the first hidden layer;

- 4: Four nodes in the second hidden layer;
- 1: One node (the potential energy) in the output layer.

Each node is connected to all the nodes of the next layer with a specific weight a_{ij}^{kl}, where i is the index of node of layer k, which is connected to the node with index j of layer l. For instance, the a_{45}^{01} connects the nodes G_4 ($k = 0$, $i = 4$) and y_5^1 ($l = 1$, $j = 5$). All the connections between nodes follow the same nomenclature. Not represented in our scheme, all the nodes in hidden and output layers are connected to so-called bias node, which scales their values by a bias weight b_i^j, where j is the layer of the target node and i the node index.

The value y_i^j of any node is computed as:

$$y_i^j = f_i^j(b_i^j + \sum_{k=1}^{N_{j-1}} a_{k,i}^{j-1,j} y_k^{j-1}) \tag{1}$$

The equation is a linear combination of the values of all nodes in the previous layer scaled by the bias weight. The term f_i^j represents the **activation function** of the NN. The activation function ensures the ability to fit complex non-linear functions. The examples of activation functions are:

- hyperbolic tangent,
- sigmoid function,
- Gaussian function,
- Softplus
- trigonometric functions
- exponential function.

According to Figure 1, the complete functional form for the energy is given by:

$$E = f_1^3(b_1^3 + \sum_{k=1}^{4} a_{k1}^{23} f_k^2(b_k^2 + \sum_{j=1}^{5} ajk^{12} f_j^1(b_j^1 + \sum_{i=1}^{4} a_{ij}^{01} G_i))) \tag{2}$$

2.2 High–dimensional NNP

Feedforward neural network discussed in previous section suffer from several drawbacks which prevented their application in the simulation of complex chemical systems. The limitations are for instance:

1. absence of symmetry between equivalent terms (for instance, the OH bonds of a water molecule)
2. system size dependency: no atoms could be deleted or added as it causes problems in the connecting weights).

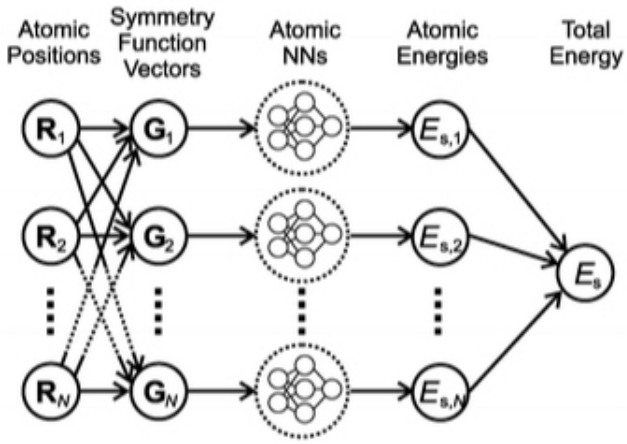


Figure 2: High dimensional NN scheme: Atomic positions are represented by a vector of symmetry functions G_i.

In 2007, Behler and Parrinello proposed a solution to this problem by expressing the total energy E_s as a sum of the atomic energy contributions E_i.[1]

$$E_s = \sum_{i=1}^{N_{atom}} E_i \tag{3}$$

Starting from the Cartesian coordinates R, every atom is represented by a set of atom-centered symmetry functions G_i, which describe the chemical environment of the atom within given cutoff. Each atom is then assigned a feedforward NN yielding the atomic contribution to the total energy. The weight of the NN are identical for atoms of the same element to ensure that the trained NNPs do not depend on the number and order of the atoms.

2.3 Atom-centered symmetry functions

Together with the NN topology, the seminal work of Behler and Parrinello introduced also new type of the symmetry functions to characterize the local environment of each atom. The so-called Atom-centered symmetry functions (ACSFs)[2] are constructed from the positions of the atoms and their closest neighbours. This representation guarantees that the atoms with the same environment yields the same atomic energy contribution which is also translationally and rotationally invariant.

The typically used ACSFs are two-body radial and three-body angular functions in a form:

$$G_i^2 = \sum_i e^{-\eta(R_{ij}-R_s)^2} \cdot f_c(R_{ij}) \tag{4}$$

$$G_i^3 = 2^{(1-\xi)} \sum_{j,k\neq i}^{\text{all}} (1+\lambda \cos\,\theta_{ijk})^{\zeta} \cdot e^{-\eta(R_{ij}^2+R_{ik}^2+R_{jk}^2)} \cdot f_c(R_{ij}) \cdot f_c(R_{ik}) \cdot f_c(R_{jk}), \tag{5}$$

where f_c is a cutoff function ensuring the smooth decay of the symmetry function to zero.

In this tutorial, we use the original Behler-Parrinello ACSFs. However, during the last years many modifications to the original ACSFs were proposed as well as new types of functional forms to describe the atomic environments. Various types of structural representations are summarized for example in Ref. 3.

2.4 Constructing and training NN

The general protocol to construct NNP for is as follows:

1. Define an initial set of structures and compute the reference energies and forces. The electronic structure method used as a reference should be accurate enough to describe the studied problem.

2. Train first versions of NNP. The fraction of the data set (*e.g.* 80 %) is used in the training (so called training set) while the remaining part is used as a test set to validate the error on unseen data. Ideally, several different NNP with different training sets should be trained to identify potential problems in the training set.

3. Perform preliminary simulations using the NNPs to evaluate the stability of the NNP and identify underrepresented areas of the PES, *e.g.* structures triggering extrapolation warnings or unphysical geometries.

4. Isolate the problematic structure, compute reference data and add them to the training set and re-train the NNP.

5. Repeat the validation and re-training of the NNP until no instabilities are found and NNP are ready to be used.

For detailed general tutorial review on Behler-Parrinello NN, see Ref. 4.

3 Requirements

This tutorial presents a workflow which requires extensive number of different software packages, scripts and libraries. The versions of the codes used here are listed below.

