Multilayer Perceptron

In my code I have built a three-layer Neural network. An input layer a hidden unit layer and the output layer. This report will delve into the MLP code provided. It will examine the results of the experiments carried out on how the model trained and the accuracy of its predictions on different data sets through the manipulation of hyperparameters namely the learning rate and the number of learning cycles(epochs). The learning rate indicates how much the weights are updated during training. Through the experiments the aim is to optimise each of these hyperparameters for the training of the various datasets.

Finding a correct learning rate is important in a neural network as a larger learning rate will allow the model to learn faster ideal if you have a lot of data but it may deliver final weights that are suboptimal. A smaller learning rate can allow the model to train more optimally but at an increase in the length of time to train it.

Q1. Train an MLP with 2 inputs, 3-4+ hidden units and one output on the following examples (XOR function).

Q2.

Predictions based upon different learning rates, epochs and number of hidden layers.

Weights were initialised to small random values using NumPy for each experiment.

w1 = np.random.rand(NH,NI)   # Weight matrix for hidden layers

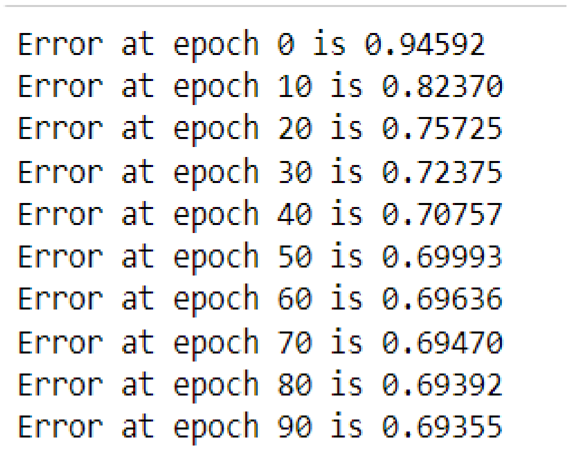
w2 = np.random.rand(NO,NH)   # Weight matrix for output layer

Experiment 1

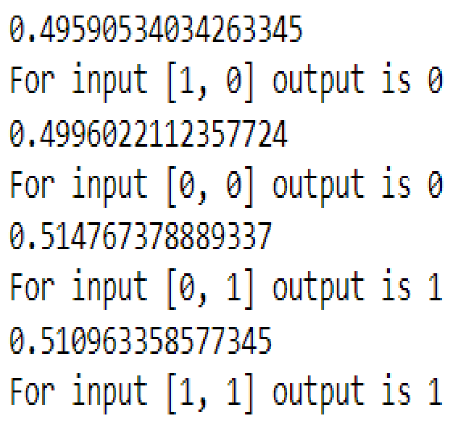
Learning rate = 0.1

Number of hidden units = 4

Epochs =100 (initial setting)



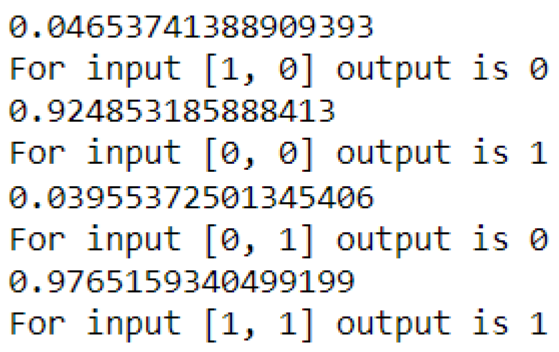
```
Error at epoch 0 is 0.94592
Error at epoch 10 is 0.82370
Error at epoch 20 is 0.75725
Error at epoch 30 is 0.72375
Error at epoch 40 is 0.70757
Error at epoch 50 is 0.69993
Error at epoch 60 is 0.69636
Error at epoch 70 is 0.69470
Error at epoch 80 is 0.69392
Error at epoch 90 is 0.69355
```



```
0.495905340342633345
For input [1, 0] output is 0
0.4996022112357724
For input [0, 0] output is 0
0.514767378889337
For input [0, 1] output is 1
0.510963358577345
For input [1, 1] output is 1
```

Learning for these parameters was not very good as it had only a 50% chance of predicting the correct output. Next, I increased the epochs (incrementally by 1000 epochs at a time) but it took training of around 10,000 epochs to give me the best predictions. This indicated to me that the learning rate was probably too small which is why it took so long to learn.

