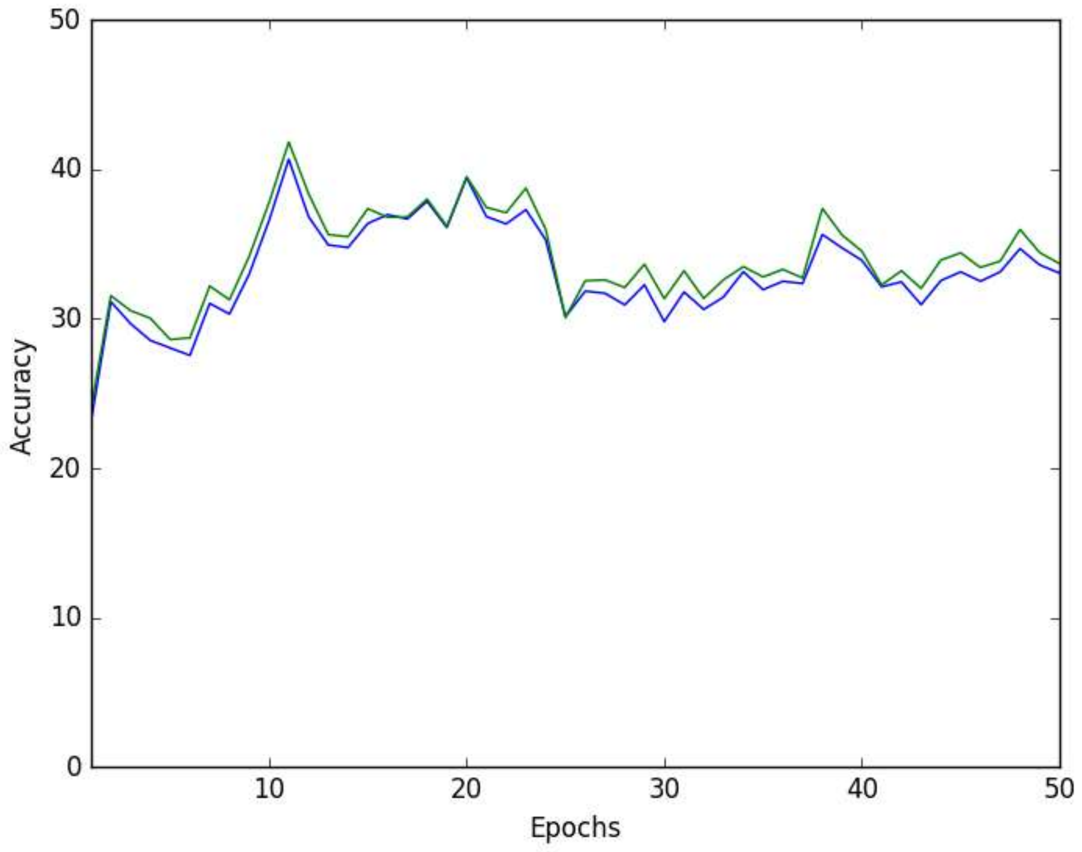
Matthew O’Brien

CS445

HW2 Neural Net Report

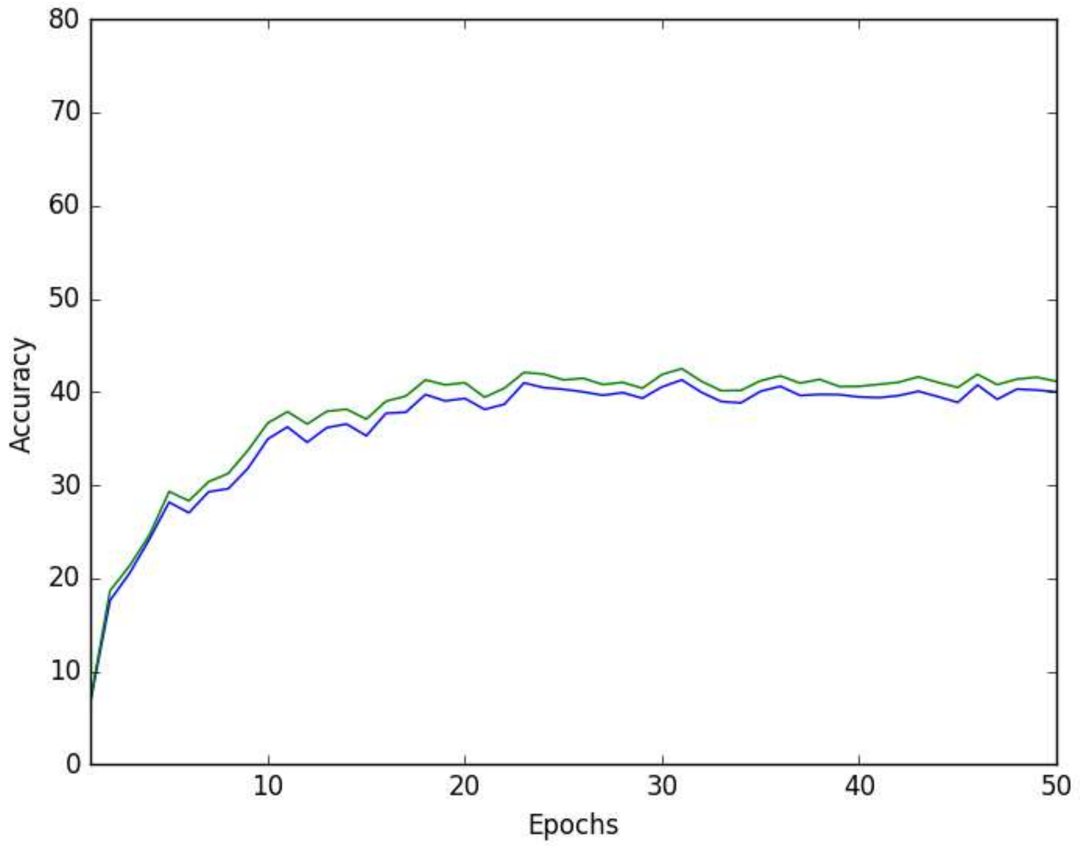
The very first thing worth mentioning for this assignment was the less than ideal way of approaching it. It was mentioned by the TA that matrix multiplication would greatly improve the performance of the net. I was not really able to wrap my brain around the matrix multiplication, without completely starting over. So due to time constraints and other issues, I used the past homework assignment implementation and just built on it for this assignment. I kept with classes of perceptron nodes with were stored within lists. The majority of the implementation has to do with looping through many different lists. Which is obviously a large hit to the speed of the net. Out of random curiosity, I ran my program over night, and was only able to get through roughly 500 epochs with a large number of hidden nodes in eight hours. Though, this wasn’t explicitly asked for in the report, I just wanted to see what would happen.

The first test had a learning rate of 0.3, momentum of 0.3, and hidden nodes of 4. The following graph was created through a basic graphing tool within the python libraries. The green line corresponds to the training data set, and the blue line to the test data set. The test was run for 50 epochs.



