Connectionist Computing Assignment - 19300753

Programming Choices:

Number of Layers of Hidden Units:

For this investigation I will be using 1 hidden layer of units. From the users response here https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw I have learned that the increase in the number of layers does not increase the performance that much. As this assignment does not ask for challenges of massive complexity it is probably not necessary to add more than one layer. Also the assignment alludes to only one layer by the arrays that are needed to be initialised. Eg Z1 Z2 only for 2 activations, one hidden and one output layer.

Init the weights:

The weights should be a small positive value. The range that is recommended is 0 - 0.3: https://machinelearningmastery.com/neural-networks-crash-course/#:~:text=Weights%20are%20often%20initialized%20to,indicate%20increased%20complexity%20and%20fragility
I replicated this with finding a random number between 0 - 1 using the javascript Math.random() and multiplying it by 0.3.

Bias weighting:

Should be random value between 0 - 1 given the recommendation of: https://towardsdatascience.com/a-beginners-guide-to-neural-networks-part-two-bd503514c71a
It is how likely a neuron is to fire a 1.

Beginning Learning Value

The learning rate that I am beginning this experiment with is 0.1 as recommended by this article: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/
This is one of the parameters that I will be adjusting throughout my experiment's investigation.

Update frequency

This is the parameter that I have set to dictate whether or not the weights should be updated for that epoch or not. The weights will be updated if a random number between 0 and 1 < this value. I chose the value of 0.33 to begin as I wanted the weights to update less than half of the time. This is one of the parameters that I will be adjusting throughout my experiment's investigation.

Number of hidden units

This was dependent on the question. For each of the questions I started with the number of hidden nodes recommended to start with. From the recommendation of this user: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw it recommended that that number of hidden nodes can be generally represented as “the number of neurons in the layer is the mean of the neurons in the input and output layers”. For my testing I worked with this in mind.

Explanation of my Forward and Backward Propagation Calculation

Forward propagation
The forward propagation needs to happen in 2 steps, once for the hidden layer and once for the output layer.
<u>Hidden layer:</u>

$$\begin{tikzcd}
 & y = \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + bias \\
\boxed{\text{Step 1:}} \arrow[ru] \arrow[rd] & \\
 & y = \sum_{i=1}^{n} \{(\omega_i * x_i) + b\} \\
\boxed{\text{Step 2:}} \arrow[r] & z = y_{pred} = \frac{1}{1 + e^{-(\omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + b\ )}}
\end{tikzcd}$$

Its calculated like this:
Step 1: To find the activation: For each node in the layer it is equal to the summation of all of the nodes * those nodes' corresponding weights that are in the layer before + the bias. Need to store this to find the weight changes in the back propagation.
Step 2: To find the actual output = 1/1 + e^-(step1)

Backward propagation
Changing the weights to reduce the error:
To find $\frac{dC}{dW}$, the rate of change with respect to weights ie. how much need to change, I need to use the chain rule to use values that I have.
Chain rule = $\frac{dC}{dW} = \frac{dA}{dW} * \frac{dC}{dA}$
C = cost function or error
W = weights
A = Activation function

$\frac{dC}{dA}$ = Finding the derivative of the cost function
* Cost function = $\frac{1}{2}(target \ - \ output)^2$
$\frac{dC}{dA}$ = (target - output)
$\frac{dA}{dW}$ = derivative of the input weights
$\frac{dC}{dW}$ = (target - output) * derivative of the input weights
Then apply to the array that holds the weight changes.

Need to find the weight changes for the other layer. Need get the error of one of its output nodes * the weight that was applied to the original node. Repeat for all of the original nodes outputs.

Return the total error given this cost function: $\frac{1}{2}(target \ - \ output)^2$

*All of my test results including graphs and tables will be uploaded separately, but I will just include the necessary graphs here.

