COMP41390 Connectionist Computing
Programming Assignment Report

Victoria Keane
16395531

Prelude– Code Rundown

For this assignment I used the Python programming language as it is what I am most adept with and have a working knowledge of libraries I understood would be useful to this project. One particular library, NumPy was integral to my code for simplifying the process of some necessary array operations.

In terms of structure, my code largely follows the suggestions in the assignment outline. My main MLP class contains three methods. get_random_weights, forward and backward. In the forward and backwards method I included an activation parameter, the methods are called with the additional input of the desired activation (tanh or sigmoid) to be performed for forward learning. In here I update the Hidden and Output variables using the desired learning algorithms performed on both the lower and upper activation arrays.

My backpropagation method "backward" then takes the inputs as arguments and the desired targeted outcome, activation or learning method, and the learning rate. From here the derivatives of "sigmoid" or "tanh" are computed and then used to find the weight deltas of the upper (output) and lower (hidden) layers In addition to these, having a target to compare output results with also enabled me to find error on the output.

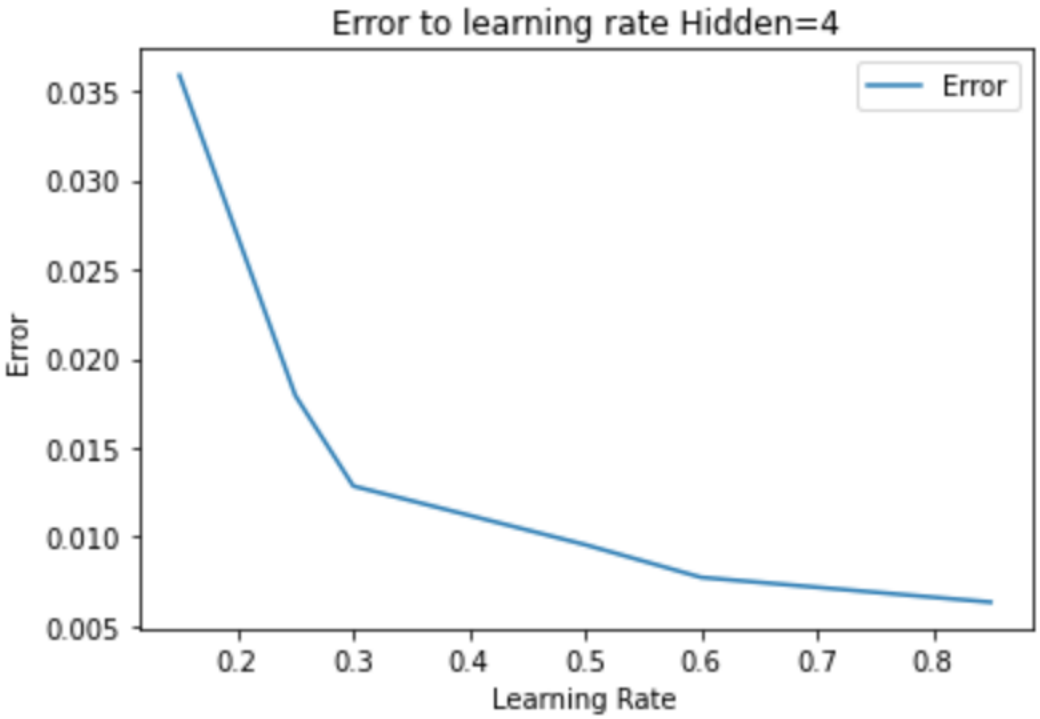
The final method, update_weights simply applies to changes found from the backwards method and resets the delta weight arrays.

For my functions created to answer the questions I followed mostly the suggested method provided to us in the assignment outline again. These methods open and write to a txt file the results of my computations using the MLP. These computations return the desired target and actual results for outputs at various different epochs and learning rates. On my first question I set the maximum epoch at 20,000 . However for the Sin question I found that anything over 5000 tended to produce decent results and 10,000 epochs seemed to hit a maximum in my computer's processing power. Musings on results will be discussed below.