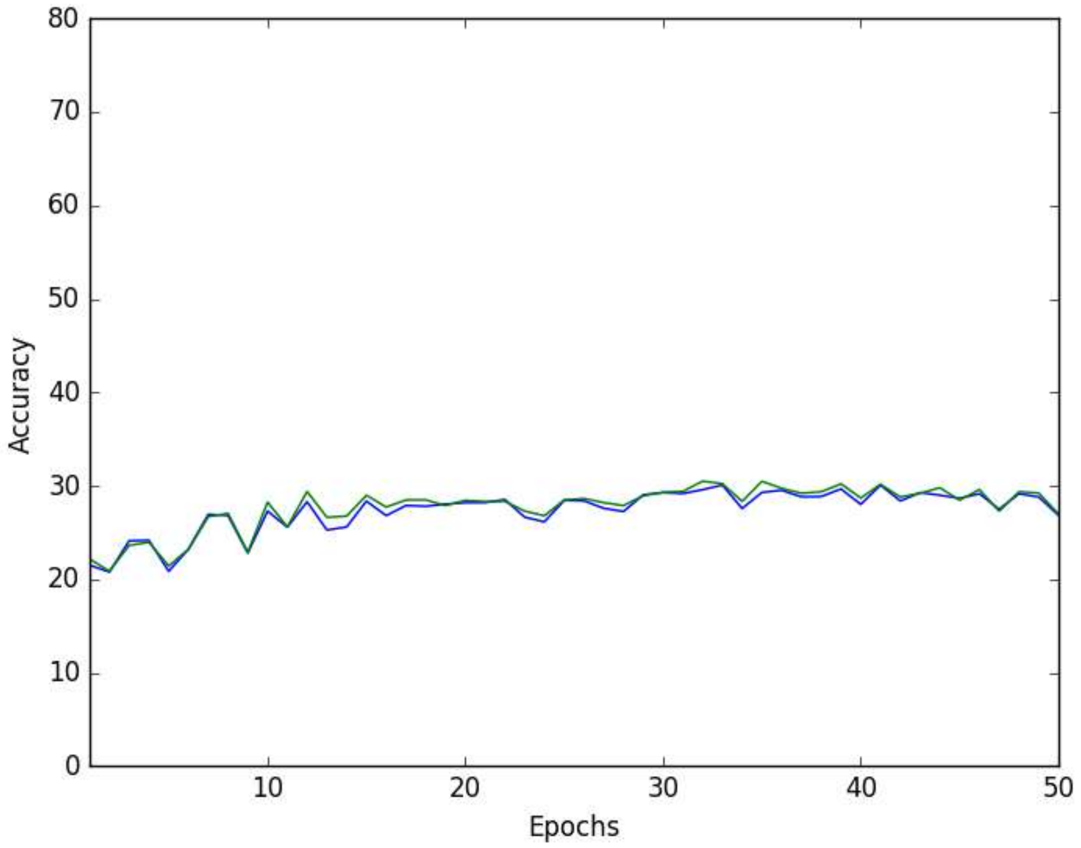
The first thing worth noting, is that the training data never actually reaches 100% accuracy. Further testing was done, at higher epochs, but the accuracies of the data sets never improved much beyond the data shown in the graph. There is some very obvious issues with this data set. The accuracy is flat-lining at roughly 35%. I would say that the parameters of this test simply reached a sort of local maximum, and were unable to push passed it with the given values. But I'm not entirely sure as to how or why this is happening. More extensive tests would definitely help. Possibly being able to run for thousands of epochs. With this implementation would take hours, so I did not do that.

