Accelerating Molecular Dynamics with Machine Learning
Workshop 4, SFB 986 Spring School 2024

Alan M. Lewis

1 Getting Started

This workshop is designed to be completed in groups of 2-4. Try to ensure your group has a mix of people with and without experience in coding, so you can help each other. If someone in the group is unable to run one or both machine learning packages, they are responsible for plotting graphs of the data produced by the other team members. Each section has subsections which should be completed by everybody, and subsections which can be divided amongst the group. These should be clearly indicated.

1.1 Bash

Note: If you have used the bash command line interface before, you can probably skip this section.

We will run the machine learning codes from the command line. If you've not used the command line before, it is simply an alternative way to navigate your computer and run commands. You shouldn't need many commands for the workshop, and all of the commands related to the machine learning programs are explained at the relevant point of this Workshop Guide. Additional commands you might find useful are:

- `cd folder_name` - Change directory into `folder_name`.
- `cd ..` - Go "up" one directory.
- `cd -` - Go back to the previous directory you were in.
- `ls` - List all files and subdirectories in this directory.
- `ls -lh` - List all files and subdirectories in this directory along with detailed information including the size of the files.

1.2 vi

If you have access to a plain text editor (e.g. Notepad on Windows, gedit on Ubuntu), you will be able to use those tools to view and modify text files. If you are restricted to just using the command line, you will need to use a command line text editor called vi to view and modify text files. A cheat sheet covering simple commands is provided here to help you with this if you need it.

- `:w` Save
- `:q!` Quit without saving
- `:x` Save and quit

- `dd` Delete a line
- `V` Select a whole line, using the up and down arrow keys will select lines above or below the current line.
- `y` Copy selected lines
- `x` Cut selected lines
- `p` Paste
- `/findtext` Search for `findtext`
- `:integer` Jump to line number `integer`.
- `:%s/findtext/replacetext/g` Will replace every instance of `findtext` in the file with `replacetext`.
- `u` Undo

2 Learning Energies and Forces

The program `gap_fit` takes some input data stored in an xyz file, and trains a 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 potential is stored in a single file, called `gap.xml`. It is very straightforward to run from the command line: `gap_fit config_file=gap_config.cfg`.

This reads a configuration file, here called `gap_config.cfg` but which could in principle be called anything, which specifies where the program should look for its training data and several other parameters which govern the machine learning model. We have covered most of these parameters in principle in the lecture part of this workshop, 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 order to see if our machine learning model is accurate, we need some way to test its predictions. We do this by pre-calculating the energies and forces for some molecular snapshots, then calculating the energies and forces of the snapshots using our ML potential and calculating the error relative to our precomputed values. This is called validation.

In the `ml-workshop` folder, you will see two python scripts, called `gap_error_validate.py` and `gap_error_train.py`. These perform this validation process on two different sets of snapshots - a validation set contained in `gap_validate.xyz`, which we will never use to train an ML model, and the set of snapshots we used to train the model, which is given in the config file, respectively. These will return as the last line of the output 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.

Questions

1. What is the RMSE in the energy when you use a ML model trained using the default parameters to predict the energies of the validation set?
2. What is the RMSE in the energy when you use a ML model trained using the default parameters to predict the energies of the training set?
3. Why are these values different?
4. What do you think will happen to each error when we train the model using more snapshots?
5. (Optional) Plot `validation_errors.csv` and `train_errors.csv` as a scatter plot. What would these plots look like if the prediction was perfect? Can you tell by eye for which set of snapshots the error is more accurately predicted.

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 improving.

To help you plot a learning curve using `gap_fit`, I have provided a number of input files labelled `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 trainig set by modifying `gap_config.cfg`.

Questions

1. Train a model using 50, 100, 200 and 400 snapshots. 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?
3. Based on this plot, do you think training the model again using more training structures would further reduce the error?
4. (Optional) Train a model using 800 snapshots. This will take a significant amount of time, so it is probably best to set this calculation up to run during the break! Only attempt this if your computer has at least 8 GB RAM.

2.3 Overfitting

This section could be completed by just one or two members of your group.

Overfitting is a common problem when performing machine learning. It describes a situation where your 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 for a simple example in Figure 1.

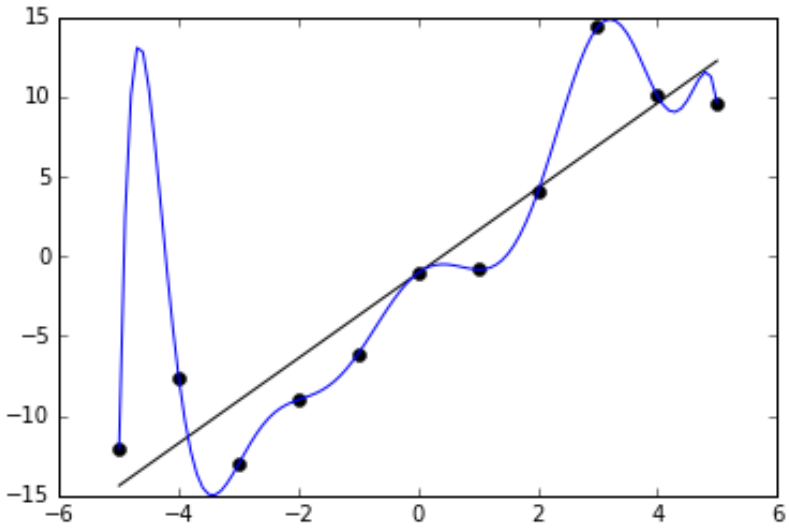

Figure 1: A graph illustrating the concept of overfitting. The blue line goes through all of the known points (training data) very closely, but using this function to interpolate values between the known results is likely to produce very poor predictions. By contrast, the black line does not reproduce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

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 fitting 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.