3.1 Software

All the codes are available at GitHub (see the last section of tutorial):

- n2p2 (version 1.0.0)
- LAMMPS (version 2)
- i–PI (version 2.0)
- Tinker (version 8.8)
- cp2k (version 6.1)

3.2 Dependencies

- python (version 2.7 and 3.6)
- libraries intel, intel-mpi, intel-mkl, gsl, eigen, gc,c mvapich2, openblas, n2p2/lib

4 Part 1: How to generate neural network potentials

Instructions

Currently, there is no universal NNP which is applicable to model all classes of systems. Despite ongoing effort in the training of general neural network-based force-fields, many system require building of the NNPs from scratch. In the following section, we provide general instructions how to train NNP using Behler-Parrinello NN. The required steps include:

1. Sampling of the phase space to produce initial training structure;
2. Generating large pool of symmetry functions;
3. Identification of representative set of fingerprints using CUR decomposition;
4. Careful selection of small set of structures for the training of NN;
5. Computation of reference forces and energies for all the selected structures;
6. Training of the NN.

The training protocol is summarized in the Fig. 3.

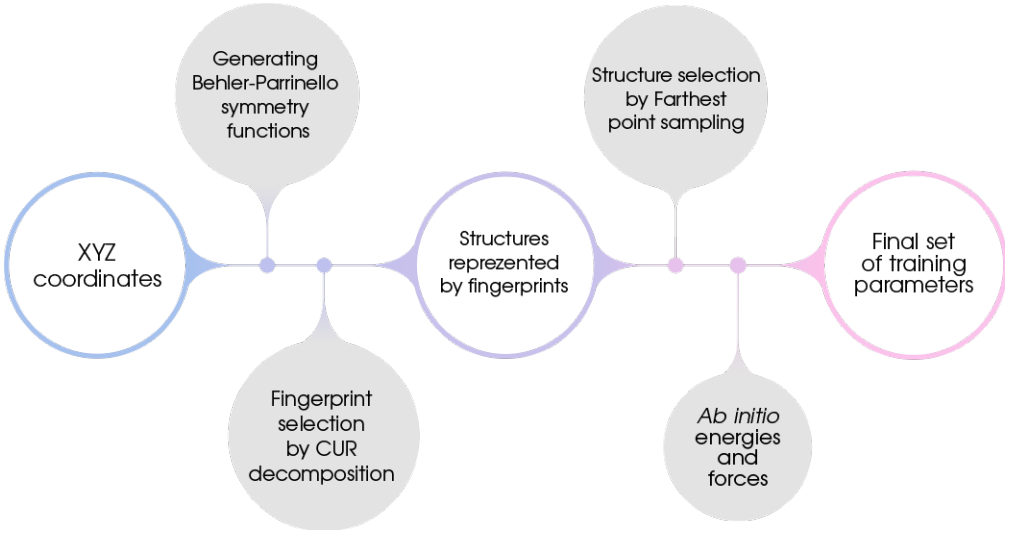


Figure 3: Scheme of the NNP preparation.

4.1 Step 0: The system

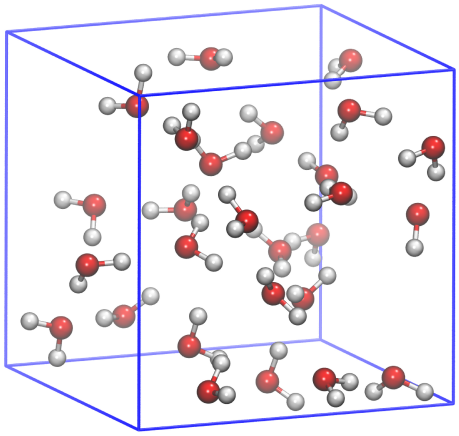
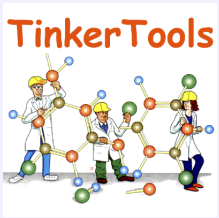


Figure 4: System containing 27 molecules in a cubic box with edge of 9.32 Å.

As a relatively simple test case, we train the NNPs for a small water box containing 27 water molecules with the cell size of 9.32 Å. This systems contain only two elements, *i.e.* H and O, which significantly simplifies the training.

4.2 Step 1: Generating the data

Files needed



- system.xyz
- system.key
- system.dyn
- water03.prm
- *dynamic* module

The first step in the tutorial is the preparation of the training set. The training set can contain systems with different size and different compositions. To keep things simple, we will train the NNPs using only one size of the water box defined above. To generate extensive set of the structures, it is beneficial to use molecular dynamics starting either from equilibrated structure or from experimental data (*e.g.*, X-ray). Random displacement of the atoms could be also applied.

As the first training set should be large enough but also contain representative geometries, the potential for the exploratory dynamics should provide reasonable cost/accuracy ratio. For example, classical force-fields might not be the best choice, as they can sample structures which are too different from the ones obtained at the reference level. The suitable compromise are therefore semiempirical methods or polarizable force-fields.

In this tutorial, we generate the system using AMOEBA polarizable force-field as available in Tinker. The files required for the simulation are *.xyz, *.key and *.prm. For more information on the preparation of the files for Tinker, see tutorials listed in section 8.

Here, we just briefly comment on the content of the files:

- The .xyz file contains the Cartesian coordinates of the initial water box in the Tinker xyz format (Caution! Tinker xyz differs significantly from standard xyz or extended xyz formats.)
- The .key file has the role of an input file. It contains the settings to run the dynamics.
- The .dyn serves as a restart file. It contains positions, velocities and accelerations of the last frame of the dynamics.
- The .prm file corresponds to the AMOEBA parameters. It contains all the terms to compute the potential energy of a specific structure.

MD using Tinker can be launched as:

dynamic system 1000000 2.0 1 2 300 > system.out

This command means that the simulation will run using the *dynamic* module for 1000000 steps with a time step of 2.0 fs (*i.e.*, the total length of the simulation is 2 ns). The next term specifies the frequency (in ps) of saving the trajectory to the .arc file, here we save it every 1 ps. The final system.arc file will thus contain 2000 frames. The following term specifies the mode of the dynamics, 2 corresponds to NVT canonical ensemble. The last term corresponds to the temperature of the simulation, here it is set to 300 K.

