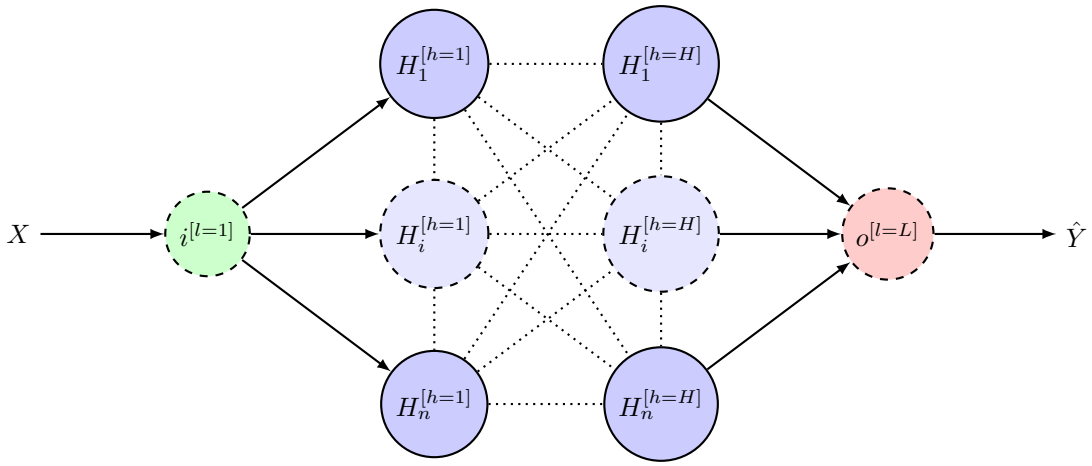
Crystal Structure Feature Prediction using a Multi-layer Perceptron

Overview

The computational expense of crystal structure prediction via ab initio quantum methods poses a significant obstacle and remains a persistent problem in computational chemistry.

Information obtained from the Crystallography Open Database (COD) will be used as a training and testing resource. If successful my model will predict crystal properties for certain chemical species with higher accuracy than a random guess constrained within acceptable boundaries. For example one paper; 'CRYSPNet: Crystal structure predictions via neural networks',[1] had strong success with bravais lattice prediction, with the metal model reaching an accuracy of 70%.

Depicted below is a rough structure for the initial model considered:



Mathematical Process

A gradient descent method will be examined, whereby an input is passed through layers of neurons which result in a calculated loss function which is then minimised by differentiation in a backpropagation process. Parameters are subsequently updated based on learned values.

The forward pass using input X:

$$Z_1 = XW_1 + b_1 \tag{1}$$

Input matrix, X, is multiplied by an initially random weights matrix, W_1 with the addition of a bias vector, b_1 (both parameters randomised within constraints for the sake of efficiency).

The resultant value, Z_1, is then activated via an activation function, for example the Sigmoid function, to give activated matrix A_1:

$$A_1 = \frac{1}{1 + e^{-Z_1}} \implies A_1 = \sigma(Z_1) \tag{2}$$

This process is vital as without an activation function, passing any matrix through the layers would result in a linear transformation.

With reference to the diagram, A_1 would be the output of the initial layer i. This A_1 matrix is then used as an input to the first of the hidden layers ($h = 1, l = 2$):

$$Z_2 = A_1 W_2 + b_2 \tag{3}$$

Z_2 is then activated via the same sigmoid function and passed on to $h = 2, l = 3$. This is continued sequentially for a given number of hidden layers each with a given number of neurons.

An overall relationship can be described in the following way:

$$Z_l = A_{l-1} W_l + b_l \tag{4}$$

When $l = L$ we reach the output layer and a final output is produced, Z_L, which may or may not be activated again dependent on design. In the case of our model this represents $\hat{Y}$.

The number neurons at any given layer is determined by the shape of the matrix considered. An input matrix of shape n, m for example would yield m neurons. Each layer of matrices can effectively take any shape constrained by the need for matrix compatibility, allowing us to choose a variable number of neurons for our hidden layers.

Finally, we calculate a loss function such as Mean Squared Error (MSE). Using MSE as a an example:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{Y}_i \right)^2 \tag{5}$$

- n: The number of samples.
- y_i: The i-th true value.
- $\hat{Y}_i$: The i-th predicted value.

