Sam Chamberlin
CS 472 – Machine Learning
Dr. Martinez
October 13, 2020

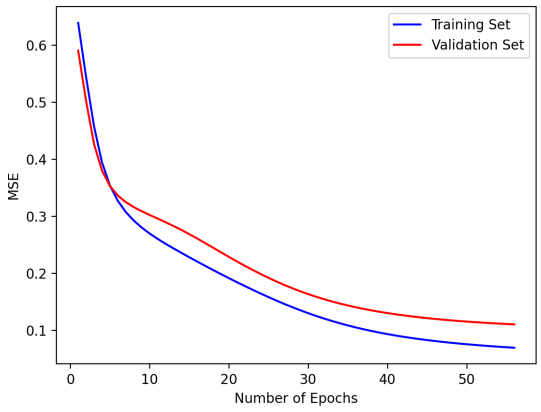
Back Propagation Lab

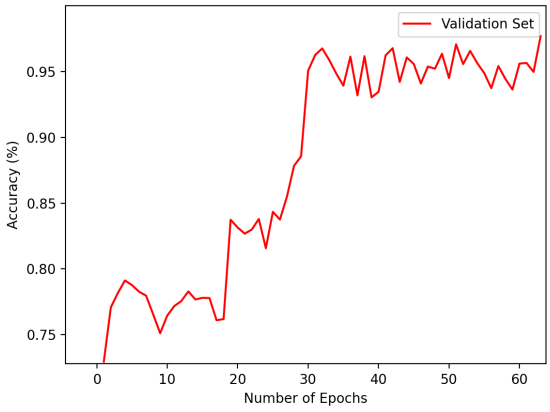
Back Propagation Learning Algorithm

A multi-layer perceptron class implementing back propagation was written with features including user ability to create a network structure with at least one hidden layer and an arbitrary number of nodes, random weight initialization and inter-epoch shuffling. A momentum term is included to allow for quicker learning and a validation set is used as stopping criteria. Weights are updated stochastically, after each training instance. The weights found for the evaluation data supplied is reported at the end of the report.

Iris Classification

The machine learning model was trained on the Iris data set from UCI. I stopped learning when the Mean Squared Error (MSE) of the Validation Set (VS) stopped improving for significantly for at least 10 epochs. I kept track of the best MSE for the VS so that once performance started to suffer, I could return to the weights of when the model was performing its best on the VS.





Vowel Dataset

Considering how typical accuracies for the Vowel data set are around 75%, this set definitely poses more significant challenges than that of the Iris data set. First off, there are many more features in the Vowel data set. As discussed in class, a set with more features than it needs leads to overfitting of data trying to capture insignificant correlations. Another difficulty of

increasing features is that need for more data rises exponentially. Thus, if you have a lot of features and not enough data to fully capture significant relationships, then the machine learning model won't be able to learn. Ultimately, features need to be reduced to avoid a "garbage in, garbage out" situation. A similar approach is taken to this data set as before with the Iris data set, however, feature selection is performed first.

To avoid node saturation, I decided to get rid of the features with minimal variety. These features included speaker number and gender. These features will hopefully be represented well through by the other real-valued features. As for the remaining features without titles, I kept them all, unsure of which I could drop. I quickly noticed that running my model on this data set took significantly longer, and even after 10-15 minutes it didn't complete. In order to see some results quicker, I limited the number of epochs to 10. I began by carrying the learning rate value to see how it influenced the model's ability to converge quickly on an MSE value. Below are the results:

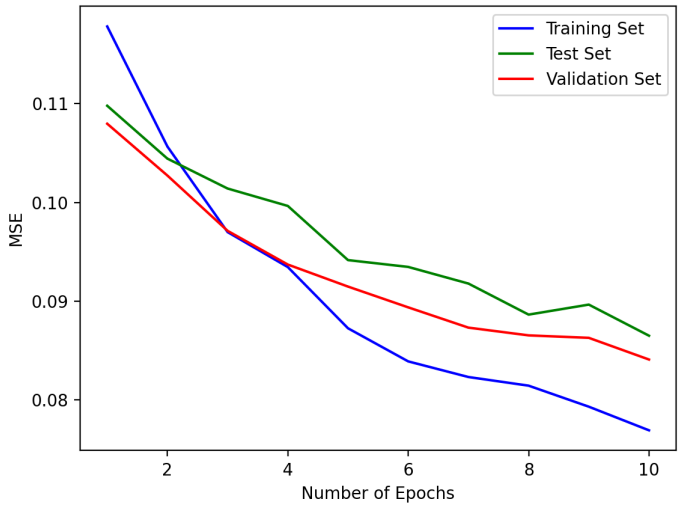


Figure 3: Learning Rate = 0.1 & Momentum = 0 for vowel data set MSE

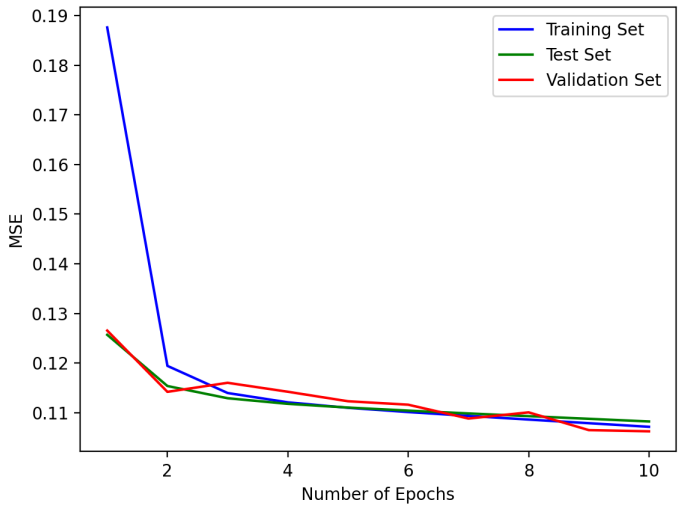


Figure 4: Learning Rate = 0.01 & Momentum = 0 for vowel data set MSE

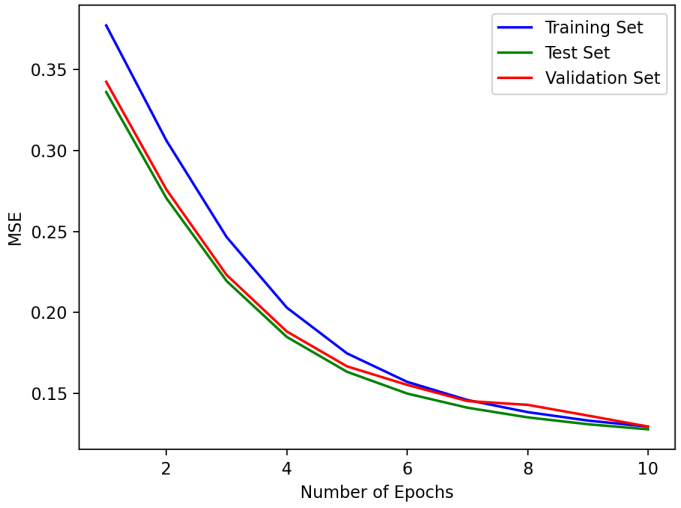


Figure 5: Learning Rate = 0.001 & Momentum = 0 for vowel data set MSE

Varying Number of Hidden Nodes

From the results above, I decided to continue implementing a learning rate of 0.001 as it seemed to converge both quickly and smoothly. Next, I varied the number of hidden nodes used in the model to see what effect that had. From the figures below, it's evident that increasing the hidden nodes up to 32 nodes benefits accuracy in my model, but it seemed to struggle past that.