Enhanced sampling

⚠As our system is quite simple, the standard molecular dynamics is sufficient to provide extensive sampling. In more complex cases, however, the simulations starting from different initial structures or using enhanced sampling techniques might be needed to ensure that all the relevant structures are present in the training set. Suitable simulations techniques include for instance:

1. Replica exchange molecular dynamics
2. Sampling techniques using bias, such as metadynamics, steered molecular dynamics, umbrella sampling, ...
3. Transition path sampling

Extending the database with more distorted structures

⚠ The NNPs have in general very limiting extrapolation capacity. As their are trained only on the information included in the training set, it is beneficial to consider more distorted structures covering the repulsive and dissociative parts of the PES. Distorted structures can be generated by simulations using for example:

1. High temperature
2. High pressure
3. Path integral molecular dynamics
4. Specific constraints and restraints
5. Random displacement

Output Files

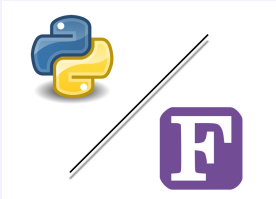
- system.out
- **system.arc** (file containing the structures)
- system.dyn

4.3 Step 2: Descriptor selection

In order to assign the structures with suitable set of the descriptors, we need to generate large pool of the symmetry function and select small representative set with minor correlations. In the following section, we therefore demonstrate how to generate the initial ACSFs and apply CUR decomposition scheme.

4.3.1 Generating symmetry functions

Input Files, Executables and Scripts



- system.arc
- from_txyz_to_n2p2.f90
- symmetry_functions.py

Let's assume that all generated structures are present in system.arc file.
Structures in this file can be converted to input files for n2p2 as follows:

1. compile the fortran code from_txyz_to_n2p2.f90: *gfortran from_txyz_to_gen.f90 -o from_txyz_to_gen*
2. place the system.arc file in the same directory of the from_txyz_to_gen executable and launch it as *./from_txyz_to_gen*

The script generates input.data file for the n2p2 containing all the structures. The definition of every structure starts with a short header:

begin
comment
lattice 17.612 0.0 0.0
lattice 0.0 17.612 0.0
lattice 0.0 0.0 17.612

"begin" and "comment" are present at the beginning of every new structure. "lattice" defines the dimension of the box. The units of the box and Cartesian coordinates are converted to Bohr. Each atom of the structure is defined as:

atom -4.367 5.267 4.603 O 0 0 0.0 0.0 0.0

The keyword "atom" specify that the line corresponds to one atom, -4.367, 5.267, 4.603 are the xyz coordinates of the atom, O is the element label, the following two columns 0 0 are not used and, finally, the last three columns 0.0 0.0 0.0 are the xyz force components acting on the atom. At this example, forces are set to 0.0 since they are not used.

The file ends with:

energy 0.000
charge 0.0
end

where "energy" is the reference total energy of the system, "charge" is the total charge of the system, and "end" closes the definition of the structure.

Units in n2p2

⚠ As indicated on the n2p2 website, the units correspond to the physical units defined in the input files. As we are combining several different codes which might have different unit specifications, keep track on the units and stay consistent within the workflow. In the inputs for n2p2, we use units as:

1. lattice: **Bohr**
2. x/y/z positions: **Bohr**
3. x/y/z forces: **Hartree Bohr^{-1}**
4. energy: **Hartree**

The conversion from Angström to Bohr is done by a multiplication by a factor **1.88973**

All settings and parameters for the training of NNP are specified in **input.nn** file. It contains three main parts:

1. GENERAL NNP SETTINGS
2. ADDITIONAL SETTINGS FOR TRAINING
3. SYMMETRY FUNCTIONS

GENERAL NNP SETTINGS specify the general set-up of the neural network, for instance number of the elements, layers and nodes, cut-offs, scaling normalization etc.

ADDITIONAL SETTINGS FOR TRAINING defines explicitly parameters of the training like number of epochs, fractions of energies and type of the optimizing algorithm.

Last part defines the symmetry functions sets used in training. While the parameters for the training can be find in the literature and possible modification directly depend on the simulated system, the symmetry functions are often build completely from scratch and require careful selection.

The n2p2 currently supports Behler-Parrinello types of symmetry functions (radial, angular, wide angular, for definition see Ref. 2), their weighted variants (Ref. 5) and polynomial symmetry functions (Ref. 6). In this tutorial, we use standard Behler-Parrinello functions.

Initial set of the symmetry function can be generate by symmetry_functions.py as

```
python symmetry_functions.py > asfs.txt
```

ASFs are thus generated and stored in the asfs.txt file, which is then appended to the input.nn as

```
cat asfs.txt >> input.nn
```

The initial input files needed for the training are therefore:

- **input.data**
- **input.nn**.

Output Files

- **input.data**
- **input.nn**

4.3.2 Launching training of the NN

Input Files, Executables and Scripts

- input.data
- input.nn
- nnp–scaling
- CurSel-integral.py
- curlib.py / iolib.py / sflib.py

```
mpirun -n 32 nnp-scaling 500 > logfile
```

Libraries

- module load intel intel-mpi intel-mkl gsl eigen is adviced !
- module load n2p2 if available !
- specify your n2p2 libraries path as: "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<path-to-your-n2p2-lib>"

3 output files are thus generated:

1. **logfile**: all instructions occuring in the nnp–scaling procedure are listed in this file. It is also very useful as it evaluates the memory for all structures.
2. **function.data**: all the functions for our system are listed in this file, which is very memory consuming. So do not try to open it manually.
3. **scaling.data**: nnp–scaling procedure compute all symmetry functions for all atoms once and store statistics about them in this file, required after for training.

Among all the ASFs generated, it is now essential to choose a reasonably as well as nonredundant set of ASFs which describe as accurately as possible our system. This task is often the most delicate and time–consuming aspects in the construction of a Behler–Parinnello style MLP. We thus decided to automatize this selection using CUR description and implemented in a home script named CurSel-integral.py. You could just place this script in the same direction as function.data and launch it as:

python CurSel-integral.py function.data logfile -n 64 –landmarks 1000 –prefix cursel > out

As before, 3 output files are generated:

1. **out**: the output file of the procedure.
2. **cursel.def**: the chosen symmetry functions.
3. **cursel.landmarks**: the frame's number selected.

The **cursel.def** will be useful in the STEP 4 and **cursel.landmarks** in STEP 3.

Checking of the ASFs !

⚠Generated ASFs should be checked by plotting them in an external software (gnuplot) in order to check if they well cover the space or not. A good balance between G2 and G3 has also to be respected !

Stability of NNP !

⚠ Although a good balance between G2 and G3 is needed, some other parameters are important to consider. One of them is the number of structures within the dataset. In our case, with only 1000 structures we can not wait for a stable NNP as the number of structures should not be sufficient enough. However, we kept this number to ensure a good ratio between understanding and short computational time for methodological understanding within this tutorial. However, this setup should not be used for production. If the user is then, after this tutorial, interested to use it for production, a setup used recently for production is provided at the end of this tutorial !

