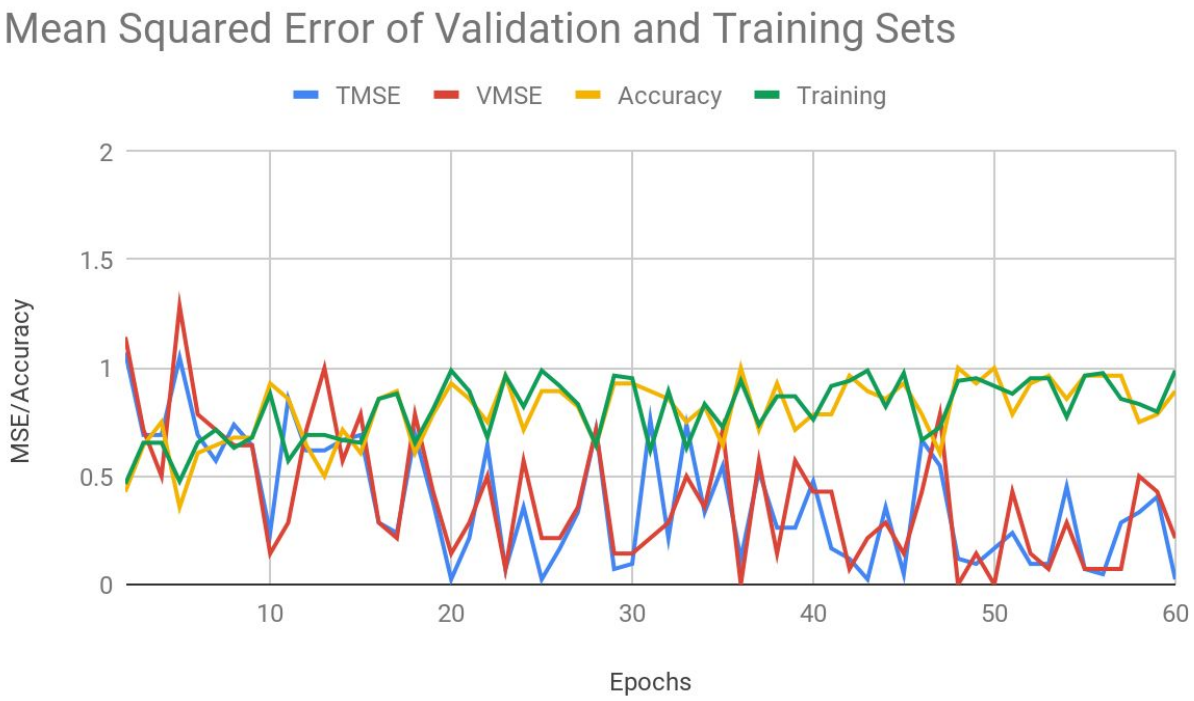
Backpropagation

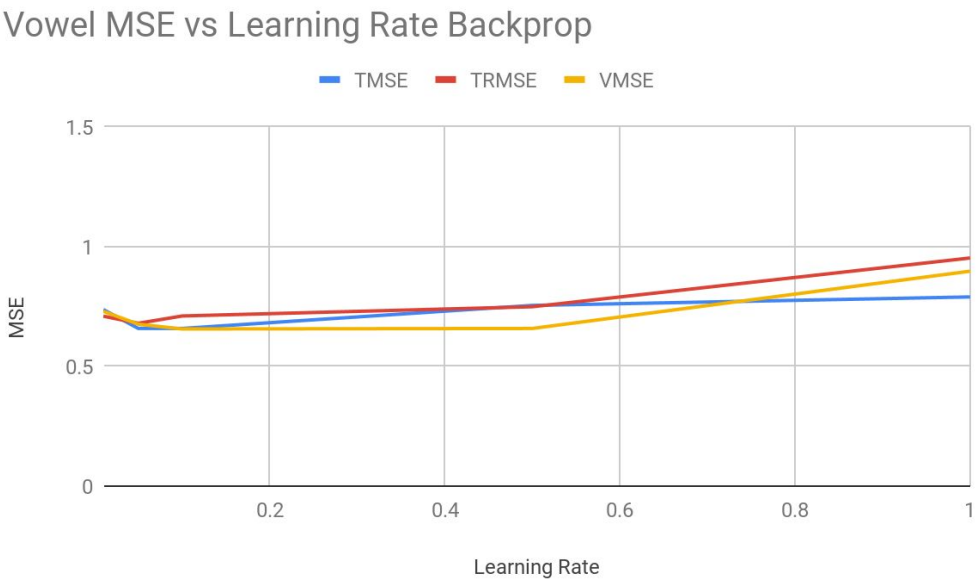
1. I correctly implemented the Backpropagation Algorithm with at least one hidden layer and an arbitrary number of nodes, random weight initialization between -.5 and .5, stochastic weight updates, a stopping criteria defined below using MSE, randomized training sets and an option to add in momentum. My code can be found on Learning Suite under the code submission. The file name is multilayer_perceptron.py

2. On the iris set is used a 75/25 split between my data for training and testing as well as I took 25% as my validation set and used that to find my stopping criteria. Specifically, in the graph below I compare the validation set to the training set's Mean Squared Error.