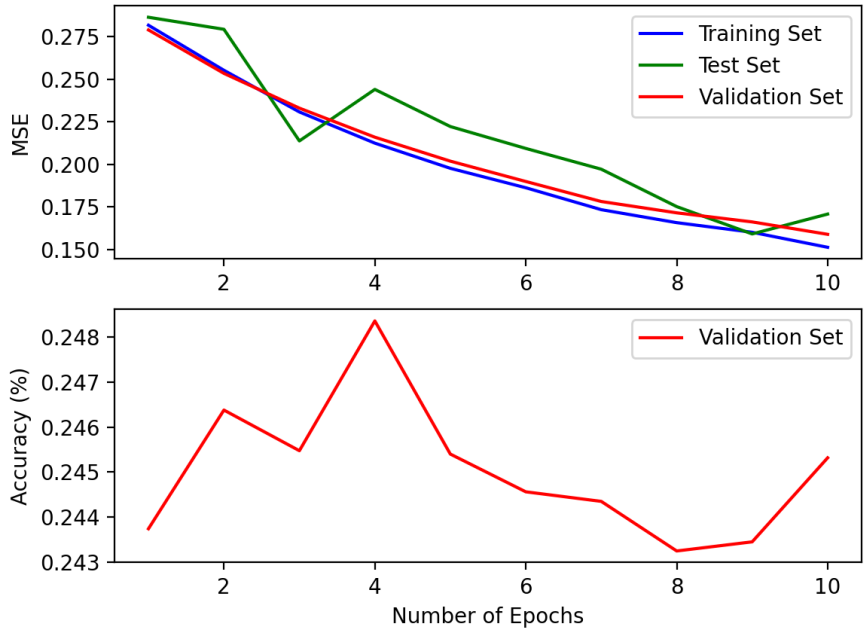


Figure 6: Learning Rate = 0.001 & 1 Hidden Node for vowel data set

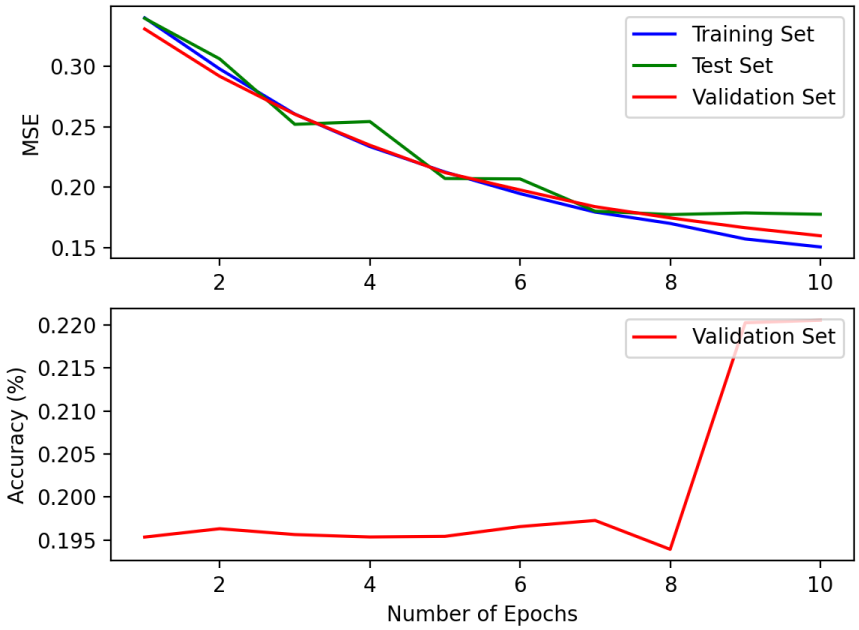


Figure 7: Learning Rate = 0.001 & 2 Hidden Nodes for vowel data set

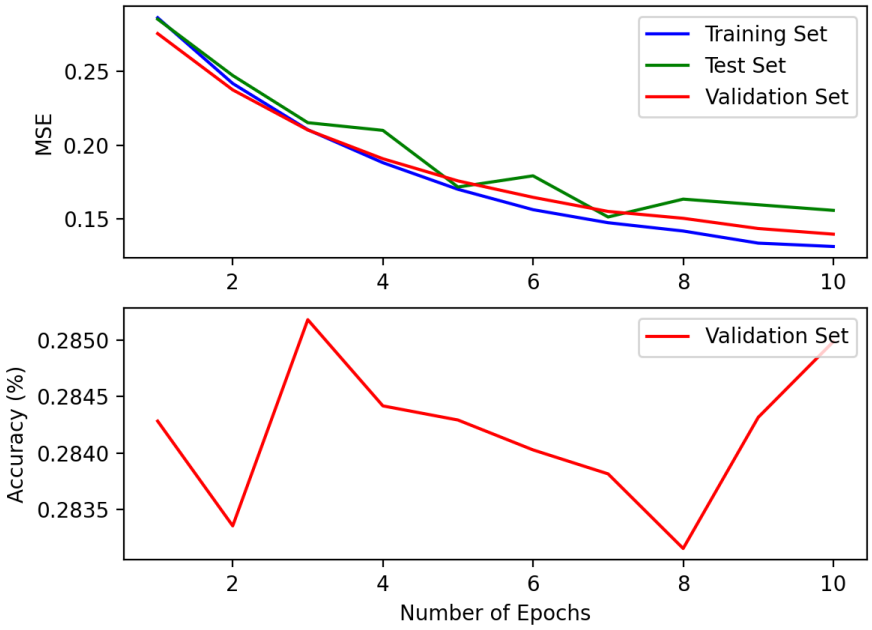


Figure 8: Learning Rate = 0.001 & 4 Hidden Nodes for vowel data set

