COMP30230 Connectionist Computing

Programming Assignment Report

Zhi Zhang

Student ID: 18210054

UCD School of Computer Science
University College Dublin

December 5, 2022

Table of Contents

Chapter 1: Assignment Description

This project is to build a multi-layer perceptron MLP, but without using any of the available libraries for neural networks/connectors/machine learning etc. It is written in a Python Jupyter notebook because it is concise, convenient to annotate, and a good record of the training process.

It contain following files:

- **Assignment_Code_Zhi_Zhang.ipynb**

  Contains all the code to build the MLP and test and visualise analysis.

- **Assignment_Code_Zhi_Zhang.html**

  A HTML version of ipynb file.

- **Assignment_Report_Zhi_Zhang.pdf**

  A report with the description of the experiments I have conducted and the results obtained.

- **XOR_log.txt**

  Contain error and results during training and test for XOR question.

- **SIN_log.txt**

  Contain error and results during training and test for SIN question.

- **Letter_recognition_log.txt**

  Contain error and results during training and test for letter recognition question.

- **Rank_log.txt**

  Log data on how the accuracy of letter recognition changes when hidden layers and epochs are changed.

- **letter-recognition.data**

  Data set for MLP training on letter recognition.

Chapter 2: Multi-Layer Perceptron

- **Initialisation**

  Define all required variants, invariants. For example, number of input (NI), hidden layers (NH), outputs (NO) and weights. In addition, choose appropriate activation functions between sigmoid function and hyperbolic tangent function.

- **Randomise**

  Deviation weights for the input and hidden layers were considered. Initialises W1 and W2 to small random values and set both dW1 and dW2 to 0.

- **Forward propagation**

  Forward propagation of the input to the training pattern through the neural network to produce the propagated output activation.

  Calculate following variables: An array containing the activations for the lower layer (Z1). An array where the values of the hidden neurons are stored (H). An array containing the activations for the upper layer (Z2) and the outputs (O).

- **Backward propagation**

  Using the training pattern objective, back-propagation of the propagated output activations is performed through the neural network to produce delta of all outputs and hidden neurons. Calculate the deltas for the upper and lower layers, then calculate the change in weights (dW1 & dW2) and calculate the error of the current input.

- **Weight update**

  Update the weights (W1 and W2) by multiply dW with learning rate. After that, reset dW1 and dW2 to 0 again.

- **Train**

  A convenient training function, by calling the above functions in order.

- **Predict**

  It is similar to the training function, but it is used to predict.

Chapter 3: Test Results

3.1 XOR

`NeuralNetwork (NI = 2, NH = 10, NO = 1, activation = "tanh")`
`with learning_rate=0.05 and epochs=10000 for training.`

Input		Expect Output	Actual Output	Errors (avg)
0	0	0	0	0.0627
0	1	1	0.98893	
1	0	1	0.98865	
1	1	0	0.00841	

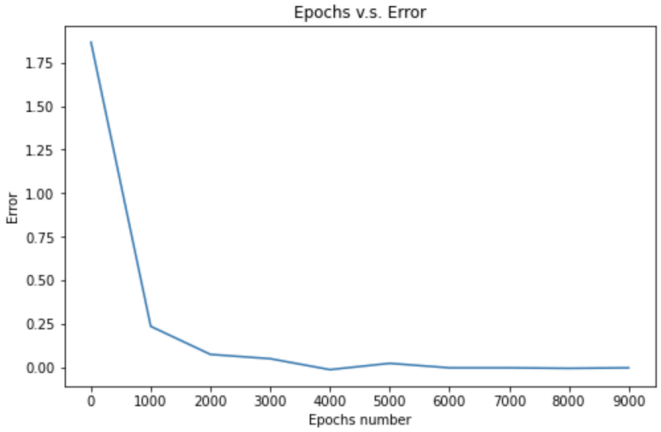

Figure 3.1: XOR Error

As we can see from Figure 3.1, the test results are very satisfactory and the error rate is gradually decreasing along with the successful results.

