# Performance Effects of Diverse Data Sets on Feed-Forward Neural Networks with Backpropagation

**Michael Downs** MICHAELD20312@GMAIL.COM
*Division of Computer Science*
*Montana State University*
*Bozeman, MT 59715, USA*

**Max Hymer** HYMERMAX1@GMAIL.COM
*Department of Electrical and Computer Engineering*
*Montana State University*
*Bozeman, MT 59715, USA*

## Abstract

In this project, we explore Feed-Forward Neural Networks with Backpropagation using varied numbers of hidden layers to find how both diverse data sets and the number of hidden layers affect the performance of this type of algorithm. Specifically, we tested each dataset on a neural network with zero, one, and two hidden layers. Three of these datasets are classification datasets and three are regression datasets. These datasets are varied in their feature counts, feature ranges, number of classes, target value ranges, and normality. So, we discovered how both the type and variation of a dataset affects our neural network with our three chosen hidden layer sizes. We used Stratified 10-Fold Cross-Validation in order to train, tune, and test our model with accurate representations of our data sets. We also Min-Max Normalized our data so that it would work correctly with our chosen activation functions. We used the Logarithmic Sigmoid Function as our hidden layer activation, the Softmax Function as our output layer activation for classification data sets, and the Linear Function as our output layer activation for regression data sets. Internally, our algorithm calculated loss for backpropagation with Cross-Entropy Loss for classification data sets and Mean Squared Error (MSE) for regression data sets. To evaluate our algorithm, we used 0/1 Loss for classification datasets and Mean Squared Error for regression datasets. We found that the breast cancer identification, forest fires, and abalone datasets performed best. We think that breast cancer identification data set performed well because it is large, well-scaled, balanced, low-noise, and low-outlier. We think that the other classification data sets performed poorly because they were either too small (soybean identification) or contained too many outliers and too much noise combined with a medium size (glass) for the neural network to properly weight nodes and find accurate patterns in the data. We think that the forest fires and abalone data sets performed well because they are large, so our neural network was able to correctly recognize patterns in the data. We think that the computer hardware data set performed worse (but not terribly) because it is significantly smaller than the other two regression data sets. So, we found data set size to be the largest contributing factor for the performance of our neural network. We also found that the average convergence rate (i.e. the speed our neural network converged) was typically proportional to the number of hidden layers among regression data sets (i.e. it was higher when the model utilized more hidden layers) and was ambiguous among classification data sets. Lastly, we found that utilizing our algorithm with two hidden layers had the best performance across all data sets.

**Keywords:** Neural Network, Backpropagation, Cross-Validation, Regression, and Classification

## 1 Introduction

Recently, there has been an explosion of findings involving neural networks and deep learning; they are cutting-edge tools to help us predict complicated relationships and find complex patterns within data. Deep learning is used in image recognition, natural language processing, speech recognition, autonomous vehicles, medical diagnosis, financial services, robotics, and more. With such a real impact of technology on our lives, it is important to understand and research how this idea can be used as a tool for humans. In this experiment, our goal is to contribute to the ongoing research surrounding neural networks by addressing how both diverse data sets and the number of hidden layers within the model affect its performance. We believe that our neural network will work best on all data sets with two hidden layers because this gives the model the ability to thoroughly identify the complex relationships and patterns in the data. We think that it will perform worst on all data sets with zero hidden layers because, in this state, it is purely linear for regression data sets and only goes through Softmax activation for classification data sets, which greatly limits its ability to make complex predictions. We think that the larger data sets will perform better than the smaller data sets for all numbers of hidden layers because the smaller data sets are prone to overfitting, sensitive to outliers, and may lack data diversity. Since both the glass and computer hardware data sets are fairly small and contain a high number of outliers, we expect them to perform poorly. The soybean identification data set is extremely small, so we expect it to perform poorly as well. The abalone, forest fires, and breast cancer identification data sets are all large, so we expect them to perform well. Essentially, we think that simply have enough data that our model will recognize patterns accurately and produce good results even if the data sets aren't perfect. Lastly, we believe that our algorithm will perform better when it converges faster (i.e. has a higher convergence rate) because it is less likely to get stuck in local minima.

## 2 Software Design

Here we will explain our approach to the project software. This means we will go into the significance of our program design, the reasons behind our evaluation metrics, how our learning algorithm works, and a thoughtful explanation of the way it all works together.