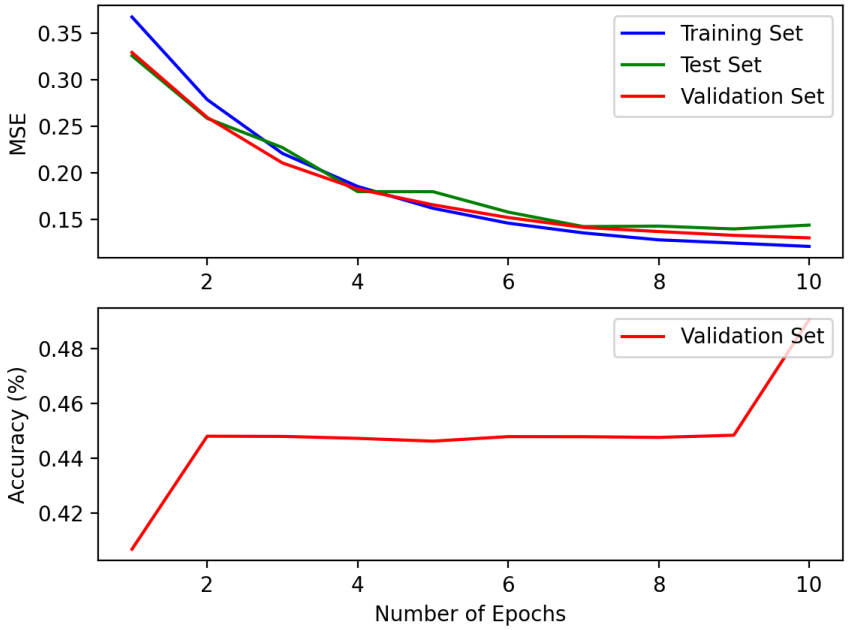


Figure 9: Learning Rate = 0.001 & 8 Hidden Nodes for vowel data set

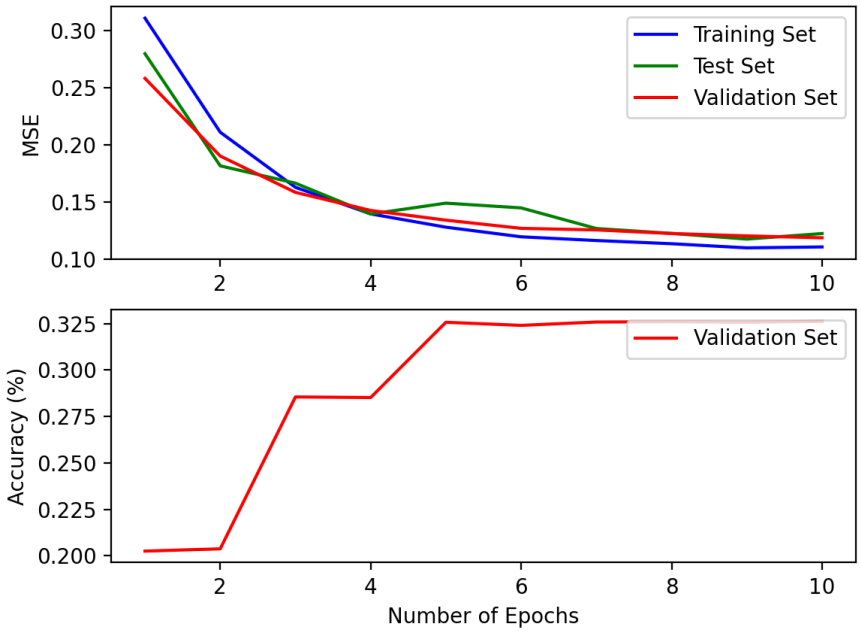


Figure 10: Learning Rate = 0.001 & 16 Hidden Nodes for vowel data set

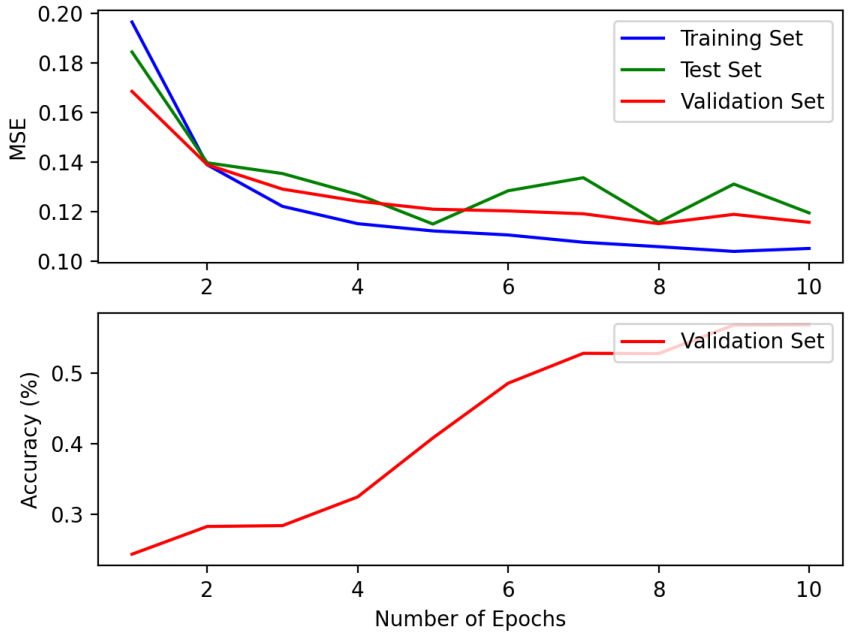


Figure 11: Learning Rate = 0.001 & 32 Hidden Nodes for vowel data set

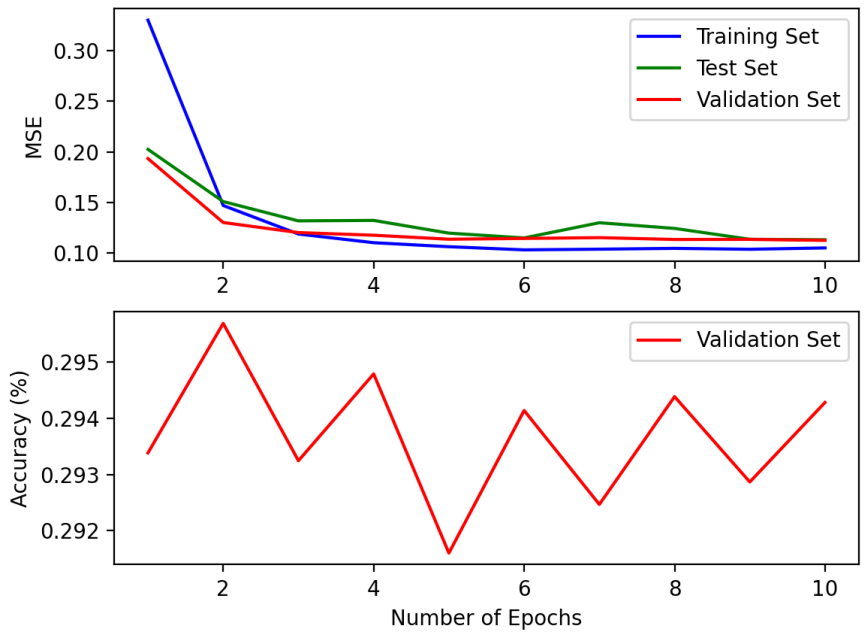


