Multipole Moments of Water using Machine Learning

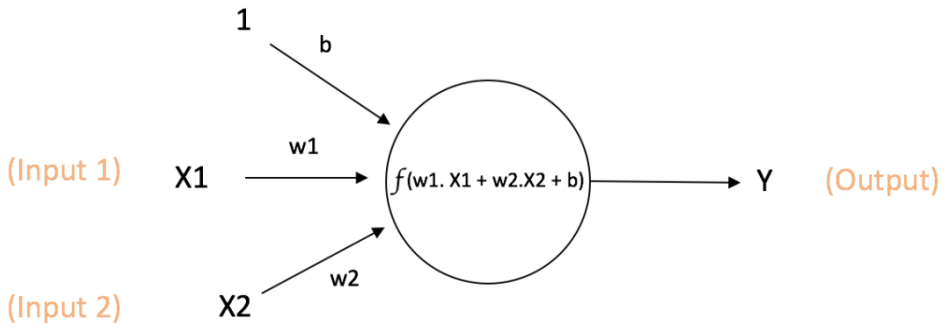
Mentor: Prof. Vishal Agarwal, Department of Chemical Engineering, IIT Kanpur
By: Lakshay Tyagi, Third Year Undergraduate, IIT Kanpur

Introduction:

Water is the harbinger of life and its study is of utmost importance to Chemical Engineering and Engineering in general. As expected given its importance water is a subject of extensive research.
In spite of the abundance of models for water, the need for accuracy and realism remains.
It is expected that water models will become more accurate if polarisation is explicitly represented in the model. Also, it has been shown that flexibility (i.e. non-rigidity of intramolecular bonds) should only be introduced once polarisation can be properly modelled. There are now a large number of chemical potentials that make use of machine learning methods. These methods allow the functions describing the potential to emerge from the data, ideally coping with a high number of independent variables. A commonly used method is a type of artificial neural network called multilayer perceptron (MLP).[1]
In this project, I have attempted to implement the paper "Optimal construction of a fast and accurate polarisable water potential based on multipole moments trained by machine learning" by Chris M. Handley, Glenn I. Hawe, Douglas B. Kellab and Paul L. A. Popelier which is an attempt to model the multipole moment and hence the Potential Energy Surface of the water molecule.

Neural Networks[2]:

A Neural Network (NN) is a computational model that is inspired by the way biological neural networks in the human brain process information. Neural Networks have generated a lot of excitement in Machine Learning research, thanks to many breakthrough results in speech recognition, computer vision and text processing.
The basic unit of computation in a neural network is the neuron. It receives input from some other neurons and computes an output. Each input has an associated weight (w), which is assigned on the basis of its relative importance to other inputs. The node applies a function f (defined below) to the weighted sum of its inputs as shown in the figure below:

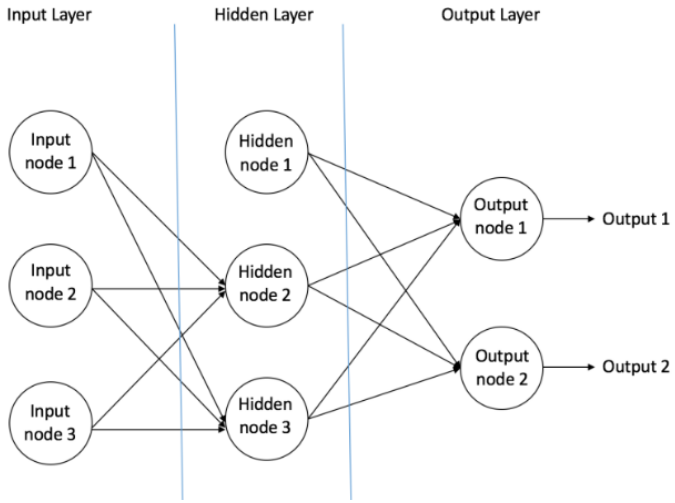


ref-2

The above network takes inputs X1 and X2 and has weights w1 and w2 associated with those inputs. The function f is non-linear and is called the Activation Function. The purpose of the activation function is to introduce non-linearity into the output of a neuron.
Some examples of activation functions are:

- Sigmoid function: Takes real-valued input and outputs a real number between 0 and 1.
  $\sigma(x) = 1 / (1 + \exp(-x))$
- Tanh function: Takes real-valued input and outputs a real number between -1 and 1.
  $\tanh(x) = 2\sigma(2x) - 1$
- ReLu function: Takes real values Input and thresholds it to zero.
  $f(x) = \max(0, x)$