### 2.1 Drivers and Helper Classes

We created two unique drivers for each data set that read in the corresponding data set, perform unique preprocessing based on the needs of that data set, and call methods to perform the rest of our experiments. For tuning drivers, a method to extract 10% of the stratified data is called. Then, the remaining data is split into 10 stratified chunks. For testing drivers, the data is just split into 10 stratified chunks. The training and testing data sets for a fold are created and Min-Max Normalized. For classification data set drivers, the training data set outputs (labels) are one-hot encoded. Then, a neural network is initialized and trained with the Min-Max Normalized training data. One final forward pass

of the neural network is performed with the Min-Max Normalized test data. The 0/1 Loss or Mean Squared Error for the fold is calculated from the predictions this forward pass returns. For classification data set drivers, the returned raw percentages are decoded into a predicted class before 0/1 loss is calculated. The test instance, prediction, actual value, 0/1 Loss or MSE, and Average Convergence Rate is printed for the fold. Once the driver iterates through all of the folds, the average MSE or 0/1 Loss and Average Convergence Rate across all of the folds is calculated and printed. The aforementioned Min-Max Normalization, Stratified 10-Fold Cross Validation (including the stratified 10% extraction), One-Hot Encoding, and MSE calculation happen within separate helper classes. The Average Convergence Rate calculation happens with the neural network.

### 2.2 Neural Network Algorithm

We created a neural network that accepts the number of inputs, an array specifying the number of hidden units per layer (which implicitly defines the number of hidden layers as well), number of outputs, activation function type (linear or softmax for the output layer based on whether the data set is classification or regression), learning rate, whether or not to use momentum (boolean), and a momentum coefficient as parameters in the constructor. So, our neural network is very flexible and customizable, which allowed us to perform our experiments in a very methodical and thorough manner. It allowed for very complex tuning as well. The neural network first intializes weights using Xavier Initialization (within a weight matrix)

$$W \sim \mathcal{N}\left(0, \left(\sqrt{\frac{1}{n_{in} + n_{out}}}\right)^2\right)$$

where $n_{in}$ is the number of inputs to the layer and $n_{out}$ is the number of outputs for the layer for each layer to ensure that each neuron computes a unique output and contributes differently to the learning process. We chose Xavier initialization because it works well with sigmoid activation functions, like the logarithmic function we chose. Each weight matrix contains the connection weights between each node in the current layer and each node in the next layer. If there are $n$ input features, $h_1$ nodes in the first hidden layer, $h_2$ nodes in the second hidden layer, $h_f$ nodes in the final hidden layer, and $o$ output features, the weight matrix between the input layer and the first hidden layer will have $n \times h_1$ entries, the weight matrix between the first hidden layer and the second hidden layer will have $h_1 \times h_2$ entries, and the weight matrix between the final hidden layer and the output layer will have $h_f \times o$ entries. Each weight in these matrices represents the strengths of the connection between a node in the first layer and a node in the second layer (e.g. each weight in the matrix between the input layer and the first hidden layer represents the strength of the connection between an input feature and a node in the first hidden layer). We also initialized biases with Small Random Initialization

$$b \sim \mathcal{N}(0, 0.01^2)$$

so that our model could fit data effectively; these biases shift the activation function up or down which allows the model to learn relationships that don't pass through the origin. Then, the neural network begins the training loop, starting with a forward pass over the

data. To start the forward pass, each neuron in a layer computes the weighted sum of its inputs. This weighted sum, denoted as $z$, is calculated as $z = W \cdot x + b$ where $W$ is the weight matrix for the layer, $x$ is the input vector (either from the previous layer or the original input to the network), and $b$ is the bias vector for the neuron in the layer. So, $W \cdot x$, a weighted sum of inputs for a neuron, plus $b$, the bias term, produces the net input to each neuron in the layer. Then, the sigmoid function (a.k.a. the logistic activation function) is applied using

$$a = \sigma(z) = \frac{1}{1 + e^{-z}}$$

where $a$ is the output after applying the activation function and $\sigma(z)$ is the sigmoid function, which squashes the weighted sum $z$ into a value between 0 and 1. This introduces non-linearity into the model, which allows the neural network to model complex, non-linear relationships in the data. This is the crux of the algorithm; it would simply be a series of linear transformations without it. This process continues through all of the hidden layers until the output layer is reached. For this output layer, a different activation function is chosen based on whether model's data set is a classification set or a regression set. For classification sets, we used the softmax activation function

