# CS 478

Scott Leland Crossen

## Backpropagation Project

### Part 1

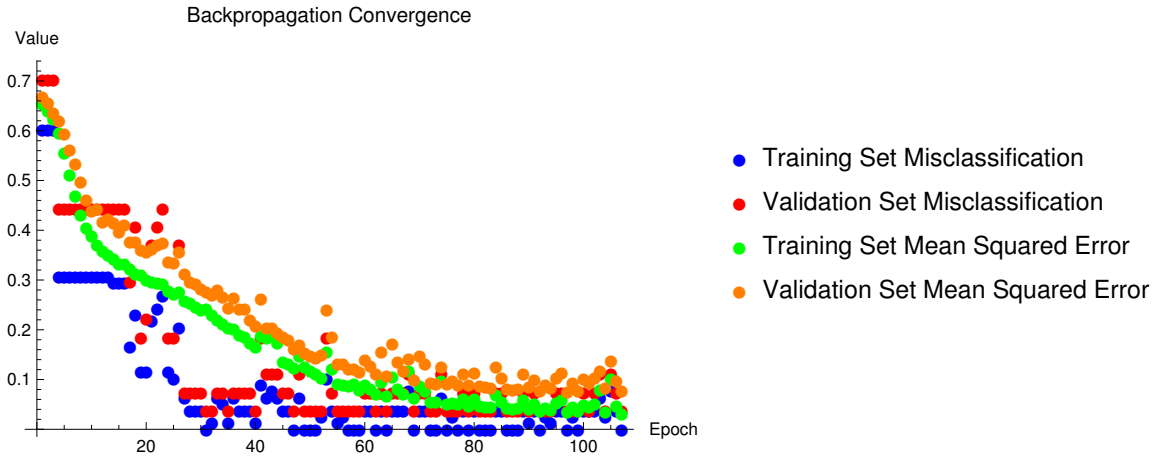
The code that implements the backpropagation rule is included with the submission of this project. It is written in Scala and follows the functional paradigm. Immutability is observed in all but the underlying learner.

### Part 2

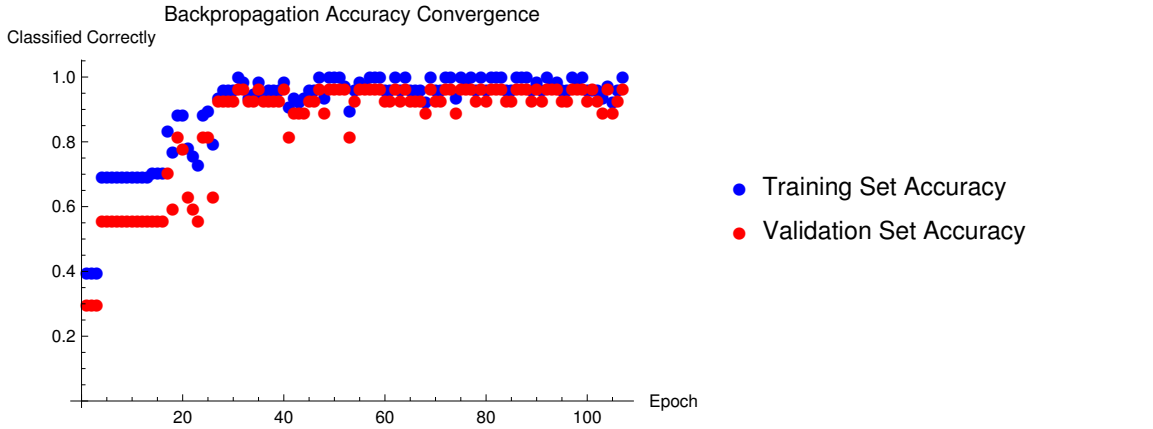
In order to show the full shape of the graph, the stopping criteria for this part was defined to be 10 successive epochs without improvement beyond the minimum mean squared error. In this case, the stopping criteria was met after 106 epochs (a total of .35 seconds). The final accuracy for this 75/25 training/testing split was 0.9904761904761905 on the training set and 0.9333333333333333 on the testing set.