Questions

1. How could you identify when overfitting is taking place?
2. Using 100 training snapshots, train models using different values of sigma and calculate the RMSE on the training and validation set each time. Do the results match your expectations? What is the optimal value of sigma, in your opinion?
3. Using 100 training snapshots, train models using different values of nmax and calculate the RMSE on the training and validation set each time. Do the results match your expectations? What is the optimal value of sigma, in your opinion.
4. (Optional) Once you have found the optimal value of some paramter(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?
5. What does this process teach you about how ML models are effectively trained?

2.4 Running Molecular Dynamics with an ML Potential

This section could be completed by just one member of your group.

So far we have only considered the energy predicted by the ML model, but the ML 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 are happy is sufficiently accurate, we can use the ML model 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.

Questions

1. Read through through the python script `molecular_dynamics.py` and make sure you understand what each command means.
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. What is the microscopic reason that we see the temperature in the log file fluctuating over the course of this simulation?
4. Run an NVT molecular dynamics simulation 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?

3 Learning Polarizabilities

We can also use the data in the `gap_input_N.xyz` files to train a ML model which can predict the polarizabilities of water snapshots. In these data files, the polarazability associated with each snapshot is listed along with the energies and forces; as with ML potential, the atomic positions and their associated polarizabilities are all we need to train a suitable model.

Unlike the single command for training the ML model for the potential energy, the software to train the model which predicts polarizabilities needs to be run in stages. To make this process simpler, all the necessary commands have collected in a single file, `polarizability.sh`, which is in the `polarizability` folder. You can open this file using a text editor to modify the commands, and then run them all by typing `./polarizability.sh` on the command line while in that folder.

The first two commands in `./polarizability.sh` calculate the descriptors describing each atomic environment present in the input file (`-f $training_data`). It appears twice because we need the descriptors which transform as $L = 0$ (`-lm 0`) and those which transform as $L = 0$ (`-lm 2`). The systems are periodic (`-p`), and we choose to keep the 200 most relevent features of each atomic environment (`-nc 200`). The atomic environments are defined by the cutoff radius (`-rc 4.0`) and the width of the Gaussian centered on each atom (`-sg 0.3`).

The next two commands simply take these descriptors and calculate the kernels which describe the similarity between each pair of atomic environments in the dataset, for both $L = 0$ and $L = 2$. There is no need to modify any of the parameters for these commands.

The final command both trains a ML models based on these descriptors and the polarizability data in the input file, and performs a validation calculation and outputs the results of this validation. In this case, no calculation of the error of the predicted polarazibilities of the training set is performed. The first three options indicate that the property to be learned is the Polarizability (`-p Polarizabilility`), which is a rank 2 tensor (`-r 2`) and symmetric (`-t 1.0`). The next two options indicate the training data for the model, namely the atomic snapshots and their associated polarizabilities (`-f $training_data`) and the kernels which relate the environments in these snapshots to one another (`-k kernel0.npy kernel2.npy`). Then come the regularisation constants, which are designed to prevent overfitting (`-reg 1e-8 1e-5`, one for each kernel). Finally, the input file is split into a set of training and validation snaphots (`-rdm 150`), and the number of training snapshots to use is selected (`-ftr 0.1`; this means that 10% of of selected 150 training snapshots will be used), and the polarizabilities of the validation set are predicted and compared to the true values (`-pr`).

Note that this separation of commands means that you don't need to necessarily rerun every command for every new calculations. For example, if you wanted to test the perfomance of models trained with different numbers of training structures, then after you have calculated the descriptors and kernels for the first time, you will only need to rerun the command `sagpr_train` with different parameters to achieve this. If you want to skip a command, simply place a # at the start of its line in `polarizability.sh` and it will not be executed.

Questions

1. Calculate a learning curve for three components of the polarizability (xx, xy, and xx). Which components have similar learning curves? Can you think of a reason why this might be the case?

2. We have kept only 200 features of each atomic environment in order to save space on our hard disk. Make a note of the size of the `PS0.npy` and `PS2.npy` files. Then recalculate the learning curves from part 1 keeping 500 features of each atomic environment instead. Does this improve the accuracy of the ML models? How much bigger are the descriptor files in this case?

3.

Predicting Polarizabilities of new structures

In Section 2.4 we calculated a short molecular dynamics trajectory using a ML potential we had trained. If we were using traditional quantum chemistry methods, we could have calculated the polarizabilities at the same time. When using ML, we need to use a separate ML model to predict the polarizabilities.