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

where $a_j$ is the output for the $j^{th}$ class, $z_j$ is the weighted sum for that class, and $K$ is the number of classes. For regression sets, we used a linear activation. Since our weighted sum $z$ provides a linear output, it becomes our linear activation. So, regression sets essentially just don't use an activation function in the output layer. After this forward pass, our neural network uses a backpropagation algorithm to adjust our weights based on loss between the predicted output and the actual target value. First, this algorithm computes the loss for the network's predictions. For regression, it uses MSE Loss

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where $y_i$ is the actual value, $\hat{y}_i$ is the predicted output, and $n$ is the number of samples. For classification, it uses Cross-Entropy Loss

$$\text{Cross-Entropy Loss} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

where $y_i$ is the actual label and $\hat{y}_i$ is the predicted probability for class $i$. Then, it applies the calculus chain rule to calculate the gradient of the loss function with respect to each weight in the network and propagate the error backward through each layer in the network. First, the gradient of the loss with respect to the output layer's weights is calculated with

$$\frac{\partial \text{Loss}}{\partial \hat{y}_i} = \hat{y}_i - y_i$$

which gives the error for each output neuron. This is used to compute the gradient with respect to the weights connecting the final hidden layer to the output layer. Next, this

error is backpropagated through the hidden layers. For each hidden layer, we compute the gradient of the loss with respect to the weights connecting the current layer to the next layer, which looks like

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z) \cdot (1 - \sigma(z))$$

for the sigmoid activation function (which is the activation function we plan on using within our hidden layers). Then, we apply the chain rule to compute the gradients for the rest of the layers. Lastly, once the gradients have been calculated, the neural network's weights are updated in order to make it more accurate. We do this with the Gradient Descent method. In this method, the weights are updated by subtracting a fraction of the gradient (controlled by the learning rate) from the current weight values

$$W_{\text{new}} = W_{\text{old}} - \eta \frac{\partial \text{Loss}}{\partial W}$$

where $\eta$ is the learning rate and $\frac{\partial \text{Loss}}{\partial W}$ is the gradient of the loss with respect to the weights. The weights are updated starting with the output layer and moving backwards towards the input layer, layer by layer. We included an option to update the neural network's weights with momentum as well. This option, if turned on, speeds up convergence and reduces oscillations in the learning process because it changes the weight update calculation to include a fraction of the previous weight update. With this option, the update rule is

$$\Delta W_t = -\eta \frac{\partial \text{Loss}}{\partial W} + \alpha \Delta W_{t-1}$$

where $\alpha$ is the momentum coefficient (a value between 0 and 1) and $\Delta W_{t-1}$ is the previous weight update from the last iteration. Therefore, the new weights are updated using

$$W_{\text{new}} = W_{\text{old}} + \Delta W_t$$

when momentum is on. After this final step in the training loop is completed, the loop repeats until the loss converges (reaches a point where it no longer decreases significantly between epochs). So, we did not define a fixed number of epochs or loss threshold; we ran our algorithm as many epochs as it takes for the loss to converge. We believe that this is the most thorough and accurate way to train our neural network because it allows the algorithm to run until ultimate completion. The neural network returns either a single regression output or an array of class probabilities depending on whether the relevant data set is regression or classification.

## 3 Experimental Approach

Our test strategy was a mostly uniform process across the six datasets. Each dataset was modeled with zero, one, and two hidden layers with no layer being larger than the input or previous layer. First, we performed data imputation. This was only necessary on the breast cancer identification data set. To do this, we replaced instances of "?" in the data with a random integer with a value of 1-10 as that is the range of the data and we wanted to keep our data imputation approach simple. Then, we performed 10-Fold Stratified Cross-Validation. For tuning, this strategy involves taking out a stratified 10% of the data and