It is noteworthy to point out how the data shows a discrete layer-nature in the accuracy plots. I attribute this to two main reason. The first is machine precision. The calculation for accuracy is computed over a fixed number of rows for which the output of each is either a “1” (classified correctly) or “0” (classified incorrectly). Because of the discrete and limited nature of the accuracy calculation it is very likely that discrete features will be seen in the graphical representation of the data. Moreover, the accuracy is an emergent behavior of the various weights that compose the neural net. At some point the neural net modifies it’s weights to overcome a threshold where it starts to classify things correctly. The speed for which this threshold is met is determined by the learning rate. At this point the algorithm stops classifying things with random probability and starts classifying correctly. This threshold is another possible reason for the discrete nature of the displayed accuracy plots.

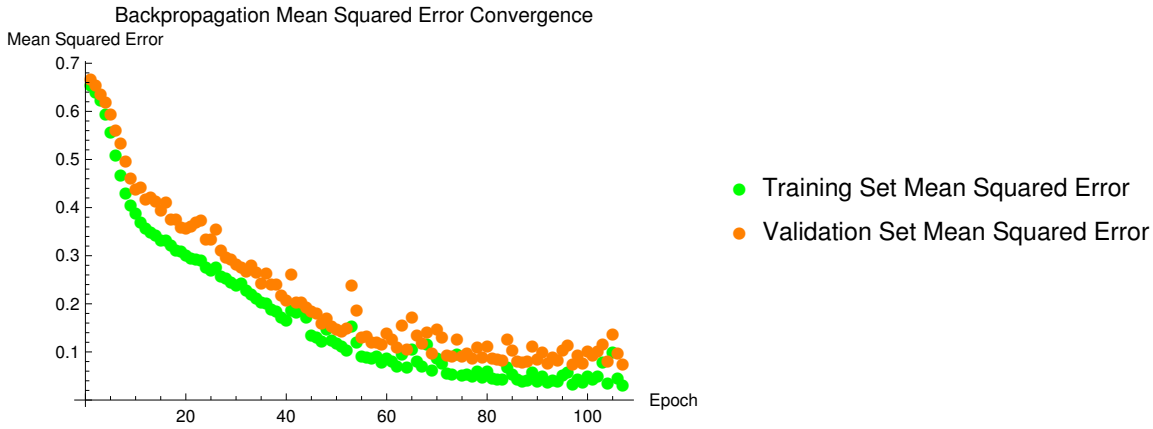
Out[362]=




Out[363]=




Out[364]=




## Part 3

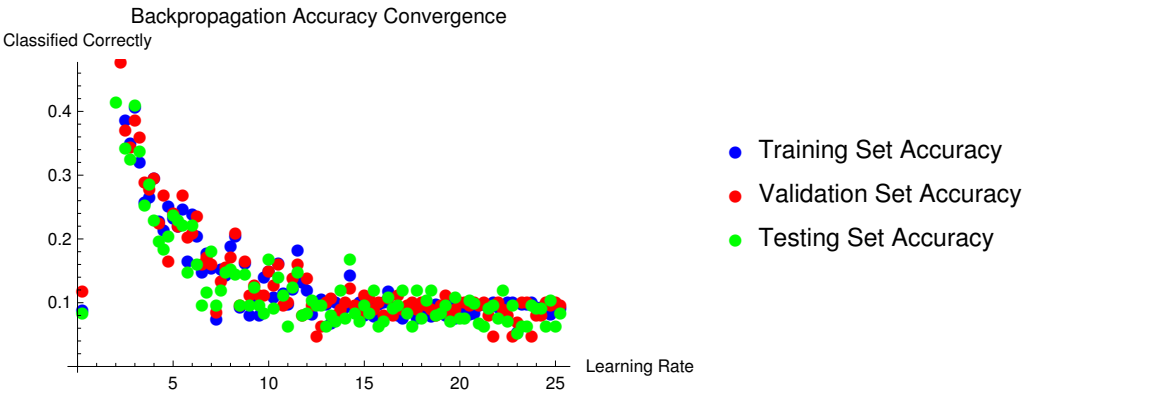
To understand a good baseline, I first ran the vowels data set on the “baseline learner” and the perceptron. the reported accuracy for these two algorithms were 0.068 and 0.25 respectively. Out of the box, the backpropagation algorithm had a test-set accuracy of 0.72 which is significantly better.

