Intro:

For our machine learning assignment, we were provided with three datasets for water molecules, and expected to train a Neural Network potential with a given machine learning library. In this brief overview of my assignment I will go into detail on the model I used, the results I got, and some modifications I made from what was suggested in the notes, along with some more optimizations that could potentially make the model more accurate.

Datasets and processing:

In our hands-on session for this assignment, we were given six .xyz and .ENER files, with the .xyz files corresponding to the positions of the 3 atoms in space, and the .ENER files corresponding to the total electric energy values of each water molecule configuration. Both were split into three parts each, with one .xyz and .ENER file representing the test water molecules, the unrotated water molecules and lastly the rotated water molecules. To extract the data from these files we were provided with two code blocks. The first reads the .xyz files by looping through each molecule, taking its values in cartesions coordinates while ignoring the headers. It then returns each molecule as a 2d numpy array, with each row corresponding to a water molecules geometry. The second reads the .ENER files and converts each energy value to a float, returning it as a numpy array which can be lined up with the corresponding geometry from the .xyz files.

After extracting and reading the six files, and after combining the two sets of training data into one larger dataset, I included a scaler in my code as shown below:

```
scaler = StandardScaler()
scaled_training_xyz = scaler.fit_transform(xyz_combined_training_data)
scaled_test_xyz = scaler.transform(xyz_test)
```

This scaler takes the geometry values of each molecule and makes it more balanced so in theory you have less wildly different values. It is used to speed up training time and avoid the learning process to become unstable or erratic. Although potentially not necessary for these specific datasets (as they are already well scaled), I included it in case I wish to re-use this code in the future for more varying datasets. It also seems to be common practice among any other machine learning models I have looked at.

Model Architecture and Training Setup:

The model used for this neural network potential was created using a Sequential model supported by Keras. This allowed for layers to be stacked in a straightforward linear order. Here specifically the model begins by accepting a 9 value vector, which corresponds to the (x, y, z) positions of the 3 atoms, and then passes this vector through two hidden layers containing 64 neurons each. These two layers used the “relu” or rectified linear unit activation function, which introduces non-linearity and thus allows makes it easier for the model to understand the relationship between atomic position and molecular energy. Then finally we have an output layer with only a single neuron, used to predict the total energy of the molecule.

Before we can train the model, we need to compile it using a loss function, an optimizer and an evaluation metric. The optimizer used is known as “Adam”, and the loss function and the tracking metric were Mean Square Error and the Mean Absolute Error respectively. The specific optimizer is chosen because of its ability to adjust learning rates during training, improving

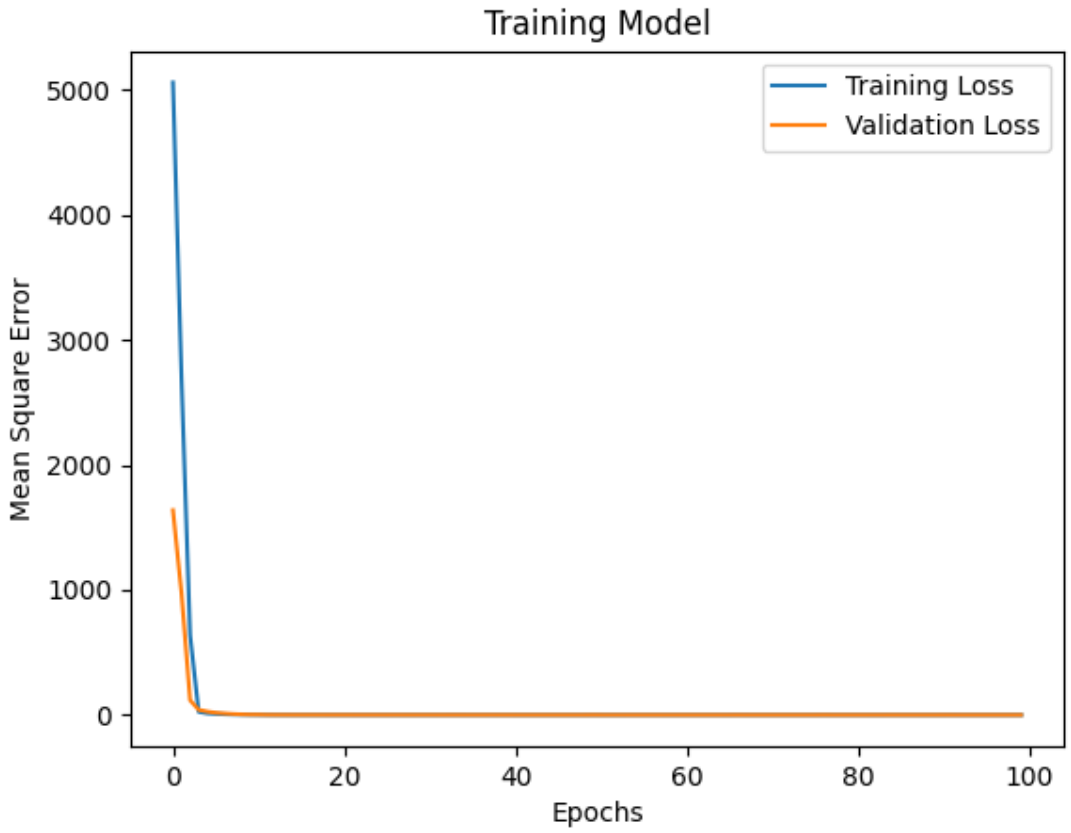
convergence. The Mean Square Error and the Mean Absolute Error are common in regression problems, with the former punishing larger prediction errors and the latter being used to track how far off the model is from the average.

After compiling, the model is trained using the fit function. The input geometry and energy output are fed into the network (in batches of 32), where training runs for 100 epochs. Here I also added in an early stopping callback into this part of the code, where the code is stopped if the validation error fails to improve for 10 consecutive epochs. This is to prevent overfitting.

Results and Discussion:

The final results of the trained neural network returned a score of 0.0310 for the Mean Absolute Error and 0.0019 for the Mean Square Error. This would suggest that the model was able to map the total molecular energy from the atomic coordinates effectively and with a high degree of accuracy.

To visualize the training process, I created a graph as shown below:



This graph shows how the Mean Square Error of both the training and validation losses evolved throughout the epochs. It shows that for training and validation alike, the loss bottomed out almost immediately, with both plateauing close to zero within the first 10 epochs. This suggests firstly that the model learned fast and converged quickly in the training process. Secondly, we have the close alignment of the validation and training loss curves which would imply that the model generalizes well and doesn't overfit the data.

Some modifications that could've been made to the model include changing the number of neurons used when building the sequential model. Other values, such as 16, 32 or 128 were all tested separately to potentially revealing underfitting or overfitting behaviour. Due to the relative

simplicity of the model however they all performed reasonably well. Alternatively, if we wished to see how the model dealt with orientation-based variability, the model could have been run on rotated vs unrotated sets independently.

Conclusion:

Overall, this assignment showed the successful development of a neural network potential which predicted the total electronic energy of a water molecule using its atomic coordinates. The training process with the graph showing a flattening out of both the training and validation loss, and no signs of overfitting. While the current setup worked well for this particular dataset, there are a number of modifications that could be made in the future if this model were applied to a more complex problem. Overall however this was a very solid starting point for using machine learning to predict water molecule properties.