using it as the test fold, splitting the remaining data into 10 stratified folds, removing 1 fold, and combining the remaining 9 folds into training data until each fold has had a turn being the removed fold. For testing, this strategy involves splitting the data into 10 stratified folds, removing 1 fold to use as the test fold, and combining the remaining 9 folds into training data until each fold has had a turn being the test fold. We Min-Max Normalized both the training and testing folds to make our data compatible with our neural network. Next, we initialized our neural network with the appropriate input size, hidden layer sizes, output size, activation type, learning rate, whether or not to use momentum, and a momentum coefficient. We used different ranges of values for each of our inputs to try and find our best parameters. We tuned and tested with learning rate values 0.1 to 0.0001 with momentum turned off, a momentum coefficient of 0.5, and a momentum coefficient of 0.9 on the model with zero hidden layers. Then, using our loss functions to evaluate the performance of our tests (0/1 Loss for categorical data sets and MSE for regression data sets), we picked one of the best learning rate and momentum coefficient pairs from the zero hidden layer tuning and testing and used it to perform testing and tuning on the number of hidden nodes per layer for one and two hidden layer models. The one hidden layer model is tuned and tested with (number of input layer nodes) - 2, (number of input layer nodes) - 3, and (number of input layer nodes) - 4 hidden nodes for all classes except soybean identification (it made more sense to pick a small, medium, and large number of hidden nodes compared to the number of input nodes here as the number of input nodes was so high). The two hidden layer model is tuned and tested with two hidden layers of the same size as the hidden layer for the one hidden layer model. It is also tuned and tested with the first layer with this number of nodes and the second layer with (this number of nodes) - 2 nodes. This way, a broad range of hidden layer sizes that make sense with each data set is tuned and tested. Then, we trained our neural network and calculated the MSE or 0/1 Loss and Average Convergence Rate for the current fold. Lastly, we calculated the average of these values across all ten folds. It is important to note that one output per class was assumed for regression. It is also important to note that all values in the results section are averages across all ten tuning or testing folds. We kept track of the Average Convergence Rate to find correlations between performance and convergence rate.

## 4 Results

This is our results section where we will represent the data that was found from our experiment. Reflection on the data and the limitations to our testing approach will be expressed in the subsections below the final data tables. Abbreviations are used within our results tables: $\eta$ is the Learning Rate, $\alpha$ is the Momentum Coefficient, 0/1 is the 0/1 Loss, MSE is the Mean Squared Error, ACR is the Average Convergence Rate, L1 and L2 are Layer 1 and Layer 2 respectively, and LS is the Layer Size.

### 4.1 Breast Cancer Identification (Classification)

| 0 Hidden Layers | | | | | |
|---|---|---|---|---|---|
| Variables | | Tuning | | Testing | |
| $\eta$ | $\alpha$ | 0/1 | $ACR$ | 0/1 | $ACR$ |
| 0.1 | OFF | 0.6522 | 0.0212 | 0.6549 | 0.0210 |
| 0.01 | OFF | 0.6403 | 0.0067 | 0.6059 | 0.0064 |
| 0.001 | OFF | 0.3217 | 0.0109 | 0.3496 | 0.0076 |
| 0.0001 | OFF | 0.3333 | 0.0057 | 0.2833 | 0.0123 |
| 0.1 | 0.5 | 0.6522 | 0.0198 | 0.6549 | 0.0202 |
| 0.01 | 0.5 | 0.6580 | 0.0071 | 0.6367 | 0.0377 |
| 0.001 | 0.5 | 0.3271 | 0.0062 | 0.3117 | 0.0096 |
| 0.0001 | 0.5 | 0.2435 | 0.0184 | 0.3146 | 0.0134 |
| 0.1 | 0.9 | 0.6522 | 0.0206 | 0.6549 | 0.0208 |
| 0.01 | 0.9 | 0.6536 | 0.0085 | 0.6404 | 0.0076 |
| 0.001 | 0.9 | 0.2957 | 0.0027 | 0.3181 | 0.0018 |
| 0.0001 | 0.9 | 0.3246 | 0.0043 | 0.3311 | 0.0041 |

| 1 Hidden Layer with $\eta = 0.0001$ and $\alpha =$ OFF | | | | |
|---|---|---|---|---|
| Variable | Tuning | | Testing | |
| $LS$ | 0/1 | $ACR$ | 0/1 | $ACR$ |
| 6 | 0.3362 | 0.0064 | 0.3378 | 0.0039 |
| 5 | 0.3464 | 0.0044 | 0.3438 | 0.0046 |
| 4 | 0.3264 | 0.0039 | 0.3451 | 0.0037 |