Upon looking at the data set it is obvious that many of the features are redundant or extraneous. For example, whenever the name “Andrew” appears, it is followed with the fact that he is a man. The same

goes with every other name included in the file. The third column therefore is not needed. Moreover, the first column is also an extraneous features. Names included in “Train” are not included in “Test” and vice-versa. The second column is the only column of the first three that actually holds all information by itself. The first and second are not needed. If these columns were included it would be likely that neural net would put their weights effectively to zero anyway.

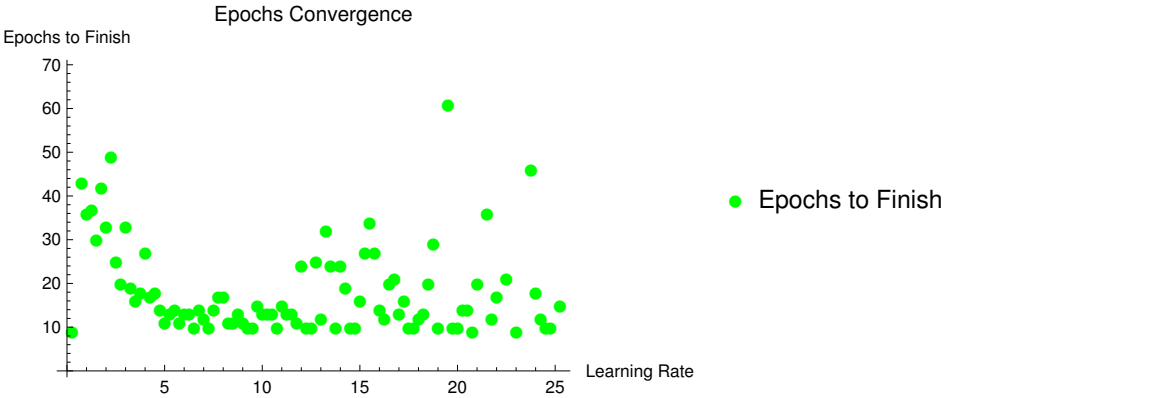
Below are the plots showing convergence vs learning rate. As shown, all metrics converge at around a learning rate of ten. I attribute this to the fact that learning rates above this value adhere to a law of diminishing returns that make increase in learning rate less viable.