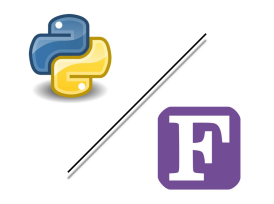
Output Files

- logfile
- function.data
- scaling.data
- out
- **cursel.def**
- **cursel.landmarks**

4.4 Step 3: Hybrid DFT force–energy computations

4.4.1 Extracting structures of interest

Input Files, Executables and Scripts



- cursel.landmarks
- input.data
- selection.f90
- from_data_to_xyz.f90
- from_xyz_to_cp2k.f90

From the cursel.landmarks file generated in the previous step, we need to extract structures in order to perform single point reference computations. From the input.data file you could apply a home script selection.f90 as:

```
gfortran selection.f90 -o selection ; ./selection
```

It will generate a newfile labeled as **new–input.data** and containing the 1000 selected structures from the CUR procedures.

The next steps is now to extract each structures and to place them in an individual file for enabling cp2k computations. This procedure corresponds, once again, to post–processing and could be performed in a two step way following this procedure:

1. *gfortran from_data_to_xyz.f90 -o from_data_to_xyz ; ./from_data_to_xyz*
2. *gfortran from_xyz_to_cp2k.f90 -o from_xyz_to_cp2k ; ./from_xyz_to_cp2k*

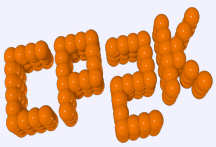
You obtain a series of 1000 files named $i-cp2k.xyz, which corresponds to the 1000 initial structure files for cp2k computations.

Output Files

- **$i-cp2k.xyz**

4.4.2 CP2K computations

Input Files, Executables and Scripts



- $i-cp2k.xyz
- input.cp2k
- prep-file.sh

We firstly need to prepare one directory for one calculation. This task could be automatized using the prep-file.sh bash file which automatically create you 1000 directories, one per frame. Each of them contain one .xyz file and .cp2k file. The input.cp2k file is where we place specific keywords to parametrize the cp2k computations. In our case, we are using the CP2K 6.1 version to perform reference energies and forces at the PBE–D3BJ level of theory. All elements are described with the TZV2P–MOLOPT basis set with cores represented by the dual–space Goedecker–Teter–Hutter pseudopotentials (GTH PBE). The plane–wave cutoff is set to 700 Ry with a relative cutoff of 70 Ry. All computations employ a Coulomb operator truncated at R=6 Å and the auxilary de,sity matrix method with a cpFIT3 fiiting basis set.

To launch a cp2k calculation, just type in the directory of your .xyz and .cp2k files the following command:

cp2k.popt -i input.cp2k -o output.cp2k

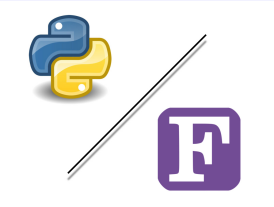
A series of files will be generated, but the most important in our case is the **output.cp2k** file. This task has to be performed for the 1000 structures, **each calculation is taken 3 minutes on 28 CPUs approximatively**.

Output Files

- **output.cp2k** 1000 times !

4.4.3 Extraction of forces and energies

Input Files, Executables and Scripts



- output.cp2k 1000 times !
- extraction.tcl
- trait.sh

From each output.cp2k generated in the previous step, we need now to extract the potential energy term and x/y/z component forces acting on each atoms of the structure. A classical tcl file named extraction.tcl has been designed to automatized this task. In the directory of your output.cp2k you could just launch:

tclsh extraction.tcl

It will generate an outputfile named **output-forces.dat**. The procedure can be easily automatized for the 1000 directories by using a classical bash file named trait.sh and launched in the directory of the 1000 directories (and one copy of the extraction.tcl file) as:

sh trait.sh

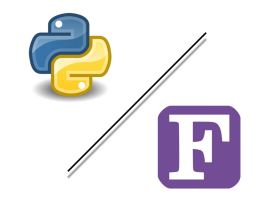
It will generate a final output named **forces_and_energies.dat**, containing all the information of your 1000 structures.

Output Files

- **forces_and_energies.dat**

4.4.4 Updating the input.data file

Input Files, Executables and Scripts



- input.data
- forces_and_energies.dat
- update.f90

After obtaining forces and energies for each of our structures, we need to inject them within our initial input.data file. A home script named update.f90 has been designed to perform this task and could be launched in the same directory as the input.data and forces_and_energies.dat files as:

gfortran update.f90 ; ./a.out

You now have to replace your old input.data by the **new-input.data** generated as the output of your script for the next step.

Output Files

- **input.data** (renamed after being new-input.data)

4.5 Step 4: Neural Network Potential training

4.5.1 Regenerate new scaling.data file

Input Files, Executables and Scripts

- input.data
- input.nn
- nnp–train

Before proceeding to the Neural Network training a new scaling.data has to be generated, with use of the new files created during the previous steps:

- input.data, coming from the "Updating the input.data file";
- input.nn, coming from "Preparing files for n2p2"

You create a new directory encompassing these two files and applied the nnp–scaling procedure similarly to the point "Preparing files for n2p2" in order to generate the right scaling.data file.

Output Files

- **scaling.data**

4.5.2 Training our Neural Network

Input Files, Executables and Scripts

- input.data
- input.nn
- scaling.data
- nnp–train

Once this step done, you create another directory encompassing this time:

- input.data

- input.nn
- scaling.data

and you launch your training procedure as:

mpirun -n 32 nnp–train > output.txt

Libraries

- module load intel intel-mpi intel-mkl gsl eigen is adviced !
- specify your n2p2 libraries path as: "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<path-to-your-n2p2-lib>"

Several files are created, but the most interesting are:

- **output.txt**: it is the main output file of the calculation. Initial parameters for the training procedure is listed to avoid some troubles in the initial setting. Sum of the training computations each epoch are listed with for each one the calculation time.
- **learning–curve.out**: this is the main file to easily plot the different learning curves using an external software.
- **weights**: for each atom a weight file is created. For instance as H is 1, for the epoch 120 you should obtain "weights.001.120.out". These files are a sum of all the weights used in the Neural Network connections (see the few reminder part on the NN). These files are crucial when the procedure should be restarted (see next subpart).