At the end of this experiment, I had excellent predictions but a model that was very slow to learn.



```
0.04653741388909393
For input [1, 0] output is 0
0.924853185888413
For input [0, 0] output is 1
0.03955372501345406
For input [0, 1] output is 0
0.9765159340499199
For input [1, 1] output is 1
```

Experiment 2

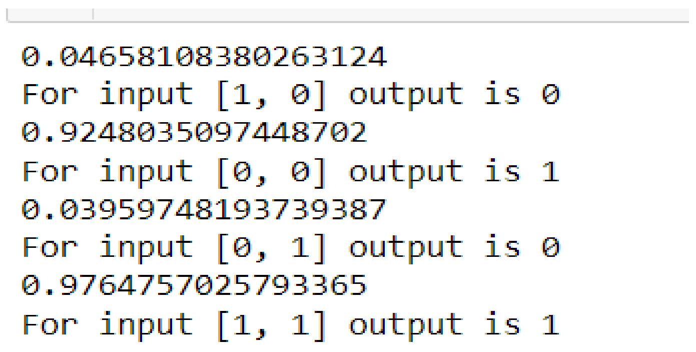
From my first experiment I could see that the learning rate was too small so I doubled it and set the epochs to 5000 as my initial values.

Learning rate = 0.2

Number of hidden units = 4

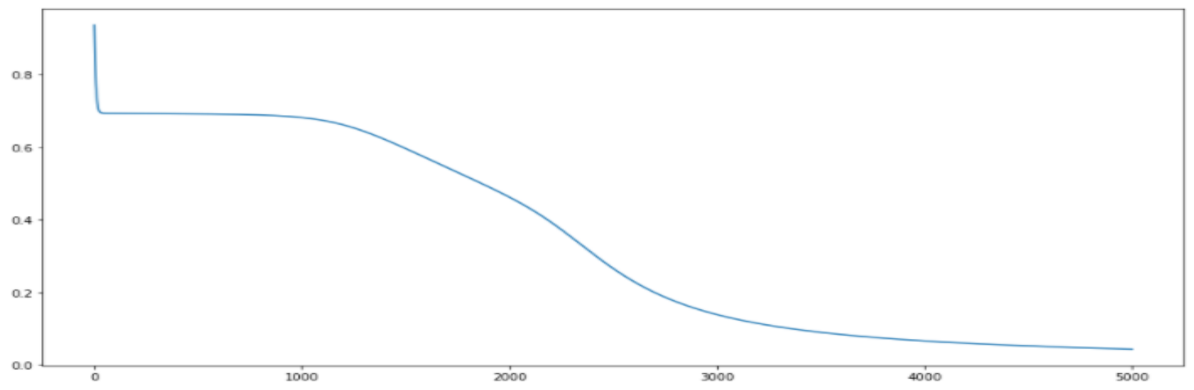
Epochs = 5000

This yielded very similar results but in half the epochs of experiment one making still just as accurate.

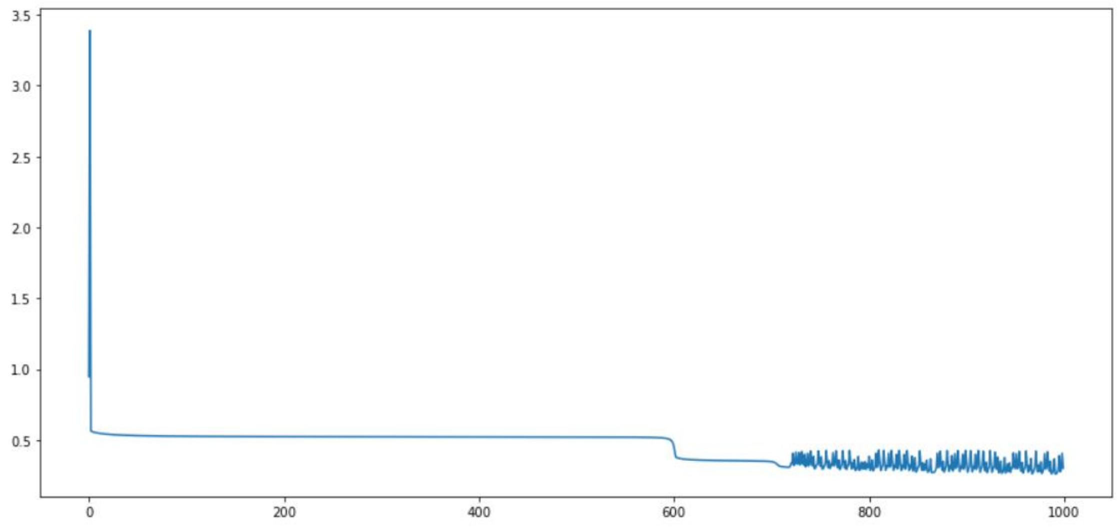


```
0.04658108380263124
For input [1, 0] output is 0
0.9248035097448702
For input [0, 0] output is 1
0.03959748193739387
For input [0, 1] output is 0
0.9764757025793365
For input [1, 1] output is 1
```