In this implementation sigmoid activation has been used for all networks trained.
The general schematic of a neural network is as below



ref-2

Here each circle denotes a neuron with input from neurons of the previous layer and it's output feeding inputs to the next layer.
The error of prediction or Loss as it is often called is a measure of the performance of the model.
For different problems there exist different loss functions to measure the accuracy of the network.
For regression problems of the sort, we will be dealing with the loss function is taken as the Mean Squared Error(MSE). Below is the formula for the MSE.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

ref-1

Here Y-hat denotes the Output from the data and Y denotes the prediction or output of the network with n being the number of examples MSE is being calculated on.
Learning can be thought of as fine-tuning the weights to minimise the Loss over the train data set.
The process by which a NN learns is called the Backpropagation algorithm. Initially, all weights are randomly assigned to some small real numbers and are then adjusted according to the error.
In general, the backpropagation update for any weight of a neural network can be written as:

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i} \quad \text{for } i = 1, \ldots, \ell,$$

ref-1

Here wi is the weight being updated and E is the error or loss of the prediction.

Data Generation:

To avoid redundancy in Input parameters, configurations were described by using Molecular Local Frame which means fixing one of the water molecules at the origin along with its orientation and then describing the positions of the rest of the water molecules with respect to that molecule. Six inputs are required now to fix the orientation of any other molecule these being r, theta and phi to decide the

co-ordinates of the central oxygen atom and then three Euler angles to decide it's orientation. Therefore, to describe the configuration of n water molecules in the MLP we need 6*(N-1) parameters.
Algorithm for generating data:

1. Generate a random number between 2 and 8 to get the distance of central oxygen atom
2. Generate random numbers between 0 and pi for theta and 0 and 2*pi for phi to decide its position in spherical coordinates.
3. Then generate three random numbers between 0 and 2*pi to get the 3 Euler angles to decide the orientation of the water molecule
4. Then convert these parameters to Cartesian Coordinates of each atom and feed this to Gaussian to get the Multipole Moments

Link for code-https://github.com/lakshayt7/Water-PES-using-ML/blob/master/traindatarunner.py

Details of Model:

First pre-treatment is done to remove test cases in which Gaussian returned an error because of very small interatomic distances.

All inputs and outputs are normalised using the following scheme:

$$x_{i,n} = \frac{(x_i - x_{\min})}{(x_{\max} - x_{\min})}$$ ref-1

Here xmin and xmax denote the minimum and maximum for that specific input or output being normalised.
Data is split into train and test data randomly with 80% going to train and 20% going to test set
Neural Network with one or two hidden layers is used for fitting train data and accuracy is measured on Test set.
MSE on test set divided by Variance along with average error is used as measures for performance

$$\left[\frac{\sum_{j=1}^{N}(a_j - b_j)^2}{\sum_{j=1}^{N}\left(a_j - \left(\frac{1}{N}\sum_{j=1}^{N}a_j\right)\right)^2}\right] \qquad \left(\frac{1}{n}\sum\frac{|Actual - Forecast|}{|Actual|}\right)*100$$ ref-1

Link to code - https://github.com/lakshayt7/Water-PES-using-ML/blob/master/dimer_MLP.py

Implementation Details:

- Tensorflow has been used for the implementation of the neural networks.
- Optimiser used for training is AdamOptimiser from tensorflow.train
- All Weights are initialised with Standard Deviation of 0.05
- L2 regularisation has been used to prevent overfitting
- Basis set used for Gaussian calculations is B3LYP-aug-cc-pVTZ

Results:

Below are the results for various multipole moments from Dipole to Hexadecapole for the initial training with the initial data set of 2000 points

Multipole Moment	MSE	Percentage Error
Dipole-X	0.79	13.2
Dipole-Y	0.70	12.1
Dipole-Z	0.63	11.7
Quadrupole-XX	0.19	5.4
Quadrupole-YY	0.23	5.9
Quadrupole-ZZ	0.41	6.1
Quadrupole-XY	0.71	12.8
Quadrupole-XZ	0.83	14.2
Quadrupole-YZ	0.46	6.7
Octapole XXX	0.68	6.1
Octapole YYY	0.58	3.8
Octapole ZZZ	0.52	3.3
Octapole XYY	0.89	10.2
Octapole XXY	0.72	9.4
Octapole XXZ	0.53	7.6
Octapole XZZ	0.83	12.7
Octapole YZZ	0.78	7.6
Octapole YYZ	0.78	7.3
Octapole XYZ	0.92	15.1

