COMP 30230 – Final Assignment

Connor Jensen 15542443

Description of Code

The randomise class takes in the number of neurons and the number of inputs in order to create the random weights needed for later. The randomise class is called twice for each network once for the lower layer which takes in the number of hidden neurons and the number of inputs. Then the randomise class is called for the upper layer which takes in the number of hidden neurons for its input value and the number of outputs for the number of neurons in that layer.

The neural network class contains four functions: sigmoid, sigmoid derivative, train and predict. Sigmoid returns the sigmoid of a given value and sigmoid derivative returns the derivative of the sigmoid. Train is where back propagation occurs this is done by having a for loop for the number of chosen epochs. This takes in the training input data and output data and calculates the error by finding the difference between the two different layers. The delta is calculated by this error multiplied by the sigmoid derivative. The lower layer error is calculated by using the dot product of the upper layer weights and its delta. The lower layer delta is calculated the same way which is the error multiplied by the sigmoid derivative for the lower layers output. The lower layer adjustments are calculated by multiplying the lower layer delta and the training set inputs. While the upper layer adjustments are calculated by the lower layers output and the upper layer delta. Each layer's weights are then updated by adding the relevant layers adjustments. At each iteration of this loop the upper layers error is appended into an array and the array is saved to a csv file at the end of the loop because the error at each iteration is required for submission.

The predict function works by forward propagation which feeds the inputted data into each layer and sends these layers output into the next layer. This is done by sending the sigmoid of the product of the input and the weights for neuron in the lower layer to the upper layer. Then the upper layer takes this result and does the same and returns the chosen number of outputs.

Experiments Performed

The first experiment required as part of this assignment was to train the network on the XOR output for each combination of two given inputs. We were then required to test this network given each possible input. Table 1 shows the results of this experiment.

Input 1	Input 2	Output
0	0	0.13750659
0	1	0.90490533
1	0	0.9050697
1	1	0.06475196

Table 1

As you can see the network successfully predicted the output for each possible given input when you round to the nearest number with each prediction being quite close to the targeted output and none of the predictions being uncertain which would be a result of around 0.5. The lower layer of the network takes in two inputs and contains three neurons. The upper layer of this network contains one neuron which takes in the output of the three neurons in the lower layer. This network was trained with 1000 epochs and Figure 1. Below shows error at each epoch