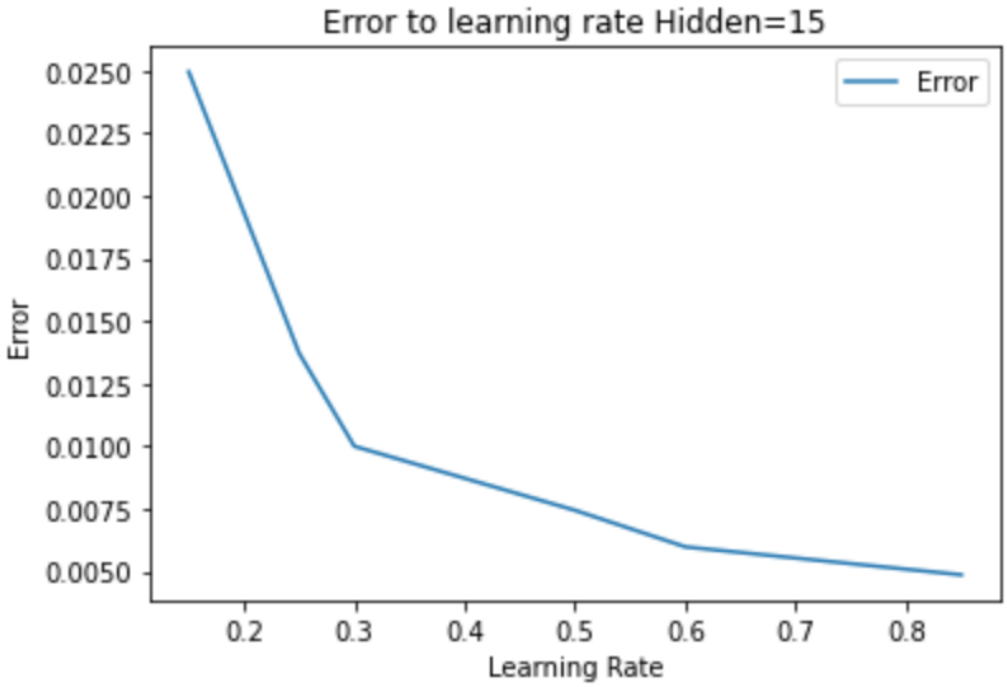
Q1

The XOR problem is a common Neural Network issue where an NN is tasked with correctly predicting the outcome of XOR logic gates when provided with binary inputs. For this task I experimented with a number of learning rates to try reach a conclusion on which could produce the best outcome.

Looking at all results collectively the margins of error across all learning rates proved to be slim, and as the rate increased the error margins would decrease. The image below displays the error rate at my max_epoch of 20,000 for each learning rate. We can see notable gains in adjusting the learning rate from 0.15 – 0.3, where the error margin reduces from 0.03 to 0.012. Error improvements between the learning rate of 0.4 to 0.6 are slimmer, whilst 0.6 onwards indicates that improvements have started to plateau somewhat.



Out of curiosity I also decided to test the NN using a larger pool of hidden units. The graph below shows a very similar curve indicating the general effect of the learning rate on improving error is similar. However we can also see that the margin of errors have overall improved again. With learning rate 0.15 yielding an error of 0.0250 – already an improvement on the same result from the initial 4 hidden units.



Q2. Below are some outtakes from my txt files. I selected the best performing learning rate (0.85) from each and their results. The results themselves are ideal as they are very close to each target – with every target of 1 reaching and output of .99 and 0 targets always returning zeros to at least 1 - 3 decimal places. I was initially sceptical that increasing the hidden units could make the model overfitted given the lack of data being passed to the MLP. Ie its few inputs. However the difference did not appear to be stark in contrast as both instances did not produce any drastically different outcomes. From analysing the below it

can be determined that the real numbers closest to the output align very similarly with the targets. The lowest margin of error is in that the image built from the MLP and 4 hidden units, here the array returns “0.001…” when a 0 is requested.

```
learning rate: 0.85
Error at Epoch: 0 is [0.00883166]
Error at Epoch: 100 is [0.00882051]
Error at Epoch: 500 is [0.0087763]
Error at Epoch: 1000 is [0.00872194]
Error at Epoch: 4000 is [0.00841533]
Error at Epoch: 8000 is [0.00805181]
Error at Epoch: 10000 is [0.00788655]
Error at Epoch: 12000 is [0.00773088]
Error at Epoch: 16000 is [0.00744487]
Error at Epoch: 20000 is [0.0071879]

target: [0]     results: [0.01251258]
target: [1]     results: [0.99282732]
target: [1]     results: [0.99283028]
target: [0]     results: [0.0018964]
```