Be focused now on the **learning–curve.out** file. It is presented in 5 columns:

1. Column 1: the current epoch
2. Column 2: RMSE of training energies per atom (physical units)
3. Column 3: RMSE of test energies per atom (physical units)
4. Column 4: RMSE of training forces (physical units)
5. Column 5: RMSE of test forces (physical units)

To ensure the stability of our NN, we have to check the RMSE values obtained in function of the epoch. This could be performed in two steps:

1. Firstly, we check the quality of the energy prediction: a good compromise should be an error around 1 meV/atom. If it is higher, be careful and you need to check the stability of your NN when you will branch it to your MD.
2. Secondly, we check the quality of the forces prediction: as forces are more complicated to be predicted compared to the energy, a good compromise between accuracy and prediction should be an error around 150 meV/atom. As for the energy, if its higher you need also to check the stability of your NN when it will be branched to your MD.

Results obtained in this file are in the cp2k units, so in our case it is in a.u (atomic units, you could easily check it in an output.cp2k you obtained previously). It is thus desirable to translate a.u in meV/atom.
Conversion: 1 a.u = 627.5 kcal/mol = 27.211 eV, so to pass from a.u to meV the factor to apply is 27211 for the energy. Concerning the forces, a factor 2 has to be added. As results are average taken on the atoms, the final result is in meV/Atom (so do not need to divide it by the number of atoms). Using an external software to plot the curves (python, gnuplot, xmgrace, excel, ...) you should be able to obtain these curves depicted on Figure.

Ratio between training and testing

⚠ Be careful between the ratio taken into account in the training procedure, especially when you have a small database. The keyword in the input.nn responsible for such a parametrization is *test_fraction* and is set to 0.2 in our case. It means that 80% of our total structures (2400 in this case) will be used for training while 20% (600 in this case) will be used for testing. Normally, no strong differences should be observed between training and testing curves.

Interpretation tool !

- Errors for the energy and forces predictions are comprised around 1meV/Atom and 75meV/Atom. This is good when we compare it to our reference step (1 meV/Atom and 150 meV/Atom).
- No differences are observed between testing and training curves for the energy and forces.

Output Files

- **learning-curve.out**
- **weights.001.001000.out**
- **weights.008.001000.out**

4.5.3 How to restart a Neural Network training ?

Imagine either the simulation time was not enough to complete your calculation or you need to add more than the initial number of epoch, it should be desirable to be able to restart our calculation from the last iteration (here in this case epoch=1000). To perform this task, follow this procedure:

1. create a new directory
2. copy the input.data input.nn scaling.data weights.001.0001000.out weights.008.0001000.out files in this new directory
3. move to this new directory

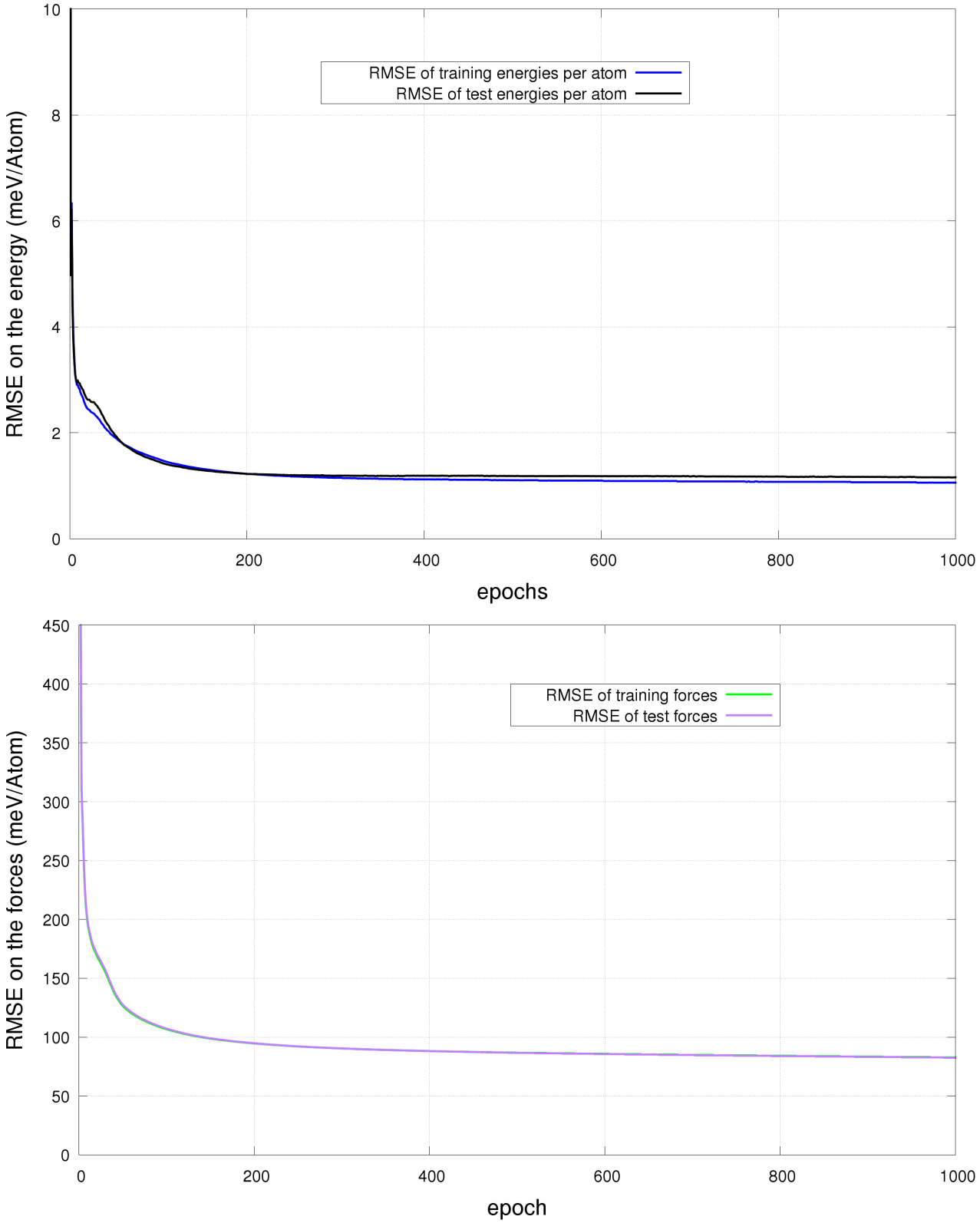


Figure 5: Energy (direct) and forces (direct) learning curves. The training has been performed on 800 structures and testing on 200 structures.