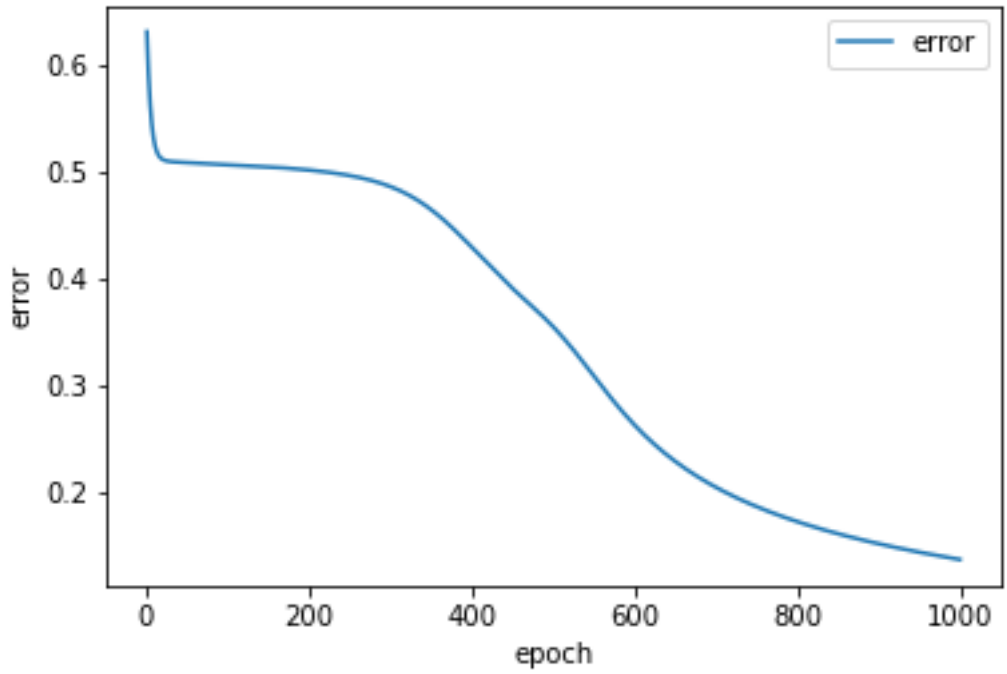

Figure 1

The second experiment required was to generate 2000 sets of 4 random vectors between 0 and 1. The output was generated by finding the Sin (Vector1 – Vector 2 + Vector 3 – Vector4). This was done by using numpy. random.rand(200,4) and applying a function that calculates the output and applies to the matrix. The first 150 vectors were then chosen as the training set and the last 50 were then used as a test set. The network was trained using 1000 epochs and 5 neurons in the lower layer. One thing noted during predictions was that if the expected output were a negative number the predicted value would approach zero, but the network never predicted a negative number. The closer the number was to negative 1 the predicted value became extremely small. Figure 2 shows the absolute value of error when plotted which can be compared to Figure 3 which shows the actual value of error. I calculated a mean error of 0.1304 between the predicted result and the actual result. This is quite low when you account for the network not being able to predict negative numbers. Table 2 shows some predicted results and actual results selected from the test set. As you can see the network is very accurate when it comes to predicting positive values.