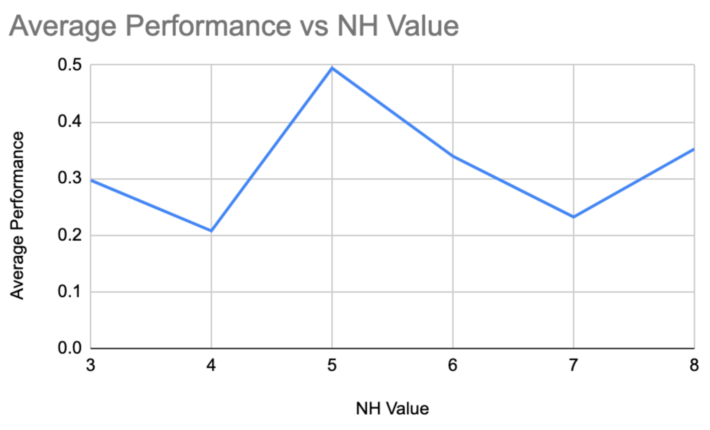
Question 1
Before testing, I find the best parameters to train the nodes with. The parameters I need to find the most optimal performance with are: Number of hidden units in the layer, the learning value and the frequency of when the weights are updated. I started with these parameters for the explanation in the Programming Choice section:

Number of Hidden Units	Learning Value	updateFreq	maxEpochs
3	0.1	0.33	10000

All of my tests for Question 1 and 2 can be seen in the "XOR" sheet of Results and Graphs.

<u>Hidden Units:</u>

Number of Hidden Units	Learning Value	updateFreq	maxEpochs
Changing	0.1	0.33	10000



The number of hidden units that I will be using will be 4.

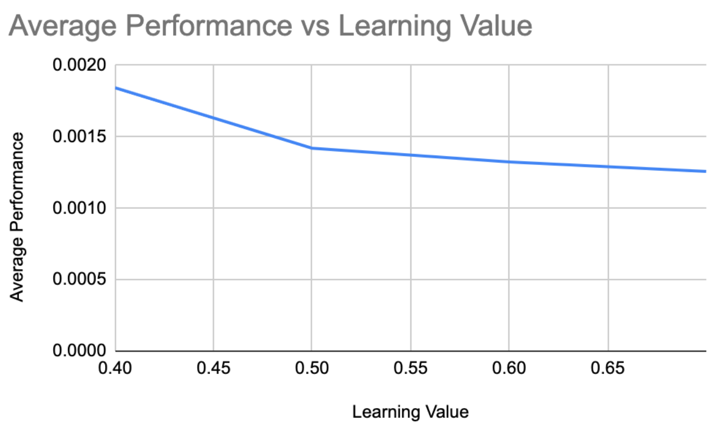
<u>Learning Value:</u>

Number of Hidden Units	Learning Value	updateFreq	maxEpochs
4	Changing	0.33	10000

For the Learning Value I tried the values = [0, 0.25, 0.5, 0.75, 1, 1.25] and got these results:

Learning Value	Average Performance
0	0.5721794236
0.25	0.07730366021
0.5	0.001307232112
0.75	0.0007969814321
1	0.0431772031

For further investigation of the specific learning value I will be investigating further the values = [0.4, 0.5, 0.6, 0.7]



The most optimal learning Value is 0.7

<u>Update Frequency:</u>

Number of Hidden Units	Learning Value	updateFreq	maxEpochs

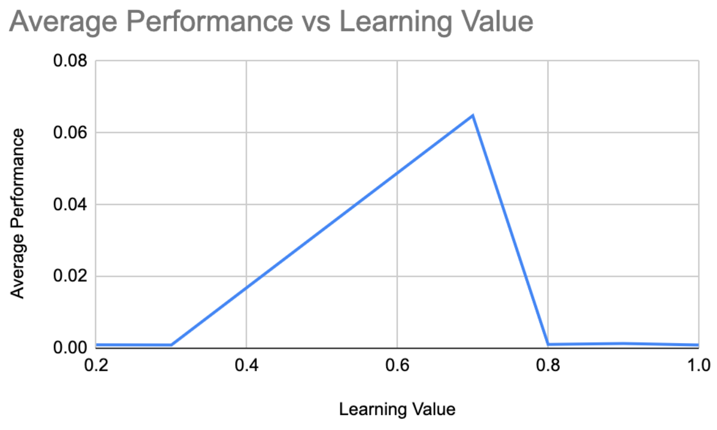
<u>Conclusion:</u>
So for my testing I will be using the following parameters:

4	0.7	Changing	10000

For the updateFreq I tried the values = [0, 0.25, 0.5, 0.75, 1] and got these results:

updateFreq	**Average Performance**
0	0.5834353971
0.25	0.0007780082106
0.5	0.04461118527
0.75	0.001084282091

I continued the investigation comparing 0.2, 0.3, 0.7, 0.8, 0.9, 1 (the lowest values on the graph):



The best performance is at update frequency of 0.2, 0.3 and 1. I will be using 0.3 for the rest of the investigation as it was the lowest between 0.2 and 0.3. Having a update frequency of 1 ie. changing the weights every time, may not be consistent and be input dependent.