4. add (or delete the # symbole) the following keyword in the ADDITIONAL SETTINGS FOR TRAINING: *use_old_weights_short*

5. launch your calculation similarly to the previous one using the nnp–train executable.

5 Part 2: Link the direct NNP to Molecular Dynamics

We finally generated our first NNP set. In order to improve it, we need to run it to detect non–physical structures. These ones will be then added to the NNP, which will be retrain and the procedure should be performed as a SCF procedure until no new structures are detected. In our case, we would like to see how to link a "ready" NNP, this is why we will not use our NNP but another one already available in the n2p2 examples directory *interface-LAMMPS/H2O_RPBE-D3/nnp-data*.

5.1 i–PI code

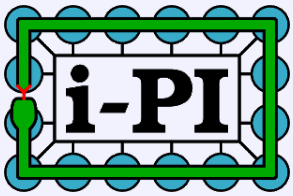
Since we have a NNP able to reproduce accurately and fastly the forces and energies of water structures, it should be desirable to couple that to a Molecular Dynamics software. Two options are available to couple n2p2:

- The LAMMPS software
- The CABANA software

Our choice in this tutorial is focused on the first, the LAMMPS software. Once this choice of coupling is done, we need now to efficiently couple both to optimize the simulation time. In this direction, the software i–PI is clearly designed to perform this task as it corresponds to a universal force engine for advanced molecular dynamics. i–PI is an interface for advanced molecular simulations written in Python, designed to be used together with an ab initio evaluation of the interactions between the atoms. The main goal is to decouple the problem of evolving the ionic positions to sample the appropriate thermodynamic ensemble and the problem of computing the inter-atomic forces. It could be thus linked to several software such as LAMMPS, which is our main interest in this tutorial. One other interest to use the i–PI interface is that it suitable for high sampling (Replica–exchange) and free energy computations (linkage with PLUMED available).

5.2 Launching the i–PI interface

Input Files, Executables and Scripts



- input.xml
- system.xyz

i–PI was designed for python 2 (even if a new version will be available as soon as possible for python 3), so please used for instance python2.7 to deal with it. Assuming the i–PI is in your directory, before any calculation you have to type in your directory the following command in order to launch the i–PI interface:

```
python2.7 i-pi input.xml >> ipi.out &
```

It will create an output file named ipi.out, and the i–PI interface will be waiting for any actions you will submit. This will be described in the next subsection.
Just a point on the input.xml file. It corresponds to the input file of i–PI and contain all the main informations of the MD, such as temperature, timestep, periodicity, units, thermodynamic ensemble, ...). You could feel free to change it if the conditions are not the same for another system of interest. The most critical block in this file is localized in the **prng** block with:

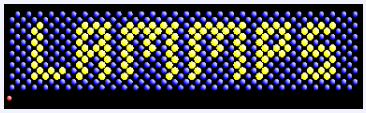
- **socket**: keep driver–lammps1 if LAMMPS will be coupled to i–PI.
- **address**: it corresponds to the adress of the node where i–PI will be launched.
- **port**: the port of the node.
- **slots**: do not touch it.
- **timeout**: time for killing calculation without having any communications.

Output Files

- **ipi.out**

5.3 Linking n2p2 with LAMMPS

Input Files, Executables and Scripts



- lmp1.in
- system.xyz
- get_lammps.sh

Once i–PI is running and is waiting for communication with another software(s), we can initialize LAMMPS computation which will serve as an intermediate with n2p2 to compute energy and forces at every timestep of the dynamics and communicate them to i–PI for forces integration. We thus start with classical LAMMPS files, which have to be:

- An input file (.in) where keyword for MD is specified,
- A coordinate file (.data), which is specific for LAMMPS.

Starting from a classical xyz structure, we could translate it in .data file by applying the get_lammps.sh script as:

sh get_lammps.sh system.xyz

Just adapt (if necessary) the script file in function of your system ! In our case, it is designed for our periodic box of water molecules, and an output file named **initial.data** is generated and correspond to the initial set of coordinates for the LAMMPS calculation. This file is called in lmp1.in as *read_data ./initial.data*, and mass objects are ordered in function of the

initial.data file. For the other lines, a classical example is provided in the lmp1.in and is close to every input file for MD (AMBER, NAMD, Tinker, ...). For more details, a tutorial of LAMMPS will be specified at the end of the tutorial.
While i–PI is running and the initial.data and lmp1.in are ready for a LAMMPS calculation, additional keywords have to be added in the lmp1.in to specify the connection between LAMMPS and n2p2:

- variable runnerCutoff equal 7.0000
- variable runnerDir string "/path/to/nnp-data"
- pair_style nnp dir ${runnerDir} showew no showewsum 1 resetew yes maxew 200000 cflength 1.889726 cfenergy 0.036749
- pair_coeff * * ${runnerCutoff}
- fix 1 all ipi 192.168.100.1 8766

The two first lines are variable declarations. ${runnerDir} is the directory of the nnp data and ${runnerCutoff} is maximum cutoff radius of all symmetry functions. **The cutoff must be given in LAMMPS length units, even if the neural network potential has been trained using a different unit system**.
The pair_style adds an interaction based on the high-dimensional neural network potential method. It uses an interface to the NNP library, which has to be declared before launching the calculation (see next subsection).
Finally, the fix line called the adress and the port where i–PI is running.

Output Files

- **initial.data**

5.4 Launching LAMMPS with the i–PI interface

Input Files, Executables and Scripts

- lmp1.in
- initial.data
- nnp–data directory
- lmp

Create the nnp–data directory and place in the following files: input.data, input.nn, scaling.data, weights.XXX.001000.out and change the respective weights001.001000.out and weights008.001000.out by weights.001.data and weights.008.data.
Once the lmp1.in and the nnp data directory are ready, just follow these instructions to launch your calculation:

1. export LD_LIBRARY_PATH=/path/to/n2p2/lib:${LD_LIBRARY_PATH}

2. lmp -i lmp1.in >> log1.lammps &

Files **simulation.force_0.xyz** and **simulation.pos_0.xyz** will be filled, according to the printing frequency we specified in the lmp1.in (here 1 at the "fix" line).

Interpretation tool !

- Simulation could be observed using either vmd or molden softwares. It is the first step to do in order to observe if the simulation is stable or not.
- From this simulation, new structures could then be extracted to improve the quality of the NNP, and training as well as simulations are then performed in a SCF procedure until no new structures are observed.