This function lends information on the accuracy of the current "prediction", but more importantly serves as the function we differentiate in order to perform a simple gradient descent optimisation as part of the backwards pass.

The backward pass considering output $\hat{Y}$:

The key things we seek to do are calculate the gradient of the loss, $\mathcal{L}$, with respect to W_l and b_l.

First we define an error term which measures the contribution of Z_l to the loss:

$$\delta_l = \frac{\partial \mathcal{L}}{\partial Z_l} = \frac{\partial \mathcal{L}}{\partial A_l} \cdot \frac{\partial A_l}{\partial Z_l} = \frac{\partial \mathcal{L}}{\partial Z_{l+1}} \cdot \frac{\partial Z_{l+1}}{\partial A_l} \cdot \frac{\partial A_l}{\partial Z_l} = \delta_{l+1} \cdot (W_{l+1})^T \cdot \sigma'(Z_l) \tag{6}$$

Now we can calculate:

$$\frac{\partial \mathcal{L}}{\partial W_l} = \frac{\partial \mathcal{L}}{\partial Z_l} \cdot \frac{\partial Z_l}{\partial W_l} = \delta_l \cdot (A_{l-1})^T \tag{7}$$

$$\frac{\partial \mathcal{L}}{\partial b_l} = \frac{\partial \mathcal{L}}{\partial Z_l} \cdot \frac{\partial Z_l}{\partial b_l} = \delta_l \tag{8}$$

Transformation is necessary in equations 6 and 7, to ensure matrix compatability on the backwards pass. Without this we would see an invalid matrix operation. In this case we are also assuming there is only one 'batch'.

Updating Parameters:

In order to increase the accuracy of the model we must change parameters W_l and b_l. This is done via assigning a learning rate, η. A linear combination of the old parameters with the product of the learning rate and the loss derivative is computed to provide new parameters for the model.

$$W_{new} = W_{old} - \eta \frac{\partial \mathcal{L}}{\partial W_l} \tag{9}$$

$$b_{new} = b_{old} - \eta \frac{\partial \mathcal{L}}{\partial b_l} \tag{10}$$

This process is performed recursively for each layer.

Once parameters are updated the input matrix is fed back through the network to generate another "prediction". Each instance of the forward and backwards cycle is conventionally referred to as an epoch.

Evaluation Metrics

In order to investigate the efficacy of the produced model, we must establish some evaluation metrics. The loss function, equation 5, considers MSE, which is especially useful when trying to assess the impact of more extreme feature values. In a more standardised case where features all have a similar impact on the error we might consider a mean absolute error metric.

$$\mathcal{L}_{\text{MAE}} = \frac{1}{n} \sum_{i=1}^{n} \left| y_i - \hat{Y}_i \right| \tag{11}$$

In a the case of crystal structure prediction, raw data processed in terms of α, β and γ, will have a much more significant impact on the overall prediction comparative to lattice parameters a, b and c, due to their relative magnitude and the fact that at present there is no data pre–processing considered within the model. Data pre–processing will be an important step forward to improve model accuracy.

In the case of a categorical prediction we might construct a confusion matrix to represent the number of true positives and negatives, to compare them with the number of false positives and negatives. A more quantitative success measure can then be calculated by considering precision and recall:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{12}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{13}$$

Results and Discussion

Note: This is an ongoing project and as such might be subject to change.

For our model, a ReLU function was employed for the hidden layers and a softmax function for the final output layer. A cross–entropy loss function was used due to the nature of multi–class classification. Performance over a small sample size of relatively simple structures and a limited number of space groups

indicated a high accuracy of anywhere between 60-80%, but this decreased and the model began to show worse outcomes for a larger sample size and more complicated crystal systems indicating model weakness or improper architecture. Future investigation will surround the use of the tanh function, variations in architecture and changes in learning rate as well as comparison with standard models such as those built from PyTorch and TensorFlow, to better discern model limitations and where improvements can be made. Additional methods of data pre–processing may also be explored, so that a more robust handling of feature descriptors can be employed.

References

1. H. Liang, V. Stanev, A. G. Kusne, *Physical Review Materials*, 2020, **4**, 123802