Figure 12: Learning Rate = 0.001 & 64 Hidden Nodes for vowel data set

Momentum Values

Lastly, with the learning rate set to 0.001 and the hidden nodes set to 32, I played with a few different momentum values. It was difficult to see major differences between the momentum values, but one common thread seemed to be that with an increased momentum value, MSE leveled out quicker but accuracy rose at a comparable rate.

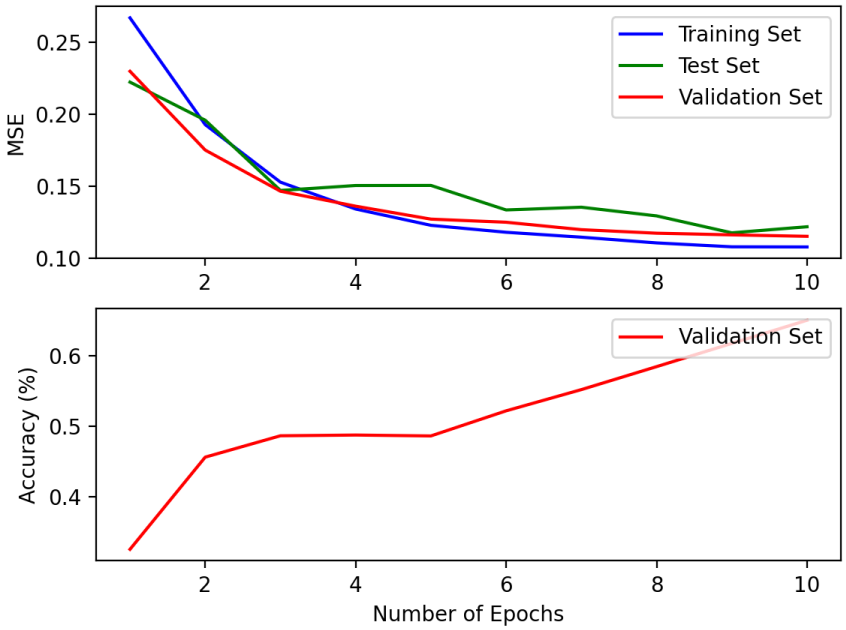


Figure 13: Learning Rate = 0.001, 32 Hidden Nodes & Momentum = 0.1 for vowel data set MSE

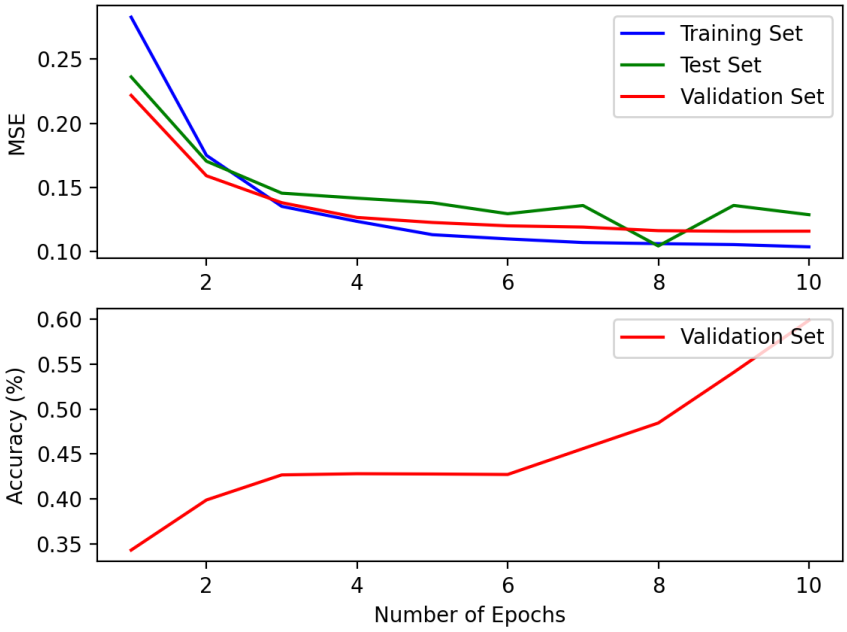


Figure 14: Learning Rate = 0.001, 32 Hidden Nodes & Momentum = 0.5 for vowel data set MSE