6 Part 3: From direct to baselined NNP

6.1 What is baselined NNP?

In the previous sections (parts 1 and 2) we learnt how to generate a NNP and how to use it within the i–PI interface. While the NNP trained directly on DFT data provide predictions at impressive speed, the stability and accuracy of these NNPs is often difficult to achieve. The problems with stability are probably the simplest ones to detect. The system simulated with non-stable NNPs has tendency to explode within the first picosecond of the simulation. The reasons might be:

1. **The quality of the generated data**: instead of generating initial structures with the Tinker MD software, we could use semi–empirical methods to generate structures more closely to *ab initio* dynamics.
2. **The choice of symmetry functions**: the number as well as the spacing chosen to generate the symmetry functions play a crucial role in the stability of the NNP.
3. **Number of data points**: Direct NNP require relatively large number of structures to provide stable dynamics. Low nuber of the structures can be reason for instabilities in sparsely represented regions of PES.

As demonstarted by the examples, stabilization of direct NNPs is not an easy task, even for a small water box. A suitable alternative is to train so called baselined NNP. Instead of reproducing directly the DFT forces and energy, the NNPs are trained to correct the forces and energy computed at a lower level of theory, for example semiempirical method. The baselined NNPs require lower number of structures and descriptors and more robust compared to the direct NNP. In the following part, we show how to adapt our NNP training protocol in order to use it as a "Baselined NNP".

6.2 Baselined forces and energy

Input Files, Executables and Scripts

- input.data
- xtb

While we already have the accurate reference energies and forces coming from the cp2k computations, we need to choose suitable baseline method. In this tutorial, we use semiempirical GFN0-xTB method. Therefore, energy and forces of the training structures need to be recomputed at this level.
To proceed to the computations, please follow these steps:

1. Create one folder per structure and extract the corresponded structure from the input.data file and place it in the corresponded folder (*prep.sh*)
2. Convert each frame (n2p2 format) in xyz format (*prep2.sh* linked with *from_data_to_xtb.f90*)
3. Submit interactively all the computations at the GFN0-xTB level of theory (*sub.sh*)
4. Once all the computations are ended, compute the differences between DFT and GFN0-xTB energies and forces for each frame and add the results in a new file called new–input.data (*calc–diff.sh*).

Output Files

- **new–input.data**

6.3 New training

Input Files, Executables and Scripts

- new–input.data
- input.nn
- nnp–scaling
- nnp–train

First, rename the new–input.data to input.data and place it in a new folder with the input.nn file. The procedure to train a Baselined–NNP is analogous to direct NNP:

1. Use the nnp–scaling executable (see Step 4) to generate the corresponding scaling.data

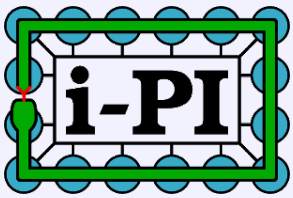
2. Use the nnp–train executable (see Step 4 again) to train Baselined–NNP.

Output Files

- **learning-curve.out**
- **weights.001.001000.out**
- **weights.008.001000.out**

6.4 Baselined–NNP with i–PI

Input Files, Executables and Scripts



- input.data // input.nn // scaling.data
- weights.001.001000.out // weights.008.001000.out
- input.xml
- initial.data // system.xyz
- lmp1.in
- xtb_ase.py // xtb_io.py

Once the training is finished, you can launch your Baselined–NN MD similarly to direct NNP. The main difference is that apart from the NNP itself, you also need to perform computation of the baseline ate every step. The submission is be done as follows:

1. Create a new directory labeled nnp–data and place there the input.data, input.nn, scaling.data and weightsXXX.data files from the last epoch
2. Change the name of the weightsXXX.data files in respectively weights.001.data (H) and weights.008.data (O).
3. Adapt the path of the nnp–data folder in the lmp1.in file
4. Also adapt the sockets in the respectively files:
   - input.xml
   - lmp1.in
   - xtb_ase.py
   - sub.run
5. Launch the simulation with the sub.sh script file !

New xtb–ase python interface

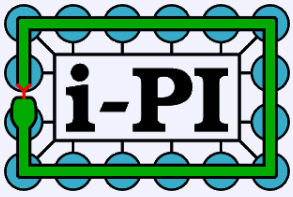
- The both files xtb_ase.py and xtb_io.py provide a new python interface between ase and xtb as the interface for xtb 6.3.3 encounters some issues ...
- This interface is a little bit primitive, so be careful about error messages ...

Interpretation tools

- This time, the simulation does not crash as it was the case for the direct NNP. It finally means that direct NNP is definitevely more sensitive to the choice of symmetry functions as well as initial structures encompassing the dataset.
- However, the EW messages present in the log.lammps are too high ... (around 50 EW per step in our cas). **A rule of thumb is to consider that a structure depicted more than 10/20 EW is badly described by the NNP**. It means that even if the simulation does not crash, the NNP is not optimal and should be optimized following the self consistent procedure described in section 2 of this tutorial.

6.5 Fast and accurate: Multiple-time-step approach

Input Files, Executables and Scripts



- input.data // input.nn // scaling.data from baselined
- input.data // input.nn // scaling.data from direct
- weights.001.001000.out // weights.008.001000.out from baselined
- weights.001.001000.out // weights.008.001000.out from direct
- input.xml
- initial.data // system.xyz
- lmp1.in
- xtb_ase.py // xtb_io.py

When direct NNP is not stable to fully run itself but not so bad in single point accuracy, it could be coupled to the baselined NNP to form what we call a mulit–time–step approach. In this approach, forces and energy are computed x times using the direct NNP and then one time with the baselined NNP. It helps to avoid some crashing from the direct NNP and

improves the speed of the simulation because the baselined is not computed at every step. Using the i–PI software, we will see how such a coupling is possible.

To run i–PI within such a mode (i.e multi–time–step), please follow the following instructions:

1. Create a folder and place the submission file, system.xyz and input.xml files
2. In this folder, create two new directories labeled nnp-data-direct and nnp-data-baselined and place in each respective folder the corresponding input.data, input.nn, scaling.data and the weights.XXX.data files
3. Place the lmp1.in in both the directories and adapt the respective path of the corresponding folder in each lmp1.in
4. Do not forget to place xtb_ase.py and xtb_io.py in the baselined directory
5. Adapt the sockets in input.xml and lmp1.in for the direct NNP
6. Adapt the sockets in input.xml, lmp1.in and xtb_ase.py for the baselined NNP
7. Launch the simulation with the sub.sh script file !