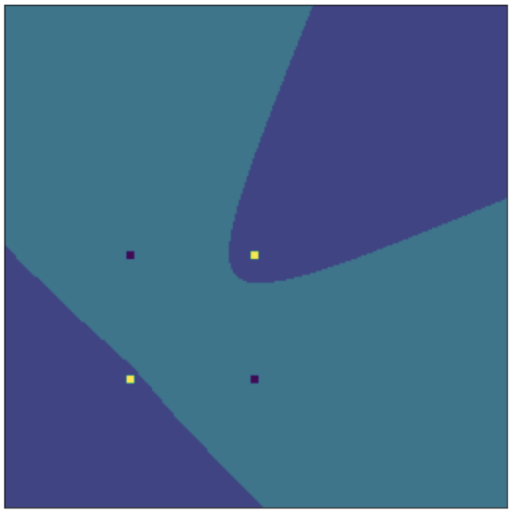

Figure 3.2: XOR result

A visualisation of the XOR is shown above, which can be found to successfully solve the problem.

3.2 SIN

```
NeuralNetwork(NI = 4, NH = 10, NO = 1, activation = "tanh")
with learning_rate=0.05 and epochs=100000 for training.
```

This exercise is to calculate $sin(x1 - x2 + x3 - x4)$, the tanh function being used. Since the sin function takes four randomly valued inputs in the range -1 to 1, they are then combined and computed to produce a vector. So we first create a matrix of size $500 * 4$, and then for each set of 4 inputs, compute $sin(x1 - x2 + x3 - x4)$ to generate an output set. We then use 400 of these to train the system and use the remaining 100 to test.

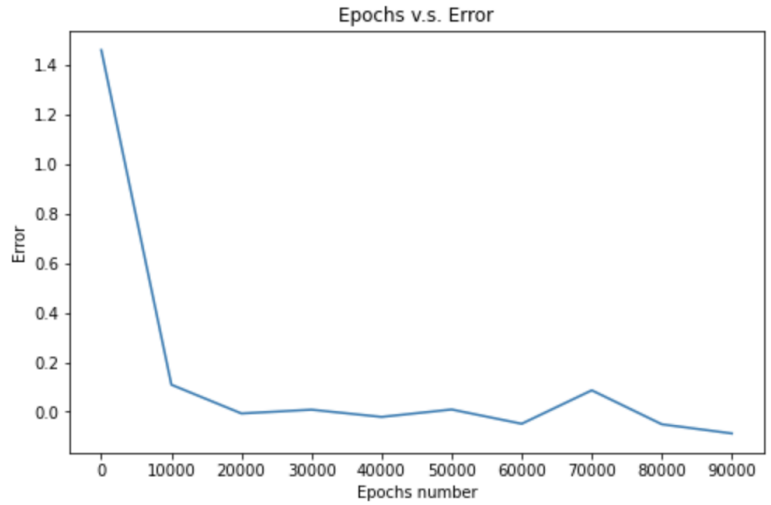

Figure 3.3: SIN Error

As can be seen from Figure 3.3, although errors can fluctuate, they eventually trend downwards and converge infinitely to zero.

```
************* Training *************

The RMSE of "training set" is 0.08  Accuracy = 0.94

*************** Test ***************

The RMSE of "test set" is 0.1  Accuracy = 0.92
```

From the output of the training and test log files, we can see that after 100,000 training sessions, our model can perform very well, with an accuracy of 92% on the test set and an RMSE value of only 0.1.

3.3 Letter recognition

This exercise was performed to train the MLP on the UCI letter recognition dataset. similarly, the dataset was also split into approximately 4/5 of the training part and 1/5 of the testing part. We start by pre-processing the data. Each item in this dataset consists of 16 attributes describing the letters, which means that our model has 16 inputs.

In addition, because the dataset is predicting letters, the number of outputs is 26. In addition, we need to convert 26 letters into an array with numbers from 0-25 by one-hot encoding. For example, letter D is 4th letter and it will converted into an array with 1 in the 4th position and 0 in the other positions, with a length of 26.