Learning rate graph



To see the effect of a to large learning rate in a graph format I gave the learning rate a value of 20 which is not the norm. The learning rate is usually between 0-1. But it allows us to see the oscillating effect caused by the weights being updated by too large a number and training error increasing and decreasing. Ideally the training error should be decreasing over training epochs like the above graph. To small a learning weight and the data will be underfitted. To large a learning rate and the data will be overfitted.



Q3

The second example dataset posed a challenge as it was very difficult to get the correct array sizes so that you could perform operations on them. Once the matrices were the correct shape the next issue was trying to optimise the hyperparameters. The learning rate was significantly smaller than the learning rate for the XOR problem.

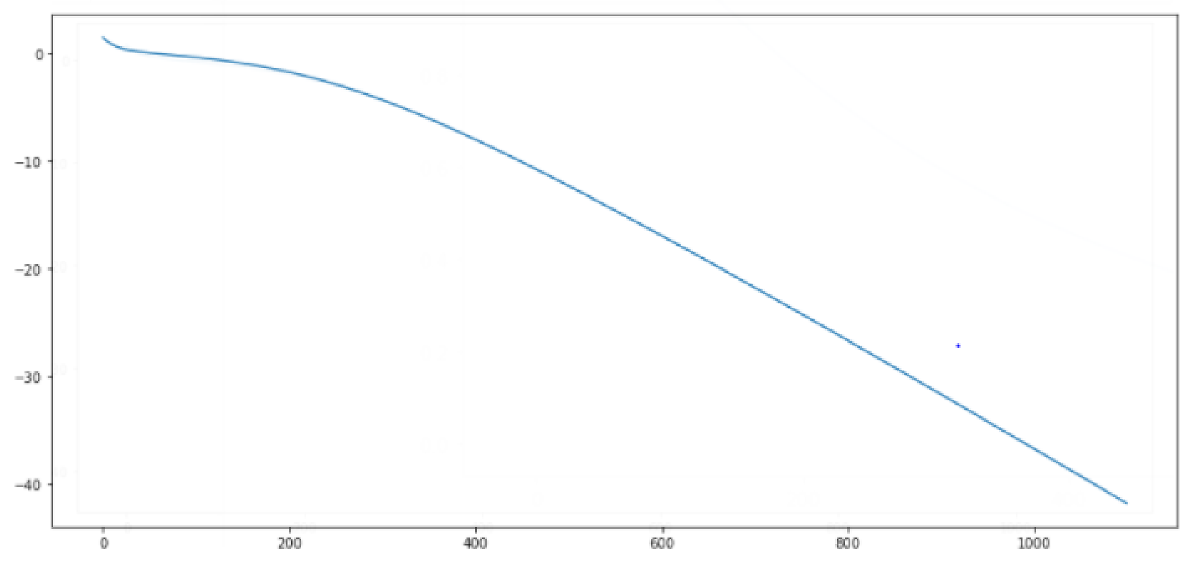
Learning rate = 0.01

Number of hidden units = 5

Epochs = 5000

At a learning rate of 0.1 you can see signs of the data being overfitted the error @ 1000 epochs is

-36.79156 and the graph displays this. We can see how the data is being over fitted. The test error at this learning rate was 0.98.