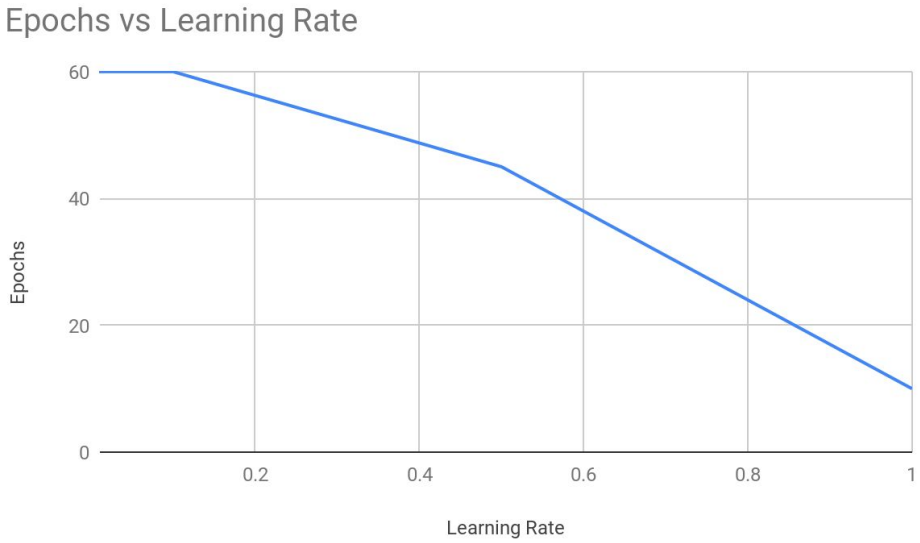


Graph of MSE of the Training Set and Validation set in the y-axis and the number of epochs on the x-axis. This graph is more noisy than would be expected, but you can see the increase in accuracy over the number of epochs and the decrease in MSE of both the Training MSE and Validation MSE, and that both level out around 60 epochs

3. I used the Backpropagation Algorithm on the vowel dataset and was able to get %60 percent accuracy. I used one layer of hidden nodes and used twice the number of hidden nodes as the input nodes. I did a random split of 75/25 to training/testing and then took another 25% for validation. See graph descriptions for more details on changes in performance based on # of hidden nodes, epochs, momentum and MSE.

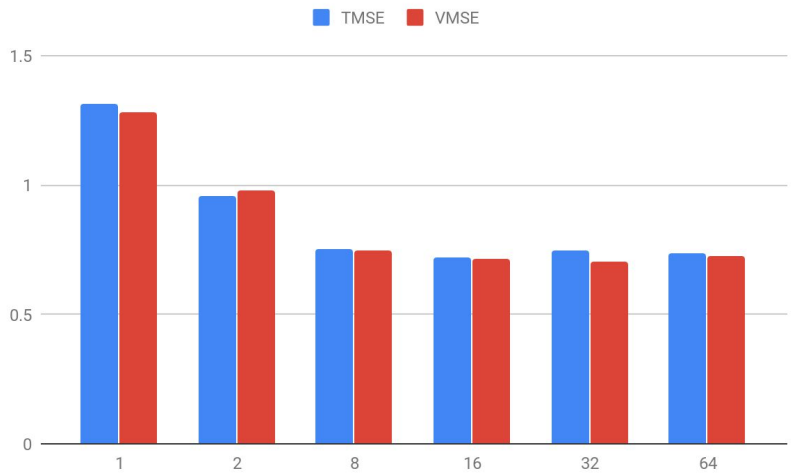


This graph shows the MSE for the training set, the validation set, and test set, at the the stopping criteria of 60 epochs. These results were not what I had expected. I thought that as the learning rate goes up, the amount of error would go down, but the graph clearly shows that the larger the learning rate, there's more error.