Furthermore, in order to construct a model with the best performance, we investigated the effect of parameter variation on the accuracy of the MLP. The results are shown in the figures below.

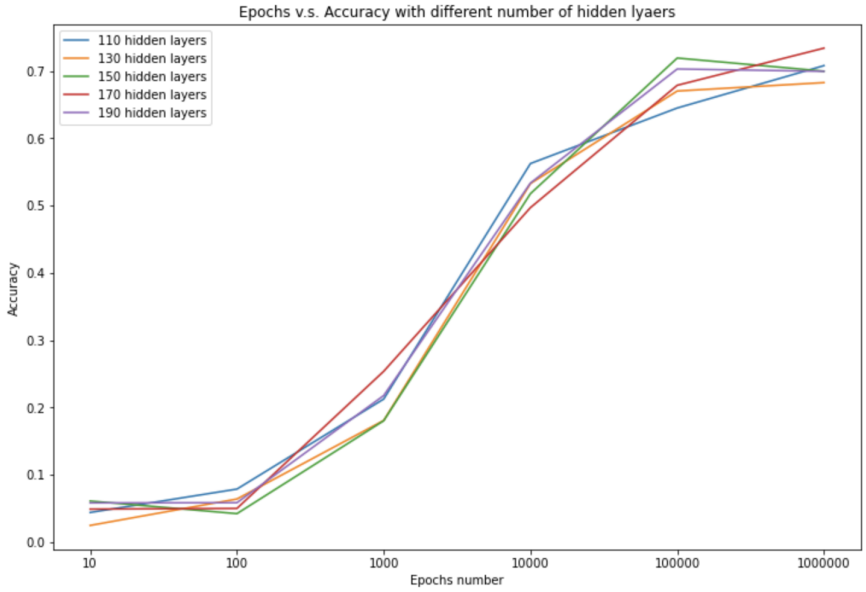

Figure 3.4: Epochs v.s. Accuracy with different number of hidden lyaers

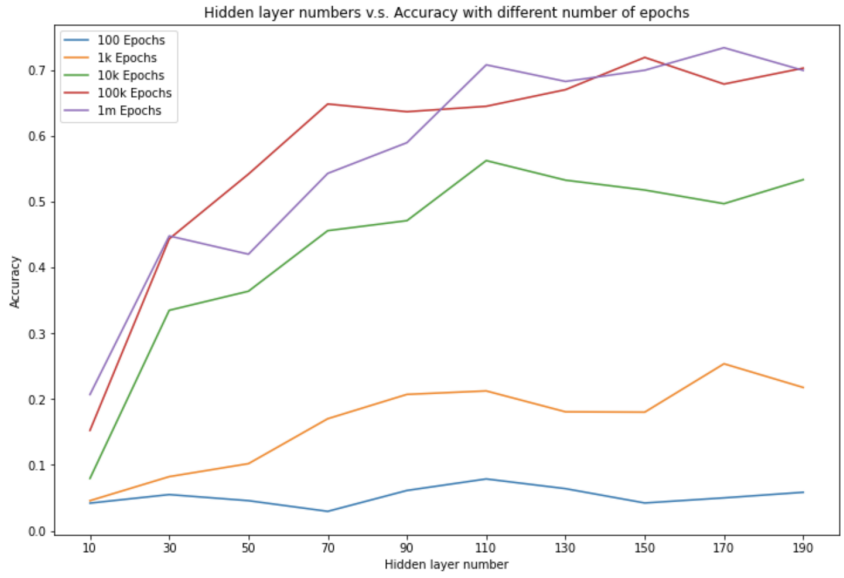

Figure 3.5: Hidden layer numbers v.s. Accuracy with different number of epochs

We found that as the number of epochs sessions increased, their accuracy rose, regardless of the number of hidden layers. But among them we identified the 170 hidden layers as the most effective.