Q4

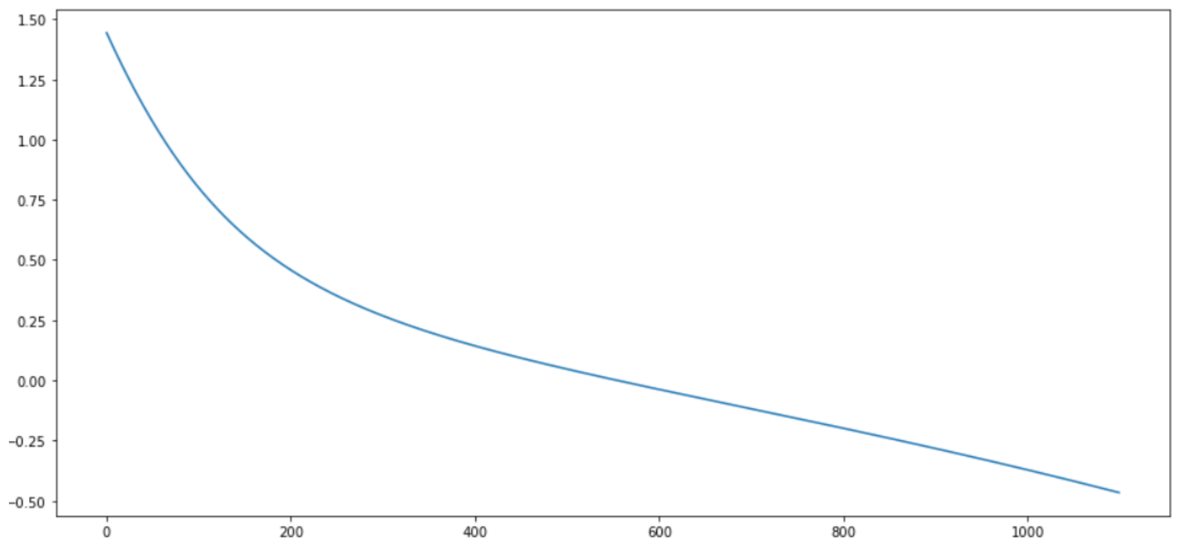
The average error on last epoch the training set was 0.04665 and the average error on the test set was 0.21441 so it hasn't learned as well as the model did when solving the XOR problem. The learning rate and number of epochs was hard to optimise for this training data. The best I managed was

lr = 0.005

epochs=1100

From the graph the training is very smooth so I don't believe these are the best parameters.

Epoch loss graph



This learning rate curve is very smooth which I think shows that it did not learn very well.

Conclusion

Multilayer perceptron’s and neural networks are a difficult concept to grasp. But after building one I think I understand them better than if I was using some preformulated code where I just had to plug in some data. Deciding on the hyper parameters felt like guesswork but as I changed them patterns began to emerge and I purposely skewed some of the numbers just to see the effects of over or underfitting on the predictions. The second data set was much more difficult to try to fine tune than the first. Small changes to the hyperparameters changed the error rates significantly whereas with XOR it didn’t seem to affect it as much. The datasets used were small so they trained very quickly even with small learning rates. However, if you were training models with very large datasets these hyperparameters are even more important to get right especially if the training was taking days. It would be a lot of time wasted just to have to try it all over again with new parameters. There are many ways to choose hyperparameters from trial and error to rules of thumb. I followed the rule of thumb first I start with a learning rate of between 0 and 1 with 1000 epochs and adjusted from there through trial and error. There are many algorithms that can help you fine tune hyperparameters such as Random Search which will iterate through the different possible parameters randomly and return the combination of parameters that performed the best. Grid search will try every possible combination from a hyperparameter grid and it return the best model with the best hyperparameters. Along with these two there are many more specifically designed algorithms for hyper parameter fine tuning. These algorithms such as Sci-kit learn and Hyperopt generally use some form of grid search, random search and Bayesian Optimization or a combination of them to fine tune the hyperparameters. So, it seems we are a long way from a perfect solution when it comes to training neural networks and still a lot of trial and error are needed to help improve machines predictive power and learning.