Outputs from 4 hidden units

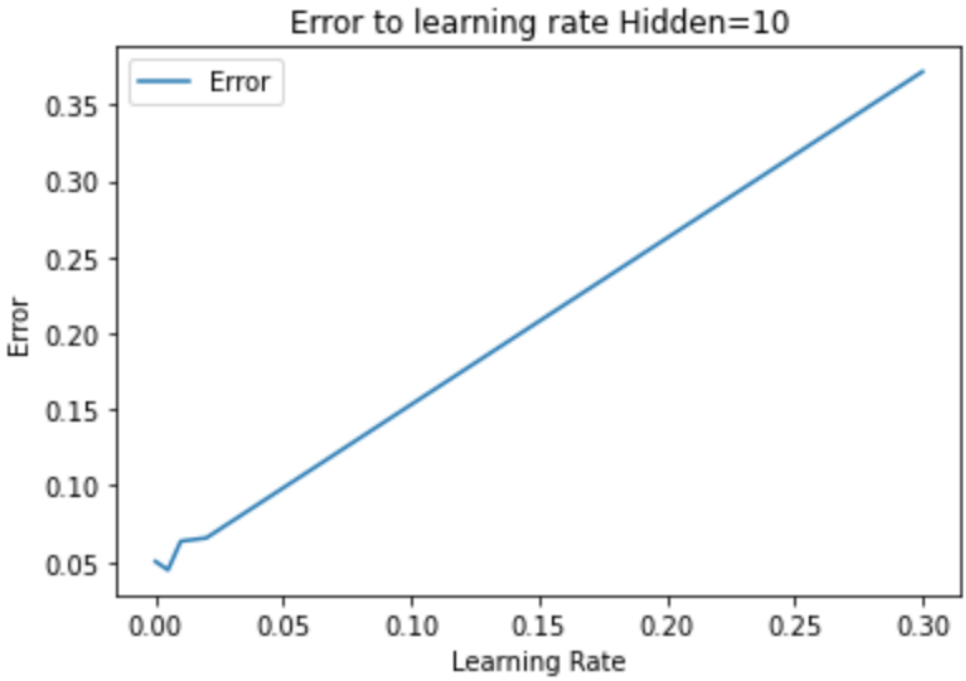
```
learning rate: 0.85
Error at Epoch: 0 is [0.00611569]
Error at Epoch: 100 is [0.00610794]
Error at Epoch: 500 is [0.0060772]
Error at Epoch: 1000 is [0.0060394]
Error at Epoch: 4000 is [0.00582615]
Error at Epoch: 8000 is [0.00557318]
Error at Epoch: 10000 is [0.00545814]
Error at Epoch: 12000 is [0.00534974]
Error at Epoch: 16000 is [0.00515053]
Error at Epoch: 20000 is [0.00497149]

target: [0]     results: [0.00540684]
target: [1]     results: [0.99513978]
target: [1]     results: [0.99507052]
target: [0]     results: [0.00468926]
```