Test two has two different parameters. The first parameter of a lower learning rate, or 0.05. Again the following graph shows the training data as green, and the testing data as blue.



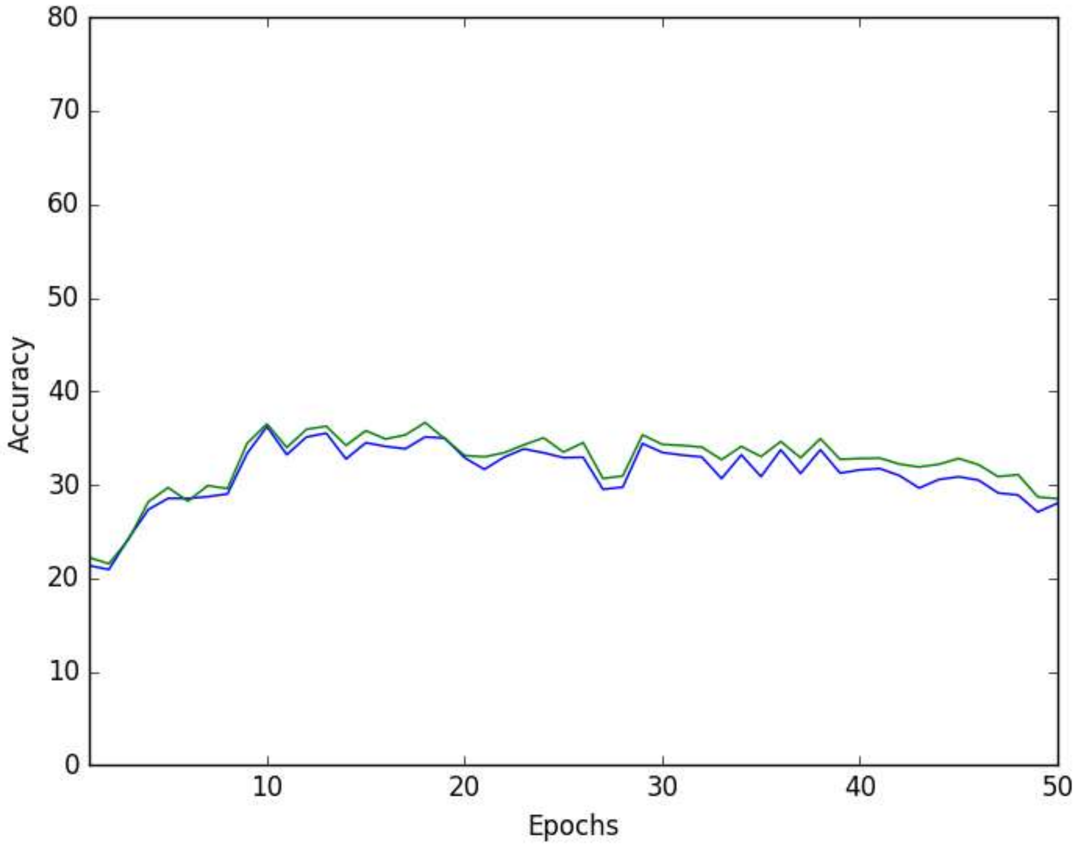
There is again a very clear flat line point for this test. It is higher than the previous test. Roughly stopping at 40% accuracy. Again, I'm not entirely sure as to how or why this is happening. I believe it's an issue of a local max for these parameters. These parameters were able to push beyond the previous situation, but still got caught. Or it is possible that the model is overfitting with the training data, and is making too drastic of changes and is unable to predict the letters with smaller variations.