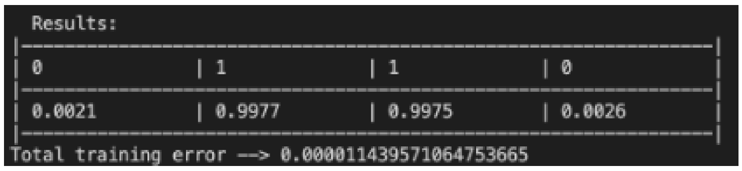
Number of Hidden Units	**Learning Value**	**updateFreq**	**maxEpochs**
4	0.7	0.3	500000

I will be increasing the epoch size for the testing. The test runs very fast and the training seems to continue to reduce in error.

Question 2

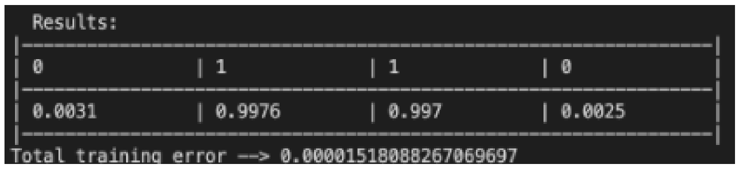
Can my MLP predict correctly for my XOR example? (See parameters in question 1).

Result 1



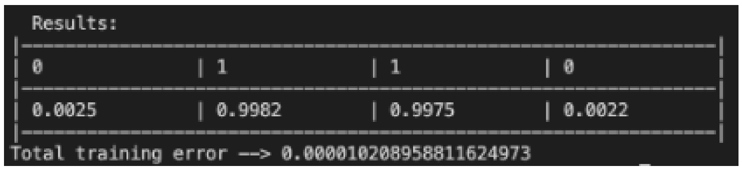
```
  Results:
|--------------------------------------------------------------|
| 0              | 1              | 1              | 0         |
|--------------------------------------------------------------|
| 0.0021         | 0.9977         | 0.9975         | 0.0026    |
|--------------------------------------------------------------|
Total training error --> 0.000011439571064753665
```

Result 2



```
  Results:
|--------------------------------------------------------------|
| 0              | 1              | 1              | 0         |
|--------------------------------------------------------------|
| 0.0031         | 0.9976         | 0.997          | 0.0025    |
|--------------------------------------------------------------|
Total training error --> 0.00001518088267069697
```

Result 3



```
  Results:
|--------------------------------------------------------------|
| 0              | 1              | 1              | 0         |
|--------------------------------------------------------------|
| 0.0025         | 0.9982         | 0.9975         | 0.0022    |
|--------------------------------------------------------------|
Total training error --> 0.000010208958811624973
```

Final Results:

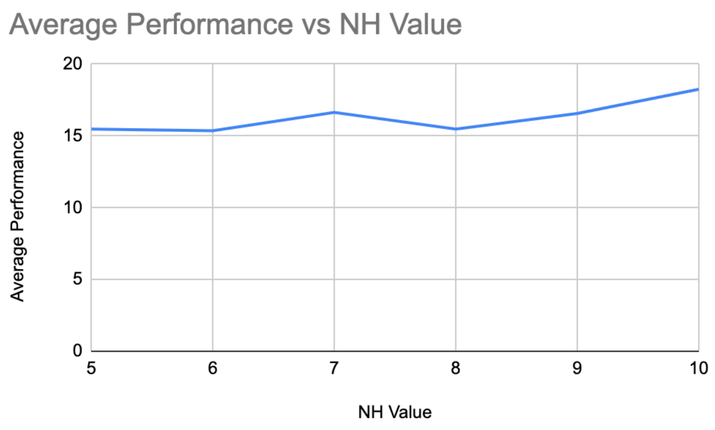
	Try 1	**Try 2**	**Try 3**	**Average Error**
Training Error	0.0000114 3957	0.0000151 8088	0.0000102 0895	0.0000122 7647

The error is very small and the system seems to predict the xor in every test example with very little error.

Question 3:
All of my tests for Question 3 and 4 can be seen in the “SIN” sheet of Results and Graphs.

<u>Hidden Units:</u>
Starting with hidden units 5 from the question.

Number of Hidden Units	Learning Value	updateFreq	maxEpochs
Changing	0.1	0.33	10000



Number of hidden units is best at 6.