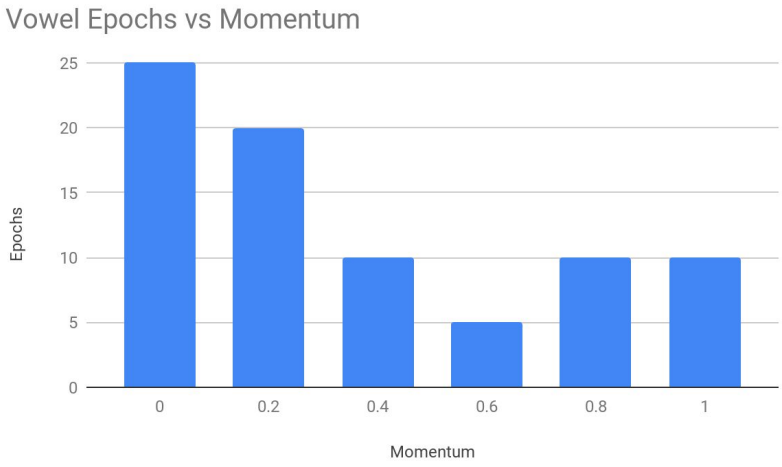
This graph shows the number of epochs needed to get to the best validation set solution. Again I thought it would decrease more at the beginning and then level off, there may be an issue with the number of epochs that were used as the window for the best so far MSE. This may also be due to the fact that my accuracy and MSE are much more noisy than expected and therefore the stopping criteria isn't as effective.

4. Using the best learning rate of .1, I experimented with hidden node sizes of 1, 2, 4, 8, 16, 32, 64. Note the graph below and the description.



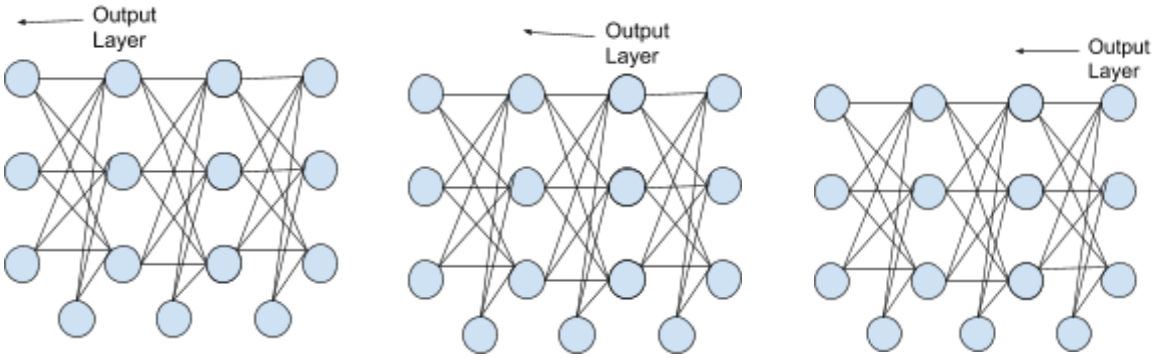
According the graph, an increase in hidden nodes, decreased both the Training MSE and the Validation MSE, but only to a point. After about 16 nodes, the graph levels out. This may change as the number of hidden layers increases.

5. Using the number 16 as best number of hidden nodes, I experimented with different momentum sizes and the corresponding number of Epochs. Note the graph below and the description



The purpose of this graph is to see how much momentum speeds up learning. Notice how the number of epochs required decreases as amount of momentum increases. One reason why I think my algorithm may hit my stopping criteria faster than others is the noisiness of the MSE.

6. If you've ever run ladders, it's the worst. You have to run to the 10 yard line, then back, then the 20 then back, then the 30 etc. I wanted to see if I could use back prop and forward propagate one layer, and then go back, then 2 layers, then go back, then 3 layers and then go back until I hit the output node. One obvious problem I ran into was the fact that you need the error of the next layer in order to calculate the error of the layer you're in. Also, unless you hit the output layer, you can't do target - output. But, to get around this or at least to try and get it to work. I made the requirement that the number of output nodes had to match the number of hidden nodes in each layer and the number of input notes. So, the NN would have an equal number of nodes. In each layer. Then, I treated the 'ladder' I was on as the output node. So if there were 5 layers. I treated the first as the output layer, then the second as the output layer then the third etc. This also solved the problem of needing the error.



I created these chartes do better show how the process would work. Each hidden layer is treated as the output layer and then is propagated backward to change the weights. My code can be seen under the filename multilayer_perceptron_tournament.py

My MSE for this was .77 and my accuracy was either .66 or .0 and that may be because this wasn't a good idea or maybe I made an error in the code. All in all this wasn't effective.