The second part of test 2 has a higher learning rate of 0.6. This data set has the following graph. Green is the training, and blue is the test.



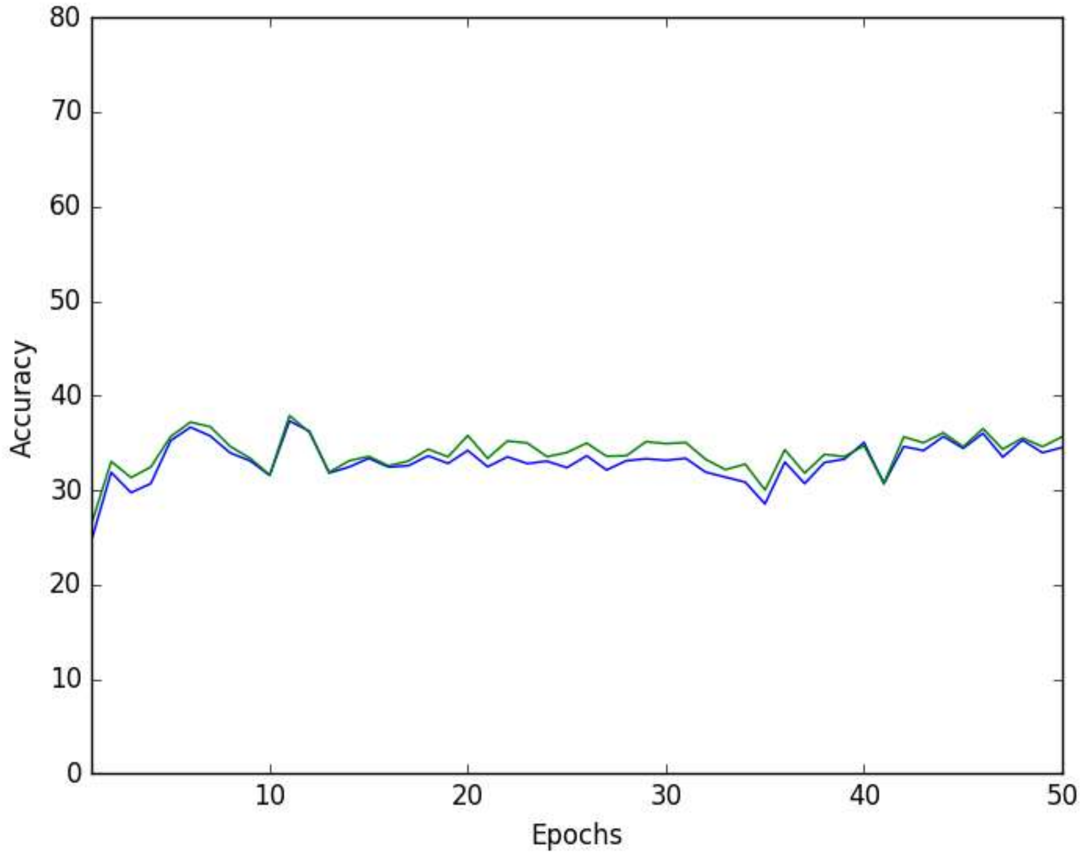
This test is clearly much worse than the previous test set. The accuracy was never able to pass beyond 30%. I still believe it's the same issue as the previous tests. But more specifically, the model was never able to get beyond this percentage because of either overfitting with the training data or a local max, and it simply oscillates in that curve. Though the momentum is designed to stop that, but the smaller value may not be enough to push beyond it.