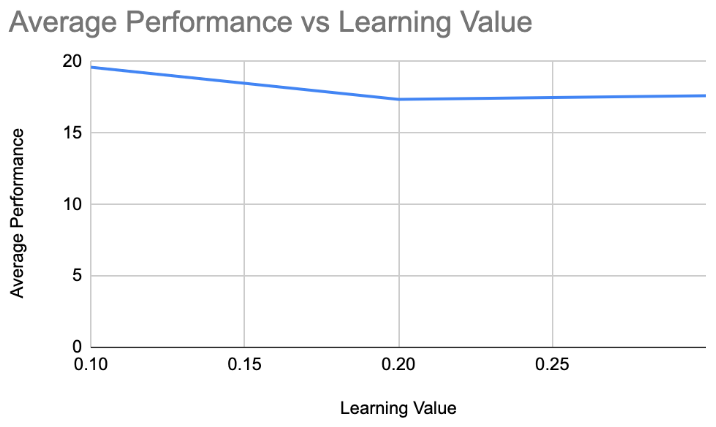
<u>Learning Value:</u>

Number of Hidden Units	Learning Value	updateFreq	maxEpochs
5	Changing	0.33	10000

For the Learning Value I tried the values = [0, 0.25, 0.5, 0.75, 1] and got these results:

Learning Value	Average Performance
0	38.44674151
0.25	16.20775001
0.5	18.07398542
0.75	31.11881936
1	33.47602626

For further investigation of the specific learning value I will be investigating further the values = [0.1, 0.2, 0.3]



The most optimal learning Value is 0.2

<u>Update Frequency:</u>

Number of Hidden Units	Learning Value	updateFreq	maxEpochs
5	0.2	Changing	10000

For the updateFreq I tried the values = [0, 0.25, 0.5, 0.75] and got these results:

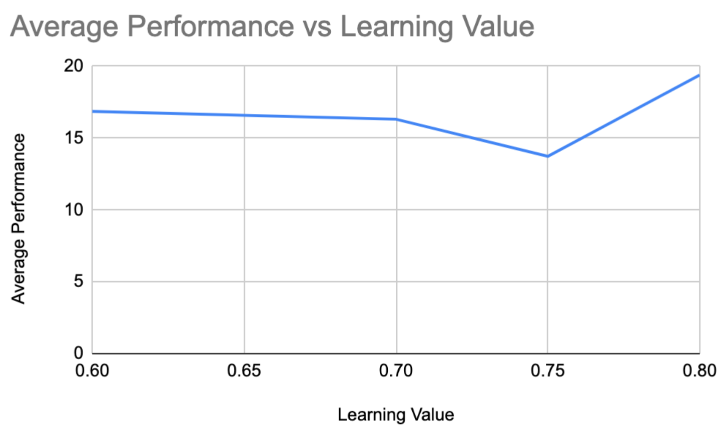
updateFreq	Average Performance
0	40.0528649
0.25	16.54059024
0.5	19.77637068
0.75	13.72764595

<u>Conclusion:</u>
So for my testing I will be using the following parameters:

Number of Hidden Units	Learning Value	updateFreq	maxEpochs
6	0.2	0.75	100000

I will be increasing the epoch size for the testing. The test runs very fast and the training seems to continue to reduce in error.

I continued the investigation comparing 0.6, 0.7, 0.8 (the lowest values on the graph):



The best performance of the updateFreq is still the 0.75 from the first experiment.

Question 4

Can my MLP predict correctly for my SIN example? (See parameters in question 2). For each of the results there are far more test examples but I didn't include them here. You can see a few more of the examples in the SIN sheet of the sheets.

<u>Result 1</u>	<u>Result 2</u>
Testing Val True Val	Testing Val True Val
|----------------------------------|	|----------------------------------|
-0.85 ---> -0.883	-0.093 ---> -0.093
|----------------------------------|	|----------------------------------|
0.406 ---> 0.406	-0.406 ---> -0.359
|----------------------------------|	|----------------------------------|
-0.067 ---> -0.076	0.97 ---> 0.951
|----------------------------------|	|----------------------------------|
-0.025 ---> -0.046	0.611 ---> 0.581
|----------------------------------|	|----------------------------------|
0.234 ---> 0.27	0.897 ---> 0.964
|----------------------------------|	|----------------------------------|
0.118 ---> 0.149	0.86 ---> 0.917
|----------------------------------|	|----------------------------------|
-0.601 ---> -0.58	0.766 ---> 0.777
|----------------------------------|	|----------------------------------|
0.892 ---> 0.996	-0.909 ---> -0.982
|----------------------------------|	|----------------------------------|
0.793 ---> 0.765	-0.686 ---> -0.686

```
|----------------------------------|
-0.894 ---> -0.94
|----------------------------------|
```

```
|----------------------------------|
-0.341 ---> -0.309
|----------------------------------|
```

