Machine-Learned Potentials
Workshop 10, Data Science in Chemistry

Alan M. Lewis

1 Getting Started

To get started, you should log into teaching0 using `ssh -X your-username@teaching0.york.ac.uk`, using your usual University username and password. You should then clone the repository containing the data for this workshop by running

```
git clone https://github.com/alanmlewis/ml-workshop
```

This will create a new folder called ml-workshop; change directory into that folder. Everything else we do in the workshop will take place in this folder or a subfolder. Before you go any further, you should run the command

```
export OMP_NUM_THREADS=4
```

to ensure you only use 4 CPUs for each calculation. You will then need to install `gap_fit`, the software which we will use to predict energies and forces, by running

```
python3 -m pip install quippy-ase
```

This should install `gap_fit` and all of its dependencies. To test this has installed correctly, run

```
gap_fit config_file=gap_config.cfg
```

This program should take a few seconds to complete, during which time you should see a lot of text produced, with the words 'Bye Bye' displayed near the end of this text.

2 Learning Energies and Forces

The program `gap_fit` takes some input data stored in an xyz file, and trains a machine-learned potential which can be used to calculate the energy and forces of a similar target system given only the atomic positions of that target system. That input data consists of a series of snapshots of atomic positions, the associated energy of that snapshot, and the electronic forces which act on each atom. The potential produced is stored in a single file, called `gap.xml`. It is very straightforward to run the program from the command line: `gap_fit config_file=gap_config.cfg`. This reads a configuration file, here called `gap_config.cfg`, which specifies where the program should look for its training data and several other parameters which govern the machine learning model. These parameters include the SOAP parameters, kernel types, and an overfitting parameter:

- `atoms_filename`, which specifies the file which contains the training data.

- `cutoff` and `atom_sigma`, which determine the radius of the atomic environment and the width of the Gaussian used to represent the atoms in that environment, respectively.
- `l_max` and `n_max`, which govern the precision with which the atomic environments are quantified (larger integers = more precise).
- `covariance_type`, how the kernels are calculated. Options are `DOT_PRODUCT` (linear kernel), `ARD_SE` (Gaussian kernel), `PP` (piecewise polynomial kernel).
- `default_sigma` specifies how precisely we should fit to the training energies and forces (smaller values = more precise).
- `gp_file` gives the filename where the trained potential will be stored.

The following keywords must also be present in the config file, but refer to more technical details which we will not cover today and do not need to be changed.

- `n_sparse`, the number of atomic environments to use in the training. We will always use fewer than 8000.
- `delta`, scaling parameter of the kernel per descriptor.
- `zeta` determines the degree of non-linearity in the kernels.
- `e0` contains the energies of isolated H and O atoms.
- `sparse_jitter` controls the numerical convergence of the training.
- `sparse_separate_file=F` prevents unnecessary extra files being written.

We have covered most of these parameters in principle during the last lectures, and will now look at the effect of how varying some of these parameters affects our ML performance.

2.1 How accurate is my model?

In the `ml-workshop` folder, you will see two python scripts, called `gap_error_validate.py` and `gap_error_train.py`. These perform validation on two different sets of snapshots - a validation set contained in `gap_validate.xyz`, and the set of snapshots we used to train the model, respectively. These python scripts will return the root mean square error in the energy across the set of the snapshots. They also produce two files called `validation_errors.csv` and `train_errors.csv`, which contain the true values of the energy for each snapshot in the first column and the corresponding predicted energy in the second column. To run these scripts, just type `python3 script_name.py`.

Exercises

1. Train a GAP model using the default parameters in the `gap_config.cfg` file.
2. What is the RMSE in the predicted energies of the validation set?
3. What is the RMSE in the predicted energies of the training set?
4. Why are these values different?

2.2 Creating a Learning Curve

The most important way to test whether our machine learning method is working as we expect is to plot a learning curve - that is, a plot of the RMSE in the predictions of the model against the number of snapshots used to train the model. We expect to see the error decrease as the number of training snapshots is increased, up to some limit when the error stops reducing.

To help you plot a learning curve using `gap_fit`, a number of input files are provided labeled `gap_input_N.xyz`, where `N` is the number of snapshots in the training set. Each set includes all of the snapshots in the previous set, plus some new snapshots. You can select your training set by modifying `gap_config.cfg`.

Exercises

1. Train a model using 50, 100, 200 and 400 snapshots. After training each one, calculate the accuracy of each model on both the validation set and the corresponding training set, making a note of each error as you go.
2. Plot the learning curves for both the RMSE over the training set and validation set on a single graph. Does what you see match your prediction from the previous section?

2.3 Overfitting

Overfitting is a common problem when performing machine learning. It describes a situation where a model can extremely accurately reproduce the data in your training set, but in doing so is extremely unpredictable and inaccurate "between" the training points. This is illustrated in Figure 1.