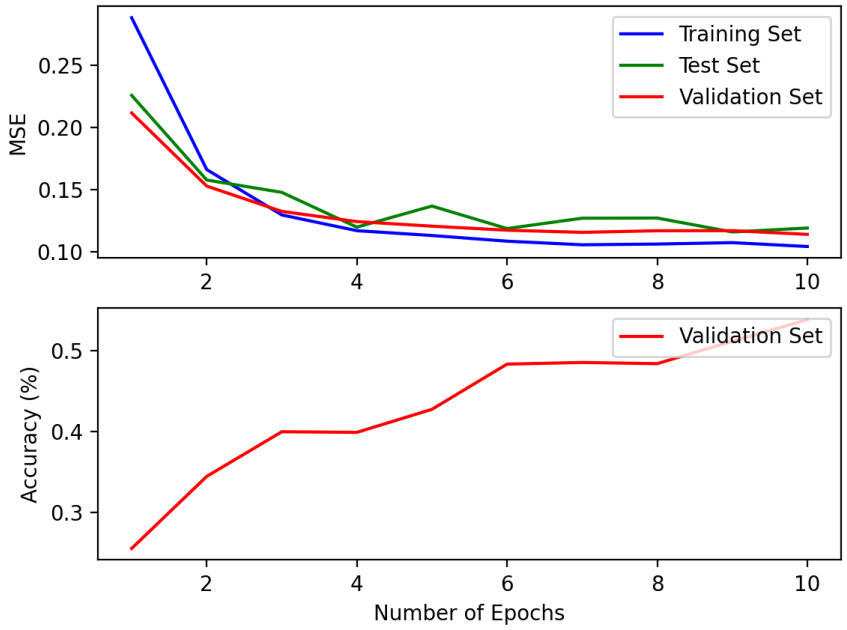


Figure 15: Learning Rate = 0.001, 32 Hidden Nodes & Momentum = 0.9 for vowel data set MSE

Comparison to the Scikit-Learn Toolkit Multi-Layer Perceptron Classifier

Finally, I attempted implementing the Scikit_Learn version of the mult-layer perceptron classifier with the vowel data set. When increasing the number of hidden nodes and layers, the learning speed decreased significantly, however, the accuracy continued to rise until I got to a hidden layer size of 6400 nodes. This was different results from mine, but my classifier also ran much slower, so using this huge number of nodes wasn't reasonable. Different activation functions had varying effects on the accuracy of the model. The identity, logistic, and tanh activations all dropped the accuracy while the relu, rectified linear unit function, activation function seemed to have the most positive influence. The different learning rates such as adaptive and invscaling didn't affect the accuracy but both increased the learning speed of the model considerably.

Additionally, the regularization term (L2 penalty) defaults at 0.0001 and as I increased this term, it didn't make any noticeable difference until I reached 0.01 and at this point it increased accuracy. Once I increased it past that value towards 0.1 and higher, accuracy dropped below the original default accuracy. As for momentum and the Nesterov's momentum term, I didn't see any noticeable changes in accuracy or learning speed. Perhaps with models with much more hidden layers and/or nodes, this would come in handy more. Including early stopping initially had a very negative effect on accuracy while improving learning speed. I jumped the number of hidden nodes up again and saw that the learning speed increase was very significant, but accuracy still suffered from when no early stopping criteria was used.

I tested out a wide range of hyper-parameters to see the best result I could find. In terms of maximizing accuracy and minimizing the amount of time it takes for the model to learn, I found the best accuracy to be 0.8347. The hyper-parameters for this test were as follows: random_state=1, max_iter=300, hidden_layer_sizes=6000, learning_rate='adaptive', alpha=0.01, momentum=0.5, nesterovs_momentum=True, early_stopping=False.