Out[437]=




Out[438]=

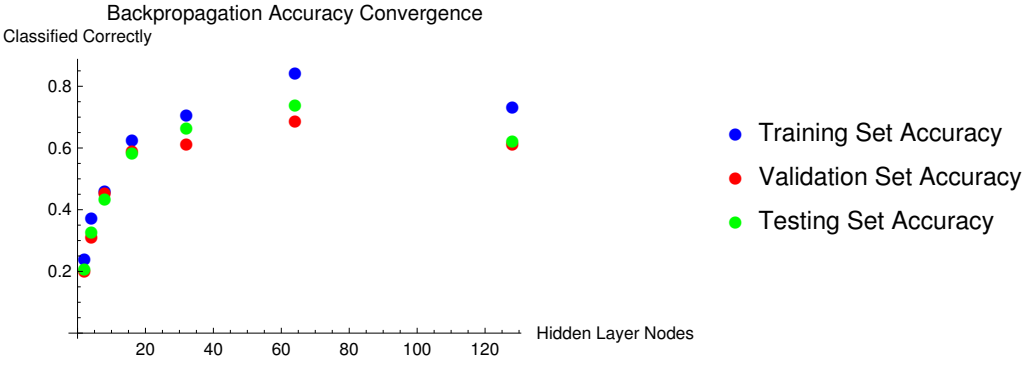

Out[439]=



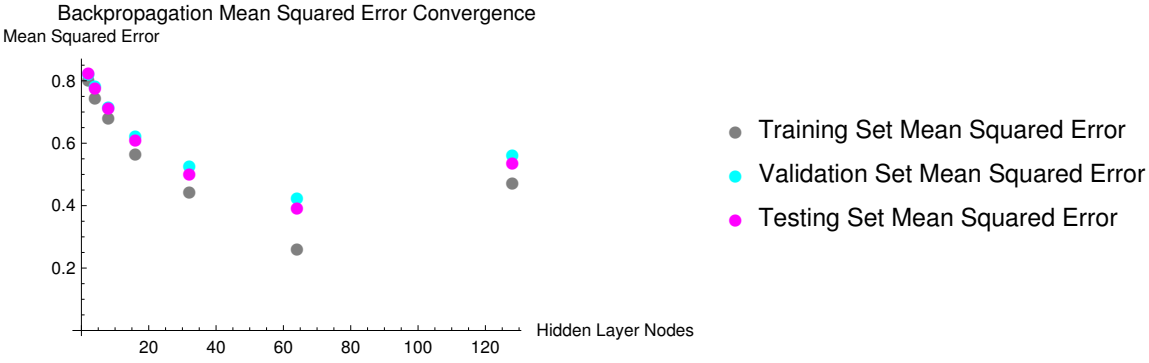
## Part 4

Below are plots showing the optimal amount of nodes in the hidden layer. As shown, the optimal number seems to be around 80. Interestingly, the number of epochs required to finish are very sporadic but show much the same thing: that there a local maximum/minimum around 80 hidden layer nodes. The likely cause of the decrease in required epochs at 128 hidden-layer nodes is likely because of the random-nature of the starting criteria. Each node has randomized weights and the more nodes that are initialized the more likely it is that some of those weights correspond to the natural weight of the system. After a few epochs, these weights will start to be favored by the system.