Overfitting can be caused by a poor choice of a number of parameters. Some of these are directly connected to the idea of overfitting. For example, in `gap_config.cfg`, the first value in the variable `default_sigma` is an estimate of the standard deviation in the energy calculations. This effectively tells the model how closely it should try to fit to the training data. A small value here will almost always lead to overfitting. However, sometimes other parameters can lead to overfitting in less obvious ways. For example, having extremely accurate descriptions of the atomic environments (determined by `nmax` and `lmax` in `gap_config.cfg`) can lead to overfitting.

Note that if you want to save the GAP models you train with certain parameters, you will need to change the `gp_file` keyword before you run `gap_fit`, otherwise the model will be saved to `gap.xml` every time, overwriting the previous model. However, if you do this you will also need to modify the python files you run to ensure that you load the correct model (all python files will load `gap.xml` by default).

Exercises

1. How could you identify when overfitting is taking place?
2. Using 100 training snapshots, train models using different values of `default_sigma` for the energy and calculate the RMSE on the training and validation set each time. Do the results match your expectations? What is the optimal value, in your opinion?
3. Repeat this exercise varying the cutoff radius of the SOAP descriptors, the number of radial functions used to describe the SOAP environment, and the type of kernel used in the calculation. Do these affect the accuracy of the model? Can they cause overfitting? What is the optimal value, in your opinion?

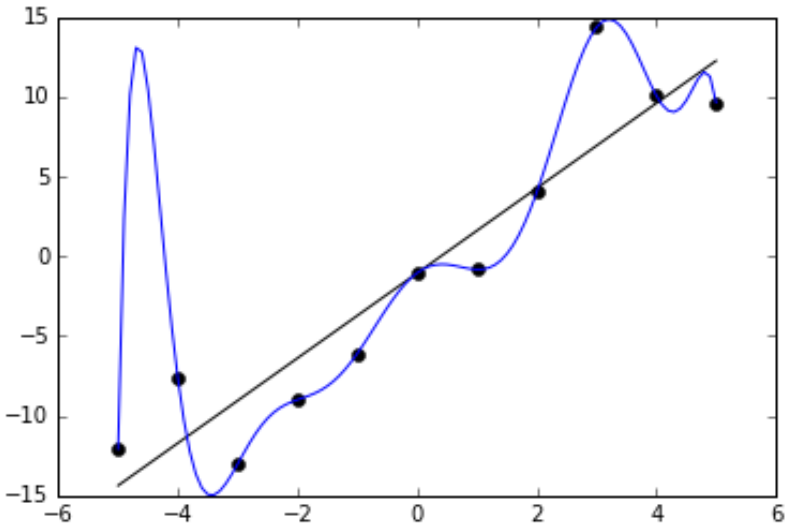

Figure 1: A graph illustrating the concept of overfitting. The blue line goes through all of the known points (training data) very closely, but using this function to interpolate values between the known results is likely to produce very poor predictions. By contrast, the black line does not reproduce the training data precisely, but gives more accurate interpolation. Image Credit: Wikipedia.

4. Once you have found the optimal value of some parameter(s), use these parameters to train a model using 200 snapshots. Does this lead to more or less overfitting than when you used 100 snapshots? Is the difference significant?

2.4 Running Molecular Dynamics with an ML Potential

So far we have only considered the energy predicted by the ML model, but the model also predicts the forces exerted on each atom by the electrons in the molecule. This means that once we have trained a machine learning model which we are happy is sufficiently accurate, we can use it to run molecular dynamics simulations. The simple script `molecular_dynamics.py` does exactly that: it takes a snapshot from the validation set as a starting point, randomly assigns velocities to each atom, and uses the machine learned potential to calculate the forces acting on each atom. That is all of the information required to perform a dynamics simulation, as we use these velocities and forces to calculate the atomic positions and velocities a short time later, and then recalculate the forces on the atoms in their new positions using the ML potential. Doing this repeatedly results in a molecular dynamics trajectory.

Exercises

1. Read through through the python script `molecular_dynamics.py` and make sure you understand what each command does.

2. Run a molecular dynamics simulation for 200 timesteps at constant energy, and check the logfile `nve.log` which is produced. What information in this log file could you use to establish if your ML potential is working as expected or not?

3. Run a molecular dynamics simulation at with a thermostat for 100 timesteps, and check the logfile `nvt.log` which is produced. You will see that the total energy of the system is not conserved. What real-life experimental conditions are we reproducing which explains this variation in the total energy?

4. The dynamics calculated using the ML potential are approximately 1400 times faster than the traditional MD simulation I performed to obtain the training data. However, that doesn't account for the time it takes to create the model in the first place. What additional timings do we need to know to make a truly fair comparison of the computation costs of the two approaches? When will it be worth training an ML model to perform dynamics?

Visualising the Trajectory

The trajectory produced by running a MD simulation is saved in the file `trajectory.xyz`. This can be visualised using software called VMD. Load this software by running `module load phys/VMD`, then load the trajectory using the command `vmd trajectory.xyz`. This will open two windows - a visualisation, and a control window. For the best visualisation, from the control window choose Graphics ¿ Representations, which will open a new window. In this window find the dropdown menu for Drawing Method, and select VDW. Close this window, and use the play button in the bottom right of the control window to visualise the trajectory.