Test three was a variation of momentum rate. This first was the lower momentum rate of 0.05. With how my code was set up, I was unable to graph them together easily. Green line for training, and blue for testing.



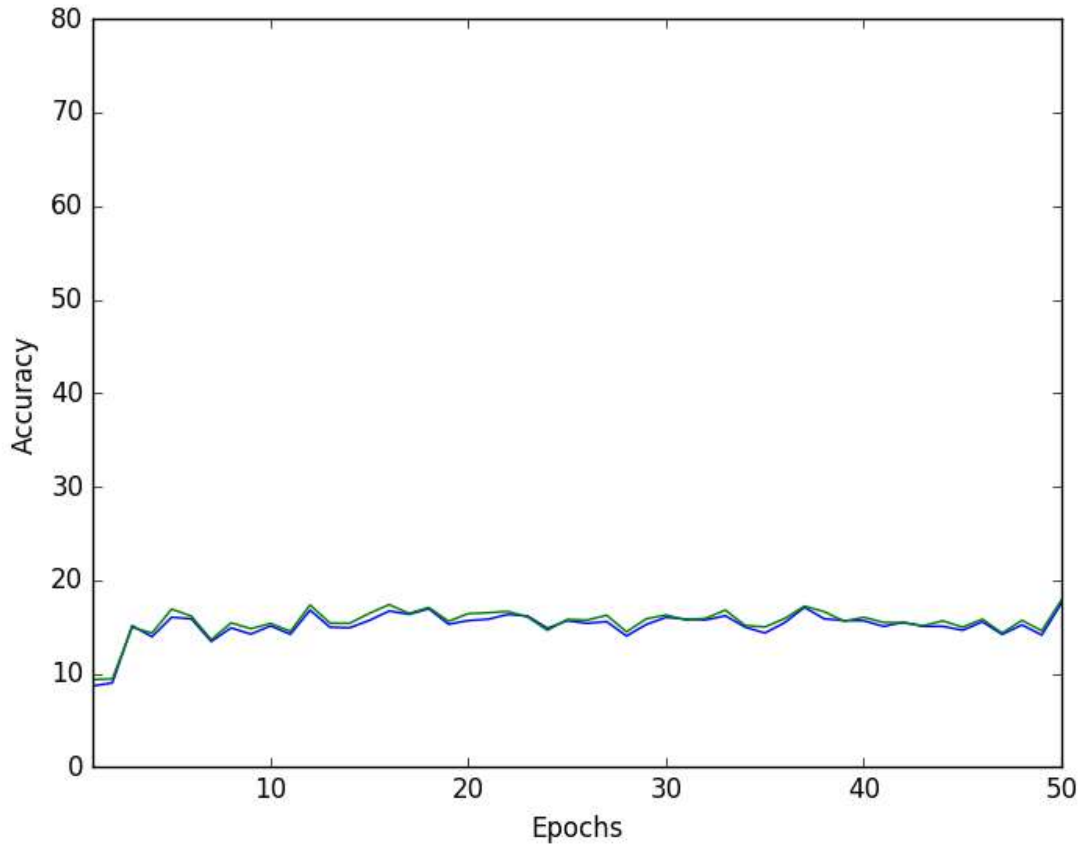
This round of testing was much less ideal. I ran it a few times to see if the data set would come out any different. But it seems to dip down after roughly 35 epochs every time. The data would oscillated around 30% accuracy for the majority of the outputs. I believe this is because the momentum value is just too small, and was unable to keep the weights moving in the necessary directions. The lack of any real help from the momentum caused the network to be caught with the same situations as the previous tests. The very interesting points around between 30 and 40, when the graph actually shows the accuracies oscillating very clearly. This is why I believe the net is caught within a local max and the lower momentum isn't allowing it to escape.

The second part of test 3 is the larger momentum value of 0.6. The green lines are training, and blue for test.



Interestingly enough, the larger momentum rate did not seem to help much at all. Which sort of ruins the previous theory of the smaller momentum getting stuck within a local max. There is also the possibility of it just oscillating from one side of the curve to the other over and over again. Again, it’s very difficult to speculate exactly what is going on with this situation. But it seems like it is just oscillating on a local max.