Outputs from 15 hidden units

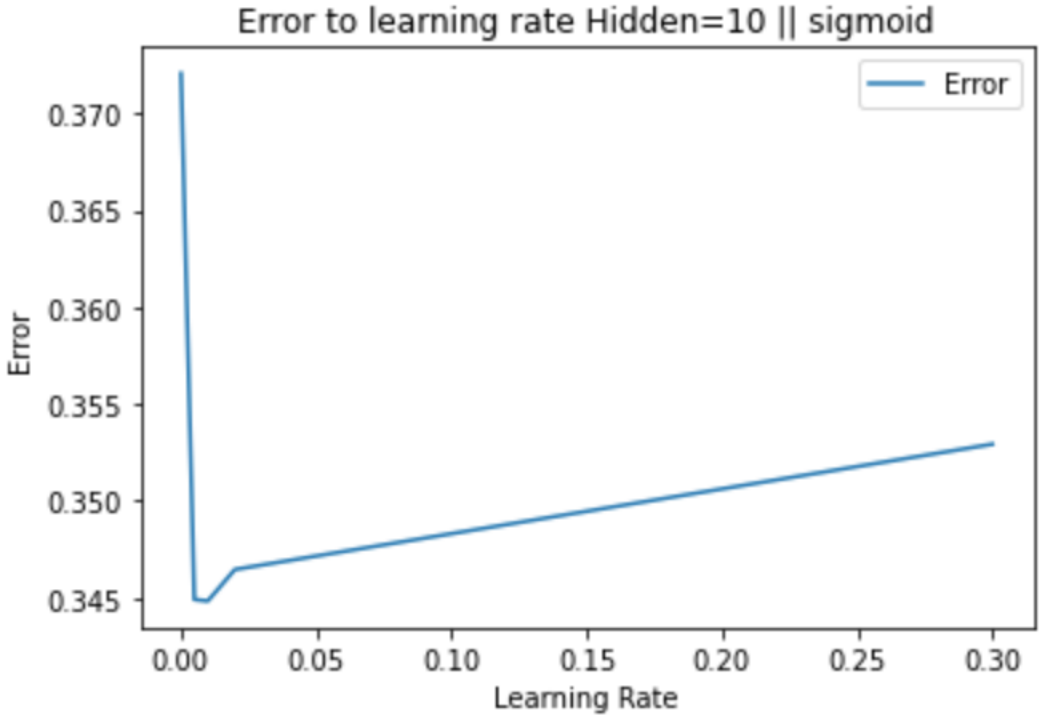
Q3

Question 3 relates to finding the sin of a group and passing it through a perceptron to achieve a desired output. In my code I used a for loop to compute sin.

For Question 3 I focused on running the code using the tanh activation function. The run time of this question was considerably higher than the last due to the large vector arrays and as a result I kept the epoch to 10,000 as the run processing time was too length. The reason for this can be deduced to the scale of the operation, s computations on 500 vectors will naturally be more time consuming than the much smaller input taken in the XOR question. As a matter of interest I also ran the question using sigmoid in order to compare results between the two. In this instance each have been run using 10 hidden units. In the below two graphs, we can see that sigmoid yields a much higher error rate than Tanh, and errors steadily climb upwards as the learning rate moves away towards 1. With tanh we can also see that learning rates closer to zero yield favourable results. We can also see that there is a marked overall improvement over sigmoid. With Tanh we can also see that after a learning rate of roughly 0.2, the error rate seems to steadily double with each learning increase of 0.05. In addition, I found that when using tanh at a learning rate of 0.3, the algorithm would only return binary outputs.