Result 3

```
Testing Val True Val
|----------------------------------|
0.771 ---> 0.753
|----------------------------------|
0.879 ---> 0.928
|----------------------------------|
-0.676 ---> -0.657
|----------------------------------|
0.242 ---> 0.235
|----------------------------------|
0.92 ---> 0.968
|----------------------------------|
0.684 ---> 0.639
|----------------------------------|
0.923 ---> 0.941
|----------------------------------|
0.281 ---> 0.316
|----------------------------------|
0.812 ---> 0.792
|----------------------------------|
0.106 ---> 0.07
|----------------------------------|
```

Final Results:

	Try 1	**Try 2**	**Try 3**	**Average Training Error**
Training Error	14.75452764	18.66526723	16.35234232	16.5907124

This is a very good total error for 100 tests. The individual predicted error for each of the tests is approx 0.16 which is very small. For each of the test results 1 - 3 you can see that the system predicts very well. The value is very often predicted to the 1st decimal place whereas for some values eg. -0.686 ---> -0.686 the value is predicted exactly to the third decimal place. I am very happy with this prediction.

Letter Recognition

I found that the letter recognition was very slow when I used it with the full data set so I am going to try find the most optimal parameters using a smaller data set using the javascript code:

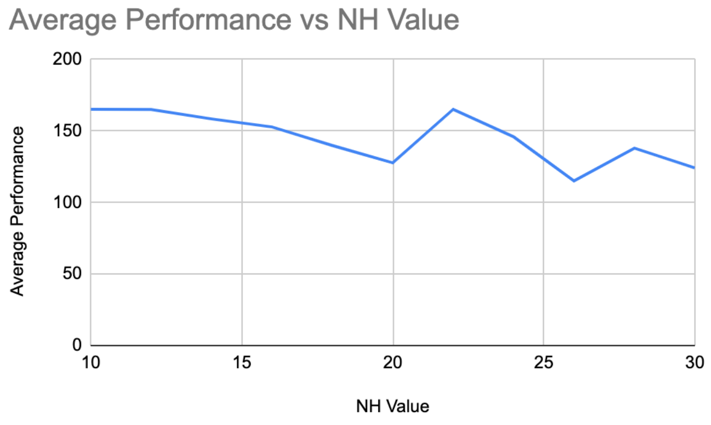
```
result = result.slice(0, 500);
```

All of my tests for this part of the assignment can be found on the “LETTER” sheets of the google sheet supplementary document.

Hidden Units:	Learning Value:

Starting with hidden units 10 from the question requisition. I am using a smaller epoch size for the training because the training very rarely trains well after this point.

Number of Hidden Units	Learning Value	updateFreq	maxEpochs
Changing	0.1	0.33	5000



Number of hidden units is best at 26.

Update Frequency:

Number of Hidden Units	Learning Value	updateFreq	maxEpochs
26	0.15	Changing	5000

For the updateFreq I tried the values = [0, 0.25, 0.5, 0.75].
I then continued my experiment for the values = [0.3, 0.4, 0.5].

I found very inconsistent results so I decided to stick with my original update Freq because I actually got the best results with that in the previous tests.

Conclusion

These are the most optimal parameters from my findings that I will be continuing the testing with. I am decreasing the maxEpochs for the full dataset because the training is very slow and the minimum of 1000 is required by the assignment question.