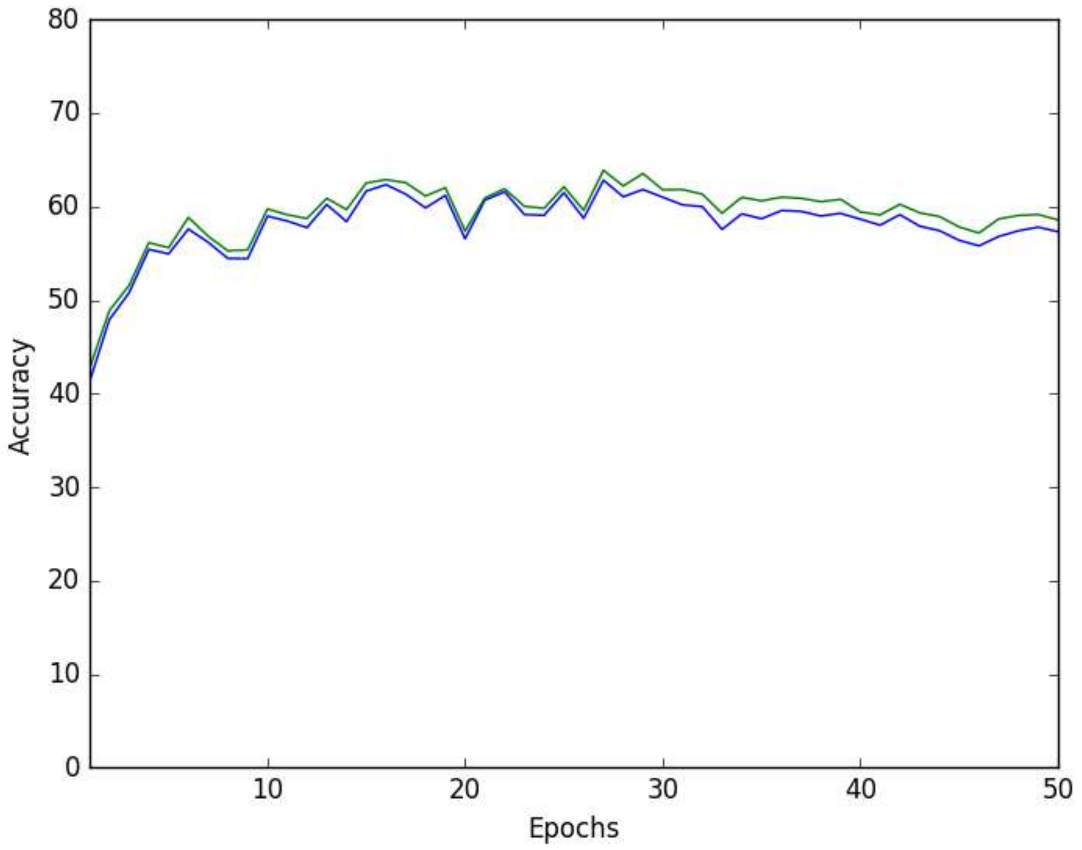
The last test was altering the amount of hidden nodes. The first with a low number of 2 hidden. With my set up, I was unable to graph them together. This test had the worst accuracy of all of the tests thus far. Green for training, and blue for test.



The theory for the poor performance on this test is just a lack of complexity. The net is too simple to actually be able to fully predict the outcome of the letters. There is not enough to actually vary or change without removing the needed portions for other letters. I ran this variation of the network for a long time again, and it was never able to reach beyond 20%. I'm not sure if that's just my poor implementation or my theory about the lack of complexity.

The last variation of the tests is the larger hidden nodes. This test actually performed the best out of all of the tests. The green is for training, and blue for test data.

The graph shows the highest accuracy of any of the tests done within this report. Though, again I find that the accuracy will flat line at some point. This time being around 60%. Clearly much higher than the previous tests. But I just do not understand as to why the net seems to always hit a wall. I suppose it could be due to overfitting the training data, and just not being able to predict the testing. But both of them are having issues. In my implementation, I shuffled the training data after each epoch. As this improved the testing accuracy. But it would be interesting to see what would happen with the training data, if nothing was shuffled.



I did a few extra tests of my own after the fact, out of curiosity. I found that very large number of hidden nodes, upwards of 20 allowed for much greater accuracy. But with my implementation, it was time consuming to test. So I was only able to do a few random tests. I can't help but feel like there had to have been another issue with my coding or implementation that caused these issues. Though, because of my implementation it was difficult to tell if time would solve the problems. Simply because of the amount of time each test would take to complete.