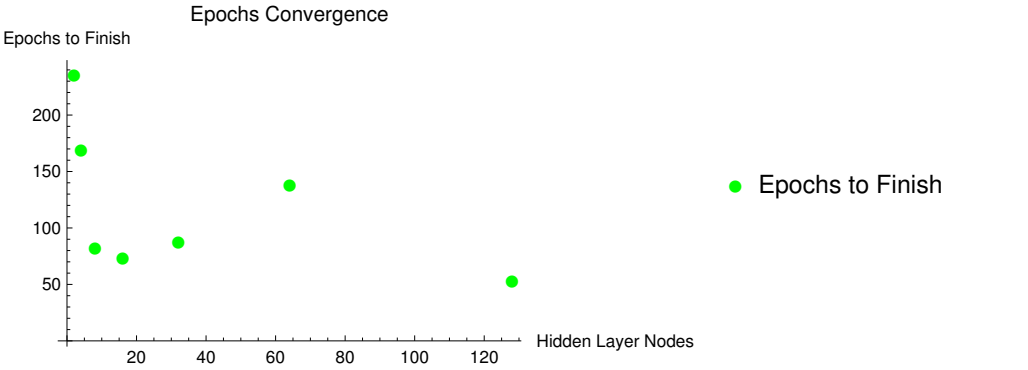
Out[471]=




Out[472]=




Out[473]=

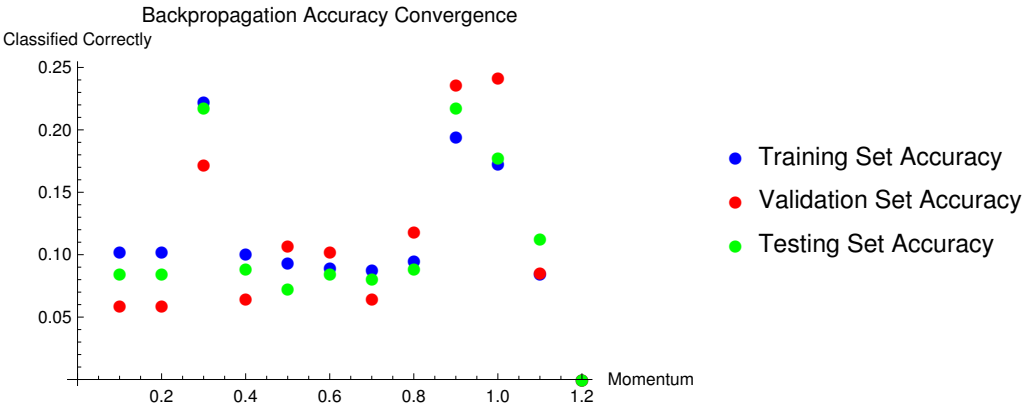



## Part 5

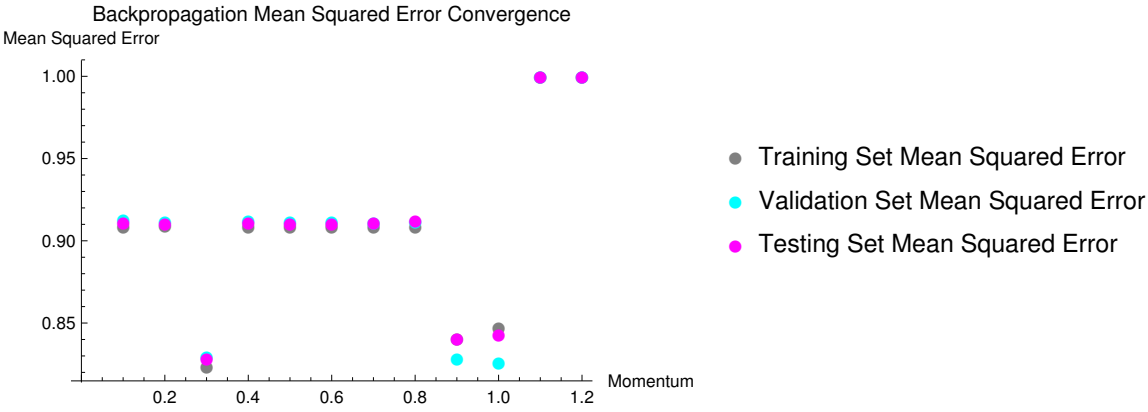
Below shows the momentum value as it changes between the values of 0 and 1.2. As shown, there is not much difference and all the variation is likely just due to stochastic reasons. I tested values greater

than 1.2 but all of these values required more than 1000 epochs to converge to the data (if they converged at all). It threw off the plots and so they are not included here. It is surprising that such a large jump occurs in the convergence of the algorithm between the values of 1.2 and 1.4 (where the convergence blew up).