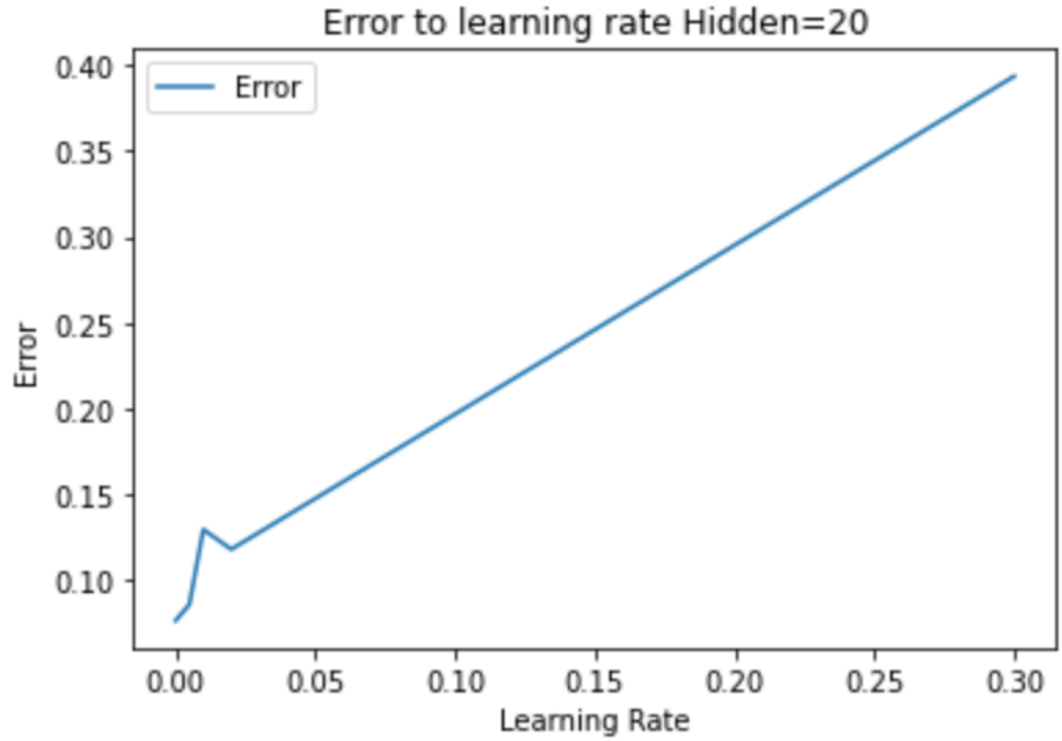


Tanh A



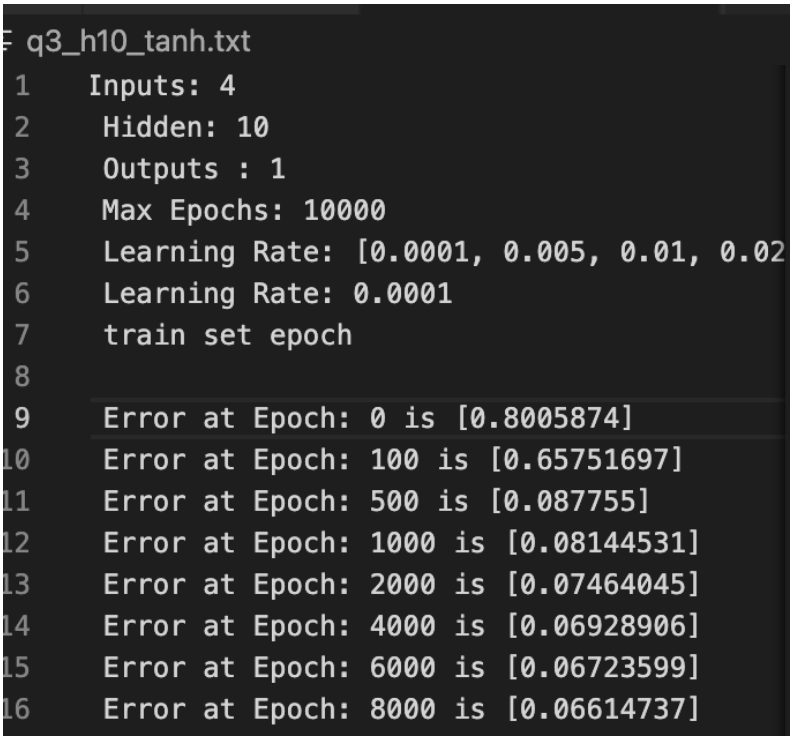
Sigmoid Q3

I again also experimented with altering the number of hidden inputs below. Again, extremely small values are proven to be favourable for results however not quite to the same extent as they are in my initial running of the question with 10 hidden inputs. Improved results from increased variables to the hidden layer could have resulted in some slight overfitting. Panagiotis Antonadis of Baeldung [1] agrees with the sentiment and determined that tanh operates as more of a “stretched” version of sigmoid. We can visualise that above in how the pattern observed in the Error to Learning Rate for Q3 with the hidden inputs set as 10 follow the same appearance. In the TanhA graph and Sigmoid Q3 graph both exhibit a dip at learning rate 0.0001 before emerging into a steady, linear gradient. Tanh provides a closer look at learning trends that may not look so visible/pertinent in sigmoid.