In our case, we launched a multi–time–step simulation with an inner timestep of 0.5 fs where forces and energy are computed with the direct NNP and an outer timestep of 3 fs where forces and energy are computed using the xTB baseline and corrected with the baselined NNP. All the relative keywords are mentioned within the input.xml and the structure of the file does not differ from the previous ones met during the tutorial.

Interpretation tools

- The accuracy of such a dynamic is very sensitive to the energy conservation during the dynamics. Therefore, the choice of 0.5/3 was made in a sens to ensure the conservation of the energy. However on other systems careful attention should be taken on this conservation.
- As mentioned before EW should still be checked but should be now lower compared to the direct NNP simulation.
- Finally, in order to improve the accuracy from the baselined more than 1 NNP can be considered during the dynamics. Indeed, i–PI enables to take the average made on more than 1 prediction. It has the main advantage to decrease the error made on the NNP prediction but impact the speed of the dynamics as more predictions has to be performed at the same time.

7 Conclusions

Thorough this tutorial we described the main idea on how to construct manually an efficient NNP and how to link it to a MD software (LAMMPS in our case). This procedure is straightforwardly transferable to every chemical systems and hope it will be helpful to include reactivity in conventional Molecular Dynamics.

8 Supplementary tutorials

A list of supplementary tutorials is given to help users if questions appear (espcially in the input files of several softwares).

8.1 Tinker(–HP)

- Frédéric CELERSE personnal tutorials (on request): cMD / preparing Tinker xyz files / Umbrella Sampling / Steered Molecular Dynamics / Gaussian accelerated Molecular Dynamics
- AMOEBA workshop: https://sites.google.com/site/amoebaworkshop/
- Tinker website: https://dasher.wustl.edu/tinker/
- GitHub (open source code): https://github.com/TinkerTools/tinker

8.2 n2p2

- GitHub (open source code): https://github.com/CompPhysVienna/n2p2
- Short tutorial: https://compphysvienna.github.io/n2p2/index.html

8.3 cp2k

- GitHub (open source code): https://github.com/cp2k/cp2k
- cp2k main website: https://www.cp2k.org/

8.4 i–PI

- GitHub (open source code): https://github.com/i-pi/i-pi
- i–PI website: http://ipi-code.org/
- i–PI manual: http://ipi-code.org/assets/pdf/manual.pdf

8.5 LAMMPS

- GitHub (open source code): https://github.com/lammps/lammps
- LAMMPS website: https://lammps.sandia.gov/
- LAMMPS tutorials: https://icme.hpc.msstate.edu/mediawiki/index.php/LAMMPS_tutorials.html

9 Optimal protocol for stable NNP

In this last section, we aim at providing optimal parameters to train a NNP which could be then used for production. Note that the proposed methods and parameters come from different works available in the literature and modifications might be needed.

SYSTEM CASE: The user want to model a direct NNP of an azobenzene in order to reproduce an *ab–initio* MD. In this case, the user has to separate two important components:

1. How he will built his own NNP ?
2. How he will train for ?

These two features are complementary but have to be treated carefully and separated. Let us see how we suggest to proceed for that !

9.1 How to build a good starting NNP ?

The complexity as well as the chemical event that any user would like to explore are two crucial parameters to take into account. Here, there is only 3 different atom types (H,C and N) and the system is made of 24 atoms. In case of the photoswitches, the event could be decoupled into two main parts:

1. **PHOTO**: a photo event comes from by an photo-excitation of the system through any wave length.
2. **SWITCH**: a switch event is related to the conformation switch of a molecule which is only due to thermal fluctuations.

In our case, the photo part needs some external excitations, which is not available in MD–NN based only on the ground state Potential Energy Surface. Concerning the switch part, we have two available conformations: E and Z. One interesting question should be thus: **Which mechanic way should enable the switch of azobenzene from E to Z ?**
To answer to this question, we need thus to prepare the NN according to this specific question. In our case, and according to the literature, azobenzene could depict two switch mechanisms: the inversion and the rotation. The aim is thus to let the choice of the desired path within our MD–NN to observe which way is preferred or not. Therefore, the NN has to know the existence of these two ways and structures taken to build the NN should be part of these pathways.
To generate these structures, you can follow the following instructions:

1. For each pathway, generate a metadynamics of several ns using a baselined (DFTB+, xTB) using i–Pi coupled to the baselined and plumed,
2. Extract the most relevant structures either manually or by using a clustering method (PCA, TICA, TSNE, ...)
3. Only for the most important structures, run a Replica Exchange MD of several ps in order to capture the fluctuation effects from the temperature. Extract as before the most relevant structures.

Once all the structures extracted, we can grab them in order to generate the first dataset which will be used as a starting point for the procedure. Note that a good compromise for the number of structures to start is localized between 5000 (for simple system) and 10 000 (for more complex system).

9.2 How to efficiently train a good NNP ?

Once the initial dataset was created, the other important part of the work is to design the architecture for future NNP.

References

1. Behler, J.; Parrinello, M. Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces. *Phys. Rev. Lett.* **2007**, *98*, 146401.

2. Behler, J. Atom-Centered Symmetry Functions for Constructing High-Dimensional Neural Network Potentials. *J. Chem. Phys.* **2011**, *134*, 074106.

3. Musil, F.; Grisafi, A.; Bartók, A. P.; Ortner, C.; Csányi, G.; Ceriotti, M. Physics-Inspired Structural Representations for Molecules and Materials. *Chem. Rev.* **2021**, *121*, 9759–9815.

4. Behler, J. Constructing high-dimensional neural network potentials: A tutorial review. *Int. J. Quantum Chem.* **2015**, *115*, 1032–1050.

5. Gastegger, M.; Schwiedrzik, L.; Bittermann, M.; Berzsenyi, F.; Marquetand, P. WACSF - Weighted atom-centered symmetry functions as descriptors in machine learning potentials. *J. Chem. Phys.* **2018**, *148*, 241709.

6. Bircher, M. P.; Singraber, A.; Dellago, C. Improved description of atomic environments using low-cost polynomial functions with compact support. *Mach. Learn. Sci. Technol.* **2021**, *2*, 035026.