**Abstract**

In this paper we will discuss the application of a neural network machine learning algorithm to predict the outcomes of both classification and continuous targets. This application will be tested on 6 different data sets of varying lengths and complexity. We will look at simple single, layer regression models, multi-layer models incorporating hidden layers with activation functions, and an autoencoder that will be transposed to a multi-layer model for testing.

**Introduction**

In this experiment, we will be creating three different neural networks for six different data sets for which we will test their ability to predict target values. Generally, we believe that the networks will perform better will more layers as the extra transformations should only help their predictive power. However, we will make the prediction that the Computer Hardware and Forest Fire data sets will show more improvements in their multiple layer adaptations relative to the simple models. We believe this to their apparent exponential outputs that do not seem to grow in a linear fashion. Therefore, we hope the non-linear transformations will provide the most benefit to their predictions. The Breast Cancer, Congress Voting, and Car Evaluation test sets are all classification sets, so we still believe that they can be properly bucketed with a linear model. The Abalone test set on the other hand will show improvement as the rest but its target values seem to have relatively clean distributions and a large sample set that should allow it to be properly evaluated with a simple regression model.

Lastly, we believe that the model that is first tuned with the autoencoder will have performance somewhere between the simple and other multiple layer models. We will discuss more of our experimental approach below, but for the sake of providing a hypothesis, our attack for this autoencoder trained model should provide a hybrid approach to our other two. Once we have trained the autoencoder and attached its other hidden layer and output layers, we will not be retraining those original weights from our input layer. Therefore, they will be trained to output close replicas of our input values, and we will only have one hidden layer that will be trained to produce the proper output. Thus if we have already predicted some improvement across the board from single to multiple layer networks, then we would expect some in-between performance improvement from the network that acts like one with only a single hidden layer.

**Experimental Approach**

We created our networks with the help of pandas and numpy libraries within python. Our networks were stored as data frames where the weights from one layer to the next are stored at the child node. In other words, the pointer from the input layer to the first hidden layer is stored on the hidden layer. We chose this for ease in updating these weights during our backpropogations. We trained our networks until we reached our version of convergence. Convergence was tested every three rounds of updates. For tuning, convergence was measured as either we did not reach a new best performance, our new best performance is less than a 1% improvement, or we already ran 15 total tests. For tests, convergence was updated to check for new best performance of less than 2% and we allowed for 30 tests. In tuning, we cared more for proving a hyperparameters efficacy rather than letting it train completely, so 15 tests seemed like a reasonable amount of time to prove its worth. We expanded this to 30 for testing to ensure that the network had sufficient time to train fully, but we increased our band to 2% for required performance improvement to ensure that this increase slack in testing time did not lead us to overfitting the model to our training data.

**Data Assumptions Made**

There were very few data assumptions made during the processing of these tests, but it is worth noting that we normalized our continuous variables, and we employed one hot encoding when faced with categorical variables. When splitting data sets, we ensured that our categorical test sets were properly stratified.

**Tuning**

For tuning, we provided three options for the learning rate (.001, .01, .1) and two options for the percentage of hidden nodes that we employed relative to the size of the inputs (50% and 75%). There were a wide range of other possibilities to choose from, but these options seemed to provide sufficient life to the model training. However, it is worth noting that some tuning results were so large for the abalone and forest fire test sets that they were omitted. Thankfully, the other options worked well in their place.

For the simple models, we were only concerned with testing out our learning rate. Those results are shown in [insert diagram number here]. There was a preference for .01 in most cases, but the Breast Cancer set chose a larger rate of .1 and the forest fires went smaller with .001.

For the multi-layer models, we also considered the proportion of hidden nodes relative to the input layer. There was more variation in the hyperparameters that were chosen here, but it seems like the categorical models preferred higher learning rates and the regression models preferred lower learning rates. The main outlier here is the forest fires example that chose a higher rate of .1. For the hidden nodes, all of the sets preferred the lower rate of .5 with the exception of congress voting that narrowly picked .75.

**Experiment Performance**

Our performance will be broken into a few parts. First, we will look at the performance of our simple regressions. Next, we will look at our multiple hidden node model. We will the pivot over to looking at the models that were first trained using an autoencoder. Lastly, we will do a comparison of the average results of the three options to do a more thorough analysis of our hypotheses laid out in the introduction.