Hexadeca XXXX	0.06	2.0
Hexadeca YYYY	0.13	2.7
Hexadeca ZZZZ	0.15	1.9
Hexadeca XXXY	0.76	9.8
Hexadeca XXXZ	0.72	9.7
Hexadeca YYYX	0.79	11.4
Hexadeca YYYZ	0.82	12.5
Hexadeca ZZZX	0.91	14.5
Hexadeca ZZZY	0.75	9.8
Hexadeca XXYY	0.03	2.3
Hexadeca XXZZ	0.01	1.9
Hexadeca YYZZ	0.15	3.1
Hexadeca XXYZ	0.89	14.1
Hexadeca YYXZ	0.84	13
Hexadeca ZZXY	0.81	11.7

These are the results along with comparison when the data set was expanded to 4000 points.

Multipole Moment	**MSE-I**	**MSE-II**
Dipole-X	0.79	0.78
Dipole-Y	0.70	0.64
Dipole-Z	0.63	0.47
Quadrupole-XX	0.19	0.19
Quadrupole-YY	0.23	0.20
Quadrupole-ZZ	0.41	0.32
Quadrupole-XY	0.71	0.63
Quadrupole-XZ	0.83	0.78

Quadrupole-YZ	0.46	0.40

Multipole Moment	MSE-I	MSE-II
Octapole XXX	0.68	0.67
Octapole YYY	0.58	0.45
Octapole ZZZ	0.52	0.49
Octapole XYY	0.89	0.87
Octapole XXY	0.72	0.72
Octapole XXZ	0.53	0.47
Octapole XZZ	0.83	0.82
Octapole YZZ	0.78	0.76
Octapole YYZ	0.78	0.77
Octapole XYZ	0.92	0.88

Hexadeca XXXX	0.06	0.05
Hexadeca YYYY	0.13	0.11
Hexadeca ZZZZ	0.15	0.14
Hexadeca XXXY	0.76	0.75
Hexadeca XXXZ	0.72	0.72
Hexadeca YYYX	0.79	0.71
Hexadeca YYYZ	0.82	0.80
Hexadeca ZZZX	0.91	0.89
Hexadeca ZZZY	0.75	0.73
Hexadeca XXYY	0.03	0.02
Hexadeca XXZZ	0.01	0.01
Hexadeca YYZZ	0.15	0.12
Hexadeca XXYZ	0.89	0.88

Hexadeca YYXZ	0.84	0.81
Hexadeca ZZXY	0.81	0.79

Conclusions:

Accuracy is Decent but not high enough on the multipole moments to get accurate energy predictions for the Potential Energy Surface.
Possible Ways to increase Accuracy:

1. Increase the size of the data set used for training as the current amount of data might not be enough for the model to learn complex non-linear functions.
2. Use Alternate Model as the current Multi-Layered Perceptron may be too simple to describe complicated functions.
3. Use more accurate calculations, the current model is fitted on calculations done by using B3LYP basis set which is not very accurate for water molecules

Out of these possible remedies, Option 1 was implemented and the change in MSE when increasing the size of data set from 2000 to almost 4000 points was calculated. These changes are shown above in the results. This did increase the accuracy slightly but the improvements are not worth the tradeoff for doubling the time taken for training as the initial 200 iterations in 1 second went down to 100 iterations in 1 second.
For Option 2 using Radial Basis Function Neural Network (RBFNN) is a good alternative. I have been working side by side for implementing this network. However, some ambiguity about how to choose some hyper-parameters makes this difficult.
As for Option 3 More accurate calculations mean more time required for Data generation. This will push up the current time of 1000 configurations per 3 hours. However, this still is a feasible approach given that we can increase the number of configurations by using data augmentation.

Future Possibilities:

I will be working on implementing the radial basis function network to see if the current results can be improved using it.
Also, I am looking forward to implementing neural networks for the task of direct Potential Energy Regression which is given as a reference below[3].

References:

Ref-1: Images and text were taken from:
“Optimal construction of a fast and accurate polarisable water potential based on multipole moments trained by machine learning” by Chris M. Handley, Glenn I. Hawe, Douglas B. Kellab and Paul L. A. Popelier
Ref-2: Images and text were taken from:
https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/
Ref-3: “A neural network potential-energy surface for the water dimer based on environment-dependent atomic energies and charges” by Tobias Morawietz, Vikas Sharma, and Jörg Behler