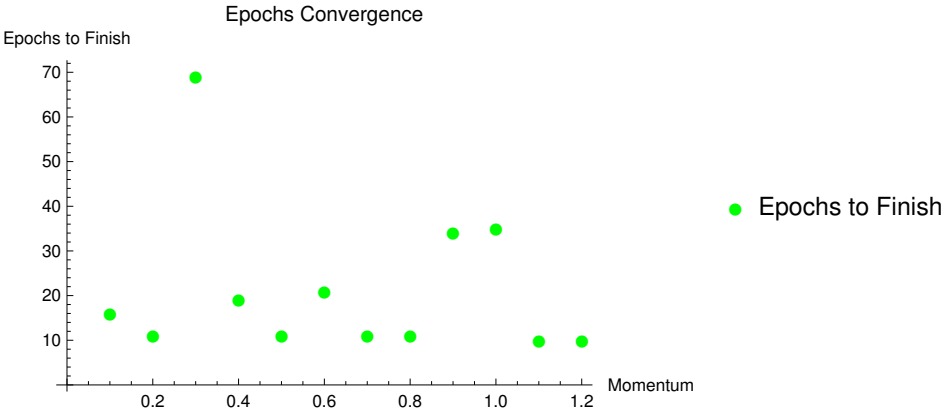
Out[448]=




Out[449]=




Out[450]=

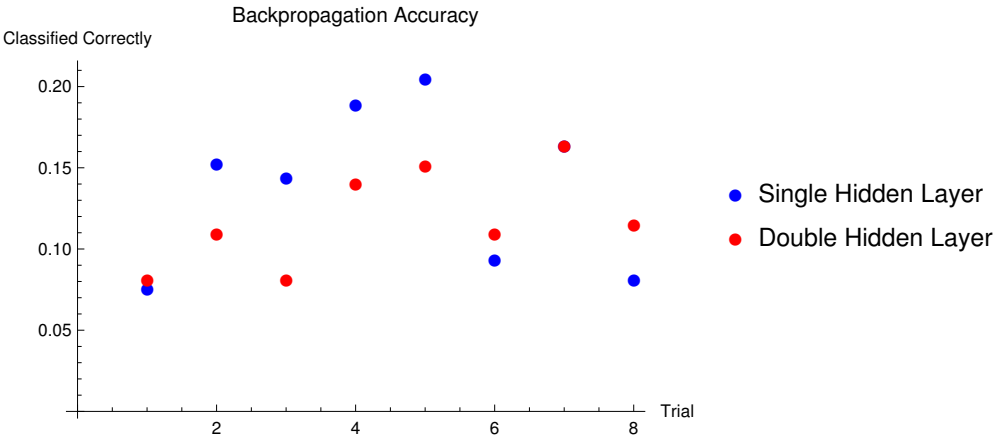



## Part 6

For my project I attempted to add another hidden layer to the neural net with nodes updated in much the same way as the other algorithm. I tested it on the same file as before (vowels) and found a very similar convergence. Essentially, adding a second layer was no more effective than adding the equivalent number of nodes to the single-layer model. Below are the convergence statistics for both algo-

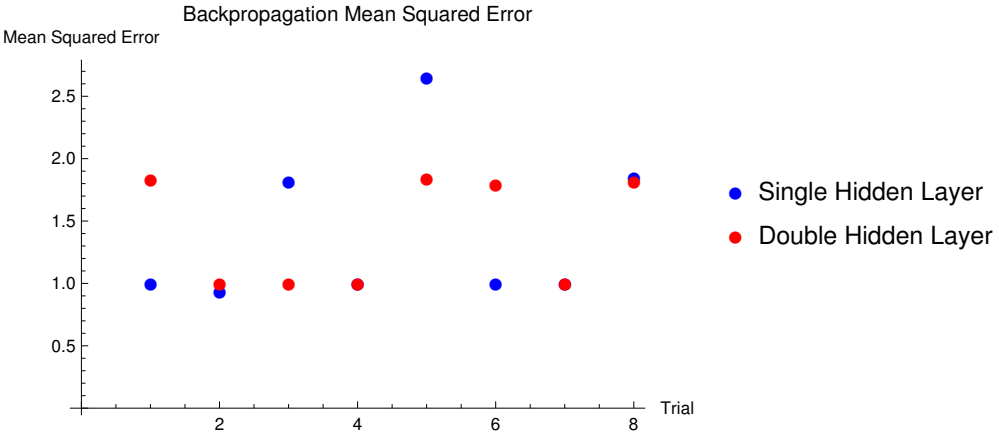
rithms. As always, each algorithm used a 75/25 training/testing data split with a stopping criteria that would trigger when 10 successive epochs were presented with no improvement above the BSSF.

Out[495]=



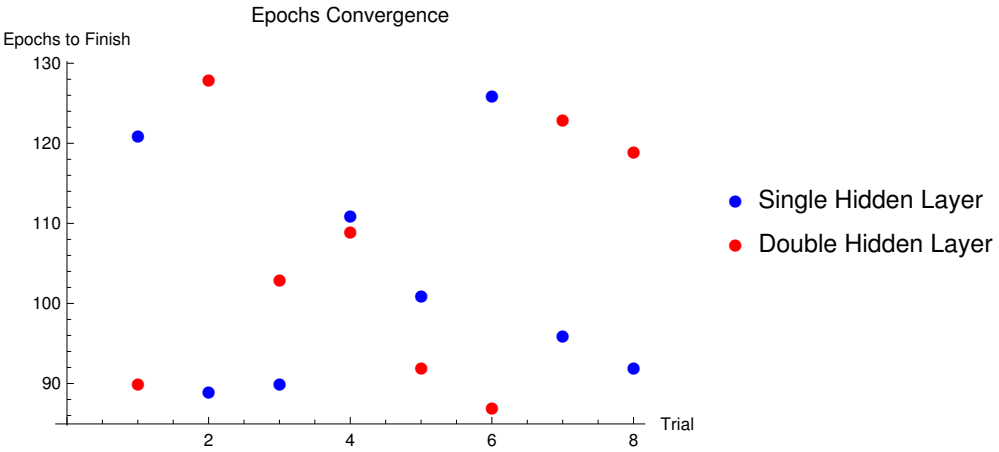

```
The average accurracy for the algorithm with one layer was 0.13804
The average accurracy for the algorithm with two layers was 0.11893
```

Out[498]=




```
The average mean squared error for the algorithm with one layer was 1.40405
The average mean squared error for the algorithm with two layers was 1.409
```

Out[501]=



The average number of epochs to finish for the algorithm with one layer was 103.25

The average number of epochs to finish for the algorithm with two layers was 106.375