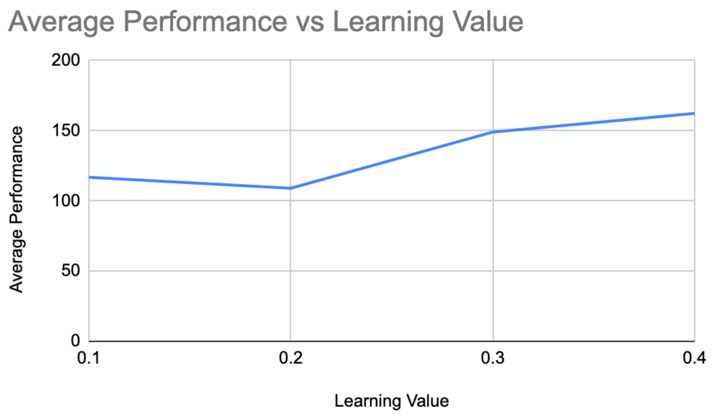
Number of Hidden Units	Learning Value	updateFreq	maxEpochs
26	0.15	0.33	1500

Number of Hidden Units	Learning Value	updateFreq	maxEpochs
26	Changing	0.33	5000

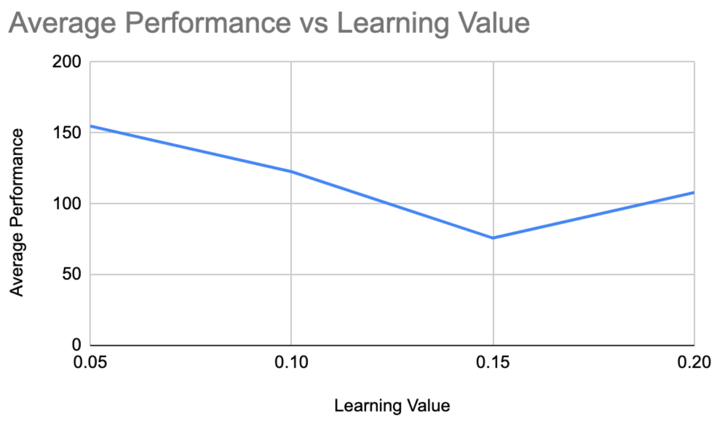
For the Learning Value I tried the values = [0.25, 0.5, 0.75] and got these results:

Learning Value	Average Performance
0.25	141.3782886
0.5	180.9769411
0.75	185.2682823

For further investigation of the specific learning value I will be investigating further the values = [0.1, 0.2, 0.3, 0.4]



I continued my investigation for more of a specific value between 0.05 and 0.2



I continued my investigation for more of a specific value but I found that 0.15 is the best learning value.

So can my MLP train correctly for the letter recognition dataset?
I trained my model on the 1500 training dataset. The error on the 500 predictions was very satisfactory on the 3 final trials:

	Try 1	**Try 2**	**Try 3**	**Average Training Error**
Training Error	591.5233286	419.9013281	458.1928669	489.8725079

All of the results of this testing are available in the Results folder of my submission in the test1.txt, test2.txt, test3.txt. They are a bit distorted from the order that the javascript appends the file but all of the information is there and I was able to sort it in the sheet. Here is a snippet of the results (There are more of these results in the sheets file):

Test 1

Actual Output	**Predicted Output**	**Error**	**Correct?**
V	V	6.61E-22	Y
V	V	4.51E-16	Y
T	Z	1.97E-24	N
M	M	2.57E-14	Y
G	Q	1.98E-09	N
D	D	8.03E-19	Y
O	Q	1.88E-08	N

Total testing error → 591.5233286010371
No correct answers → 3517
Accuracy → 70.34%

Test 2

Actual Output	**Predicted Output**	**Error**	**Correct?**
V	V	8.17E-09	Y
V	V	6.50E-08	Y
T	J	0.00001422012356	N
M	M	6.88E-21	Y
G	Q	0.0002245171969	N
D	R	0.0001454531533	N
O	R	0.0003187984781	N

Total testing error → 419.90132806953926
No correct answers → 3624
Accuracy → 72.48%

Test 3

Actual Output	**Predicted Output**	**Error**	**Correct?**
V	V	1.56E-21	Y
V	V	2.90E-21	Y
T	T	2.68E-13	Y
M	D	3.36E-09	N
G	G	1.33E-10	Y