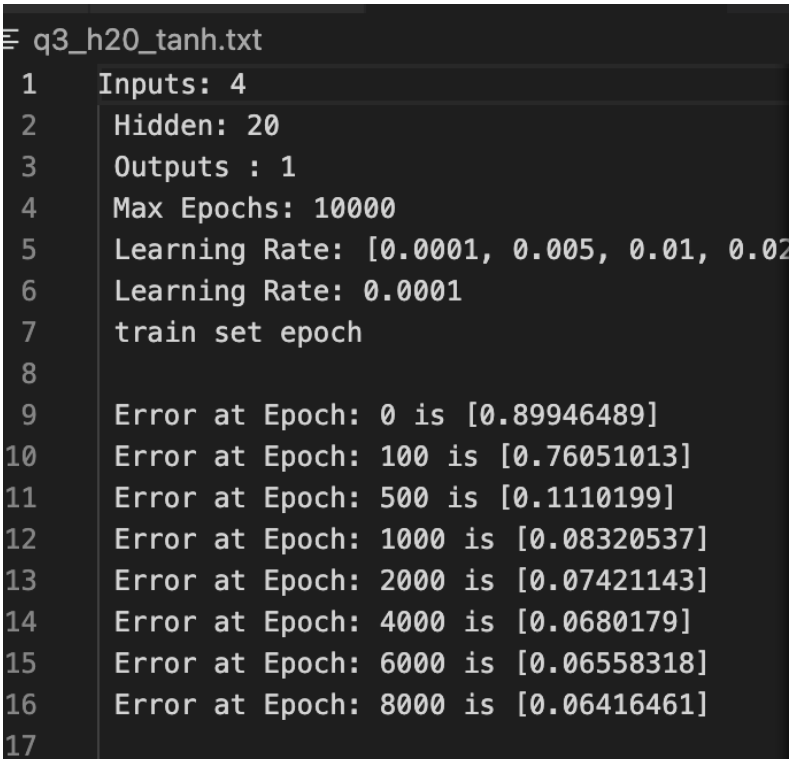


Q4

For question 4 I consulted with the best performing learning rates (0.0001) of Q3 where hidden units equals 10 or 20. During training it became very apparent that the best outcome would be situated as close to zero as possible as any increase in small numbers as learning rates could negatively affect the error rate. We can see this again in the tanh graphs above. In terms of training, the model performed well. On the 10 hidden input instance the epoch error decreased as far as 0.06%.

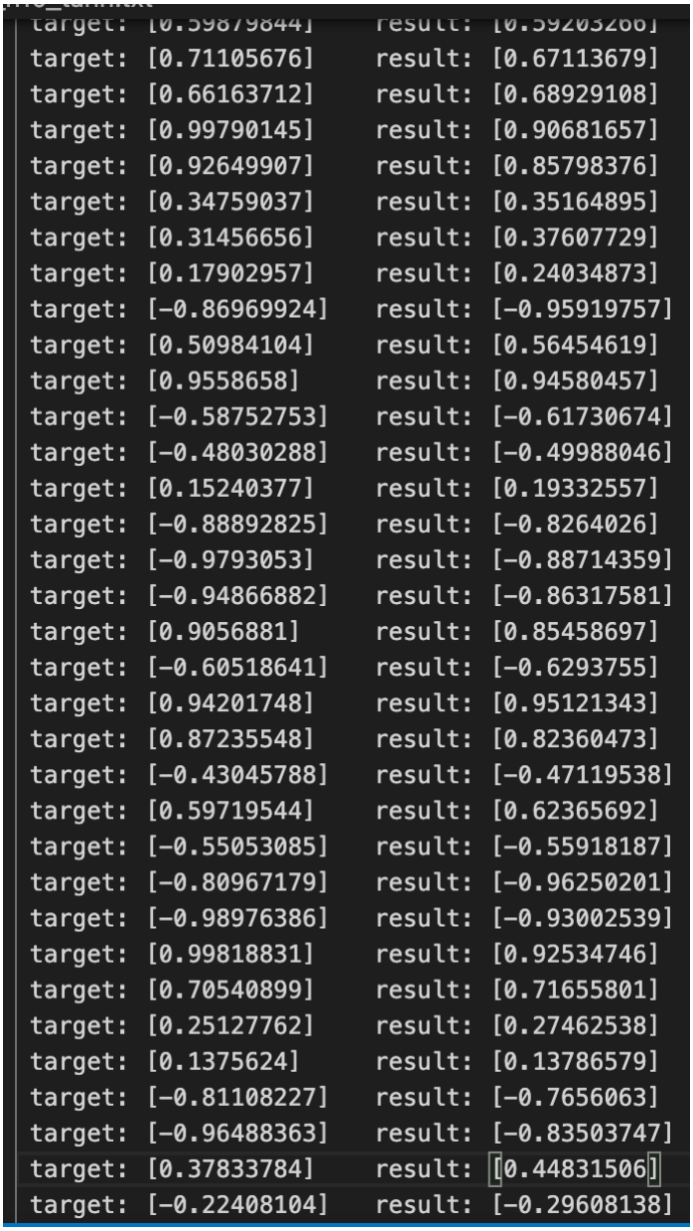


Training Error where Hidden Units = 10



Training Error where hidden units = 20

Here I have a collection of predictions that are the result of the 0.0001 learning rate on hidden units = 10. From these we can see quite high accuracy and little deviation in the outcomes.



Q5 – Special Question

For Question 5 I did not manage to pass the given letters database and return letter predictions. To my understanding I thought of it as follows. Letter Recognition had to be passed as a df and. I then split the df into into a “Letter Capital” column and another df that would hold the remaining variables. A key issue was effectively translating language into the Relevant associated UTF-8 chracter.