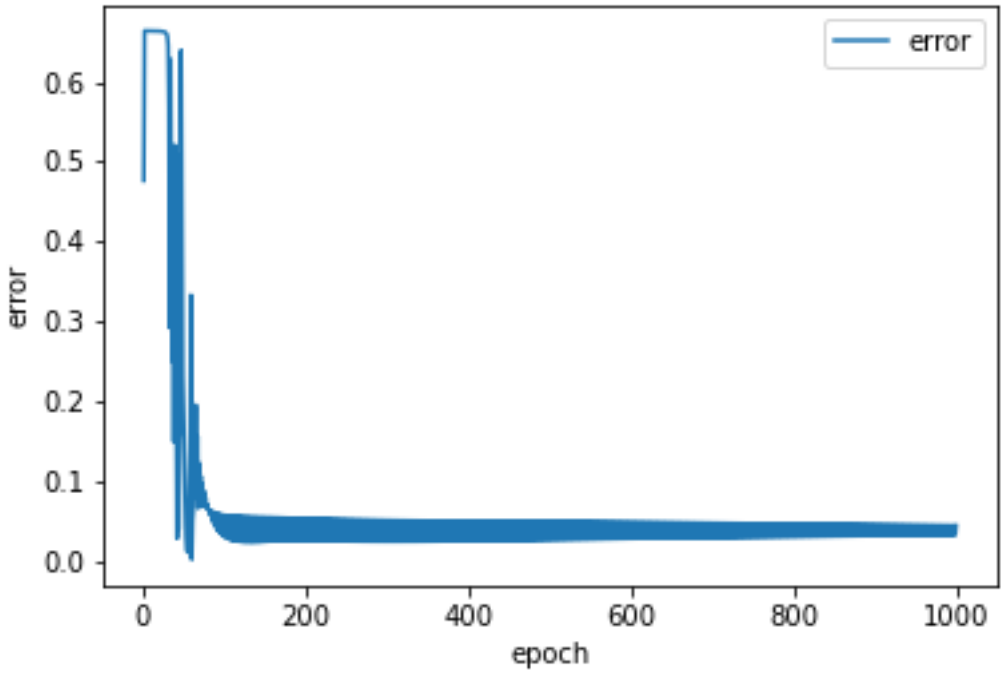

Figure 2

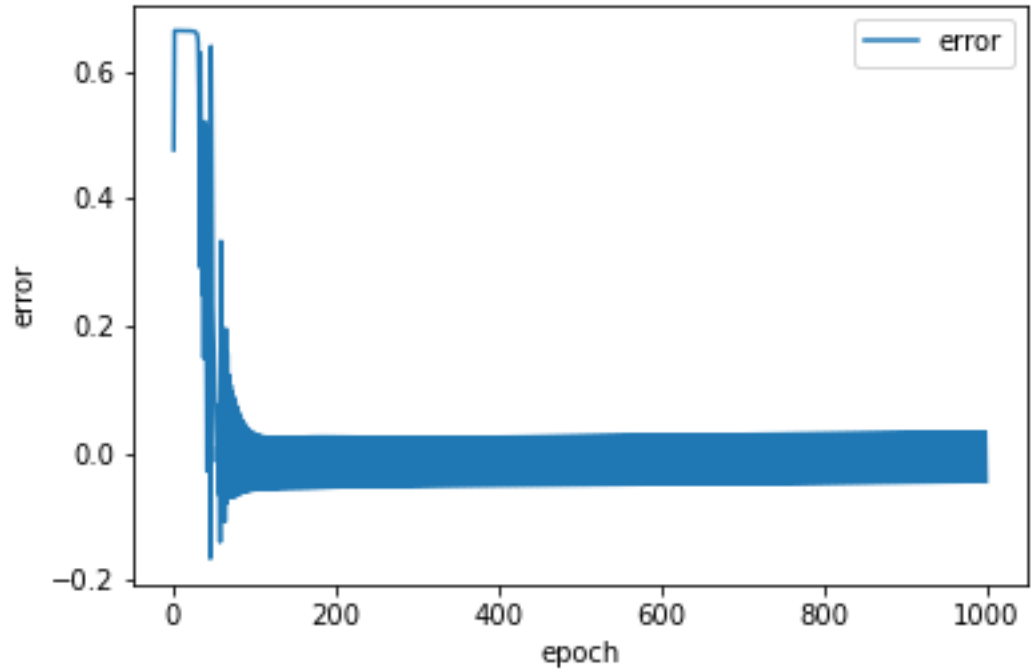

Figure 3

Predicted Result	Actual Result
0.01621728964700631	0. 007552665132724138
0.5261252007351235	0.5147122204964539
0.9132873788600925	0.9999000143200438
0.01905665990120327	-0.05313776086090345
3.5954052145565415e-05	-0.865705641010121

Conclusions

As I noted during the second experiment the network was not able to predict negative numbers. This is because I used the sigmoid as part of the training algorithm. Upon further research A way to address this could be to normalize the data between 0 and 1 or alternatively use a different function as part of training an example of this would be the Tanh function. Figure 4 shows us a comparison of the Tanh function and the Sigma function as you can see the Tan H function has a lower bound of -1 and an upper of 1 while sigma has a lower bound of 0 and an upper bound of 1. So, an advantage of the sigmoid function is that it maps into predicting probabilities more effectively.

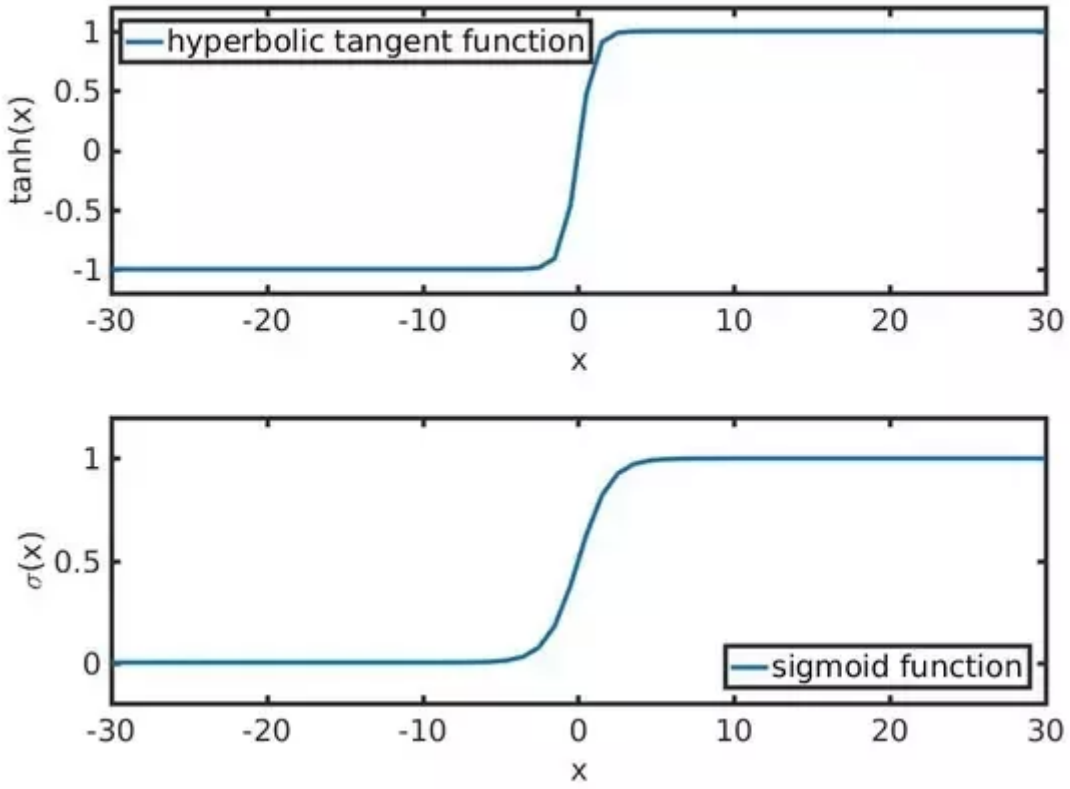

Figure 4