Conclusion

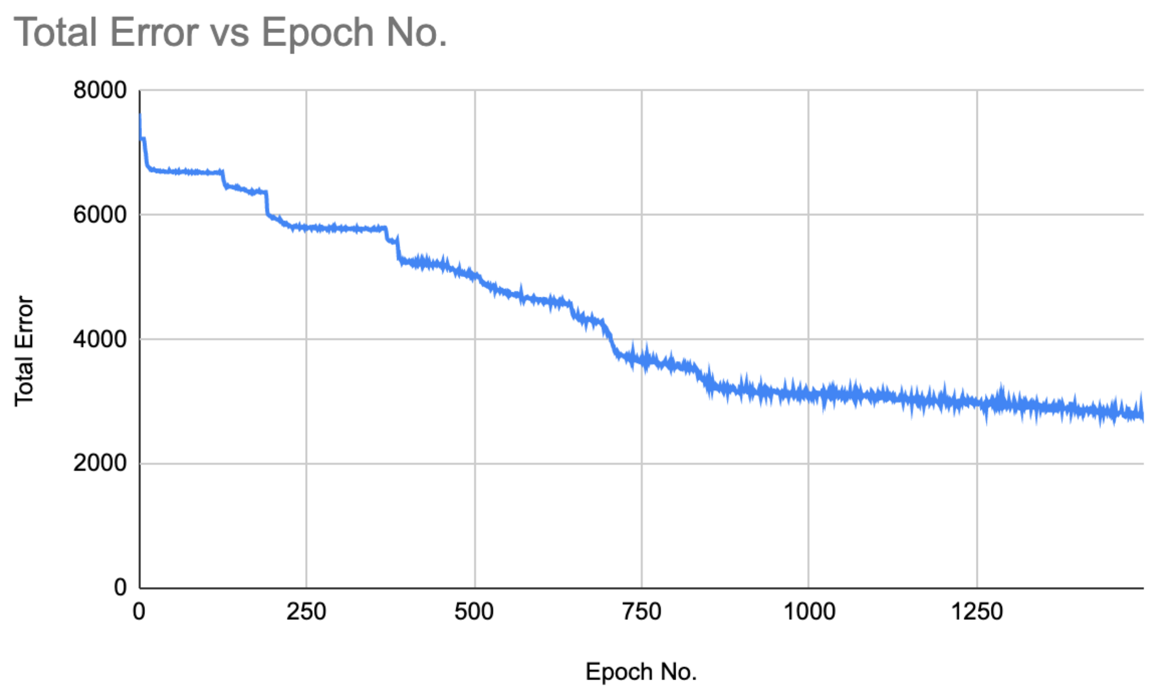
I am very happy with the prediction level of this system, I didn't think it would work as well as it did. When I was training on a smaller dataset it was difficult. The predictions were not very strong but the prediction error reduced a significant amount in my last tests. With extra investigation or by conducting more tests I could definitely get a lot more of an exact result but I think from my tests I have proven that my system works well.

D	D	0.00210333735	Y
O	O	0.000001764768421	Y

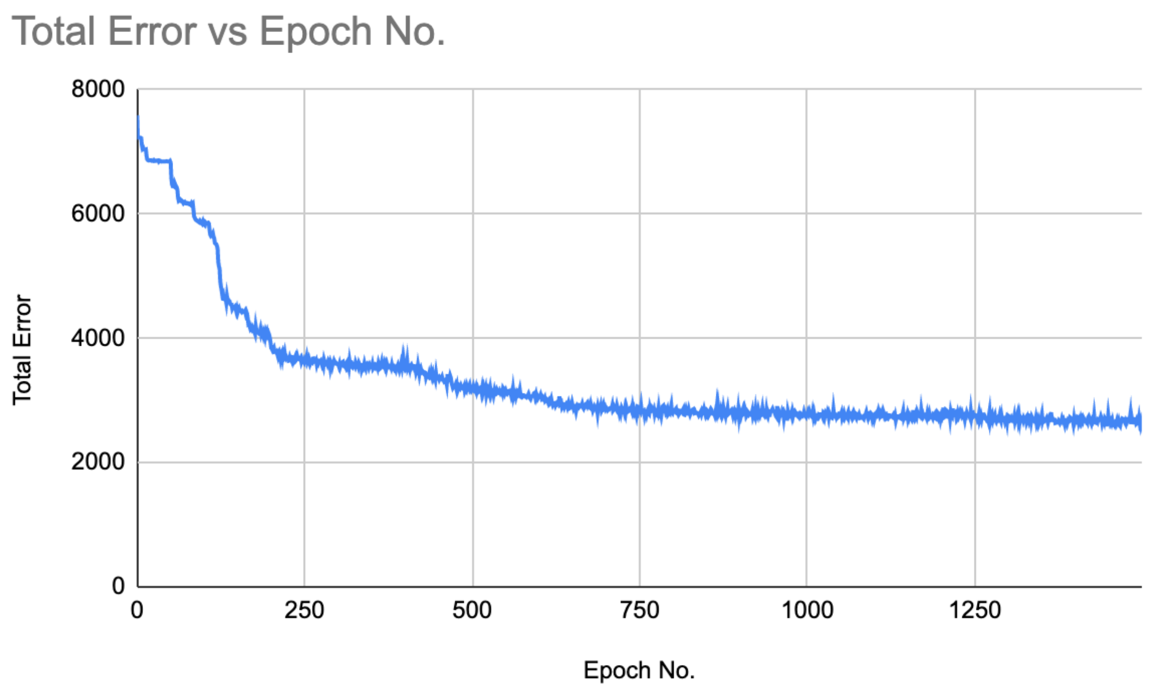
Total testing error → 458.1928668969892
No correct answers → 3514
Accuracy → 70.28%