Furthermore, it was found that the best accuracy was achieved when trained 1 million times at 170 hidden layers shown in Figure 3.5, therefore, we decide use following parameters as configuration.

```
NeuralNetwork (NI = 16, NH = 170, NO = 26, activation = "sigmoid")
with learning_rate=0.05 and epochs=1000000 for training.

...

The accuracy of "training set" is 0.72425
The accuracy of "test set" is 0.72775
```

To be honest, this test took a long time to perform, but it did in fact confirm my claim that the accuracy of the predicted letters did improve. Its accuracy in predicting each letter is shown in the graph below.

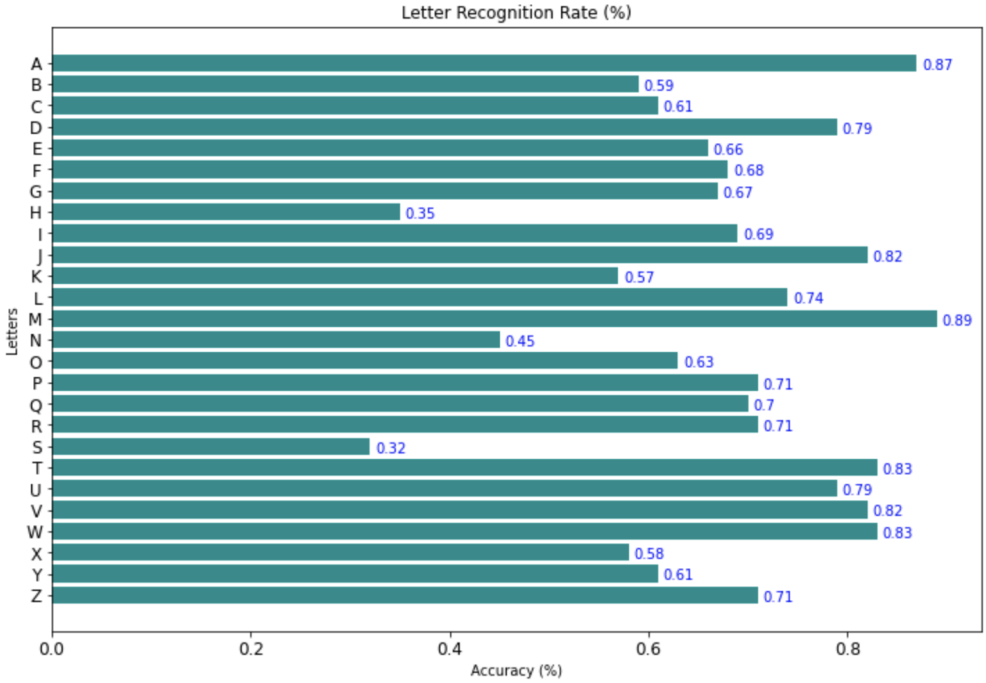

Figure 3.6: Letter recognition performance

We can see that this model is strongest in recognising the letter 'M', followed by the letters 'A', 'T' and 'W'. However, the recognition of the letters 'S' and 'H' is not as good as it should be.

Chapter 4: **Conclusion**

In general, I am quite satisfied with my results, they are better than I expected as it scored well in all the tests. I learnt in more depth about writing neural networks in practice and realised how important it is to choose a suitable activation function.

I found that more hidden units allowed the neural network to converge quickly in the early stages, but the training results were not necessarily better than a neural network with fewer hidden units. If the target will be negative one needs to use 'tanh' as the activation function. If it is all positive, then 'sigmoid' is used as the activation function.

In addition, I recognised in these exercises that increasing the number of epochs was helpful in improving the accuracy of the predictions, which seemed intuitive as it gave the model more time to learn and reduce errors. However, it does significantly affect the execution time, so it may not be the most optimal approach. So I also explored the effect of different parameters on the accuracy of the model. While accuracy does depend on factors such as the number of hidden units/layers, the number of epochs, etc., it also depends on other factors such as the quality of the data the model is trained on and the quality of the model itself.