Here is an overview of the learning that happened in the last letter recognition tests. Data recognition can be found in "Letter recognition" sheet of the results.

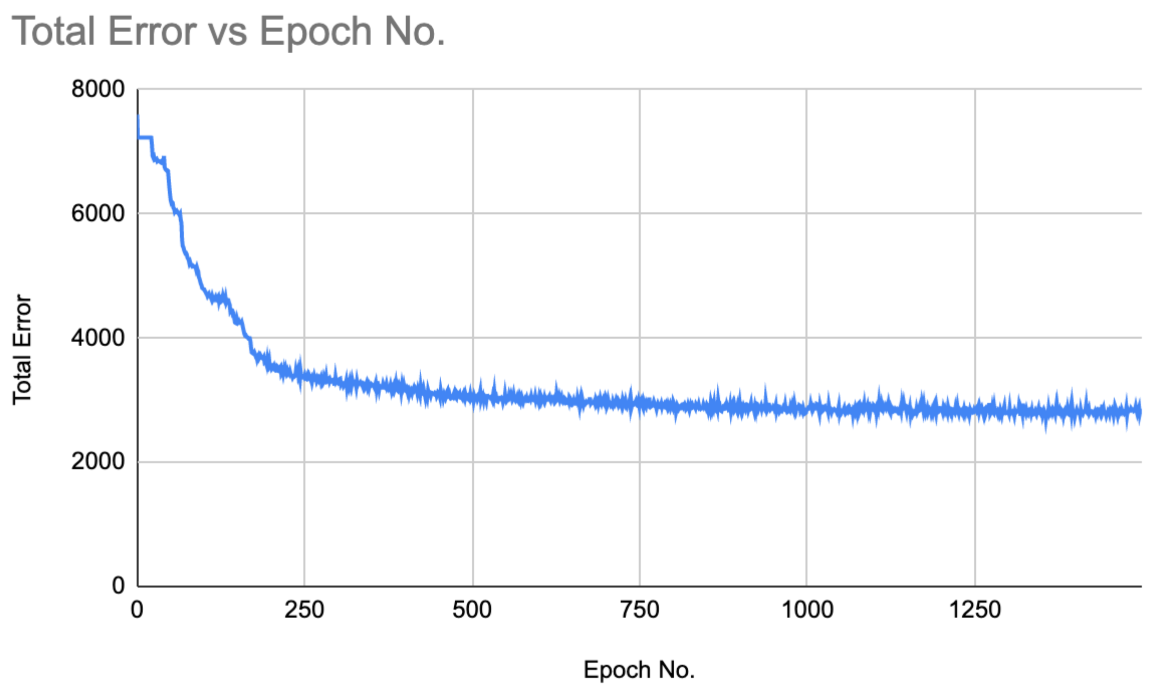
Test 1



Test 2



Test 3



Final Conclusion

From a workload perspective, it was very difficult and required a lot of trial and error. I certainly did not think I would have to spend as much time as I did to complete the assignment. Getting the NN to complete however was extremely satisfying and I'm glad it worked out and I have the results to prove my efforts.

I have learned a lot from this assignment. It was great to understand the structure behind the theory and learn the fundamentals of neural networks. It is a very important machine learning technique so it was nice to spend some time researching and learning about it.

The most challenging part of the assignment was the backward propagation element of the NN. The other functions were very easy to implement, I found. This taught me to really forward plan my implementation because I wasn’t able to program it from the top of my head. This is why I chose to explain my process in such detail in the report.

The rest of the project was not difficult as such, it just took a lot of time to complete, both the code and the testing parts. Overall I found the assignment hard but rewarding.