| 2 Hidden Layers with $\eta = 0.0001$ and $\alpha =$ OFF | | | | | |
|---|---|---|---|---|---|
| Variables | | Tuning | | Testing | |
| $L1$ $LS$ | $L2$ $LS$ | 0/1 | $ACR$ | 0/1 | $ACR$ |
| 6 | 6 | 0.3478 | 0.0067 | 0.3453 | 0.0082 |
| 6 | 4 | 0.3478 | 0.0072 | 0.3461 | 0.0044 |
| 5 | 5 | 0.3478 | 0.0089 | 0.3451 | 0.0059 |
| 5 | 3 | 0.3478 | 0.0081 | 0.3441 | 0.0053 |
| 4 | 4 | 0.3478 | 0.0070 | 0.3451 | 0.0051 |
| 4 | 2 | 0.3478 | 0.0036 | 0.3262 | 0.0035 |

Figure 1: Breast Cancer Identification Results Tables

### 4.2 Soybean Identification (Classification)

| 0 Hidden Layers | | | | | |
|---|---|---|---|---|---|
| Variables | | Tuning | | Testing | |
| $\eta$ | $\alpha$ | 0/1 | $ACR$ | 0/1 | $ACR$ |
| 0.1 | OFF | 0.7500 | 0.0577 | 0.7023 | 0.0075 |
| 0.01 | OFF | 0.7750 | 0.0133 | 0.7386 | 0.0099 |
| 0.001 | OFF | 0.7000 | 0.0029 | 0.7659 | 0.0041 |
| 0.0001 | OFF | 0.8000 | 0.0013 | 0.7523 | 0.0014 |
| 0.1 | 0.5 | 0.7500 | 0.0077 | 0.7023 | 0.0090 |
| 0.01 | 0.5 | 0.7250 | 0.0157 | 0.7568 | 0.0161 |
| 0.001 | 0.5 | 0.7750 | 0.0056 | 0.7227 | 0.0054 |
| 0.0001 | 0.5 | 0.7000 | 0.0001 | 0.6727 | 0.0009 |
| 0.1 | 0.9 | 0.7500 | 0.0035 | 0.7023 | 0.0042 |
| 0.01 | 0.9 | 0.7250 | 0.0266 | 0.7477 | 0.0276 |
| 0.001 | 0.9 | 0.7000 | 0.0050 | 0.7545 | 0.0049 |
| 0.0001 | 0.9 | 0.7750 | 0.0020 | 0.7500 | 0.0012 |

| 1 Hidden Layer with $\eta = 0.001$ and $\alpha = 0.9$ | | | | |
|---|---|---|---|---|
| Variable | Tuning | | Testing | |
| $LS$ | 0/1 | $ACR$ | 0/1 | $ACR$ |
| 33 | 0.6750 | 0.0026 | 0.7159 | 0.0024 |
| 15 | 0.6500 | 0.0049 | 0.7068 | 0.0058 |
| 5 | 0.6750 | 0.0031 | 0.7250 | 0.0047 |

| 2 Hidden Layers with $\eta = 0.001$ and $\alpha = 0.9$ | | | | | |
|---|---|---|---|---|---|
| Variables | | Tuning | | Testing | |
| $L1$ $LS$ | $L2$ $LS$ | 0/1 | $ACR$ | 0/1 | $ACR$ |
| 33 | 33 | 0.7250 | 0.0057 | 0.6932 | 0.0106 |
| 33 | 15 | 0.6750 | 0.0052 | 0.6909 | 0.0106 |
| 15 | 15 | 0.7250 | 0.0049 | 0.7409 | 0.0071 |
| 15 | 5 | 0.7500 | 0.0049 | 0.7023 | 0.0056 |
| 5 | 5 | 0.7250 | 0.0037 | 0.7409 | 0.0043 |
| 5 | 3 | 0.7500 | 0.0024 | 0.7477 | 0.0033 |

Figure 2: Soybean Identification Results Tables

### 4.3 Glass Identification (Classification)

| 0 Hidden Layers | | | | | |
|---|---|---|---|---|---|
| Variables | | Tuning | | Testing | |
| $\eta$ | $\alpha$ | 0/1 | $ACR$ | 0/1 | $ACR$ |
| 0.1 | OFF | 0.9444 | 0.0139 | 0.9423 | 0.0140 |
| 0.01 | OFF | 0.7944 | 0.0044 | 0.7750 | 0.0051 |
| 0.001 | OFF | 0.6333 | 0.0071 | 0.6308 | 0.0077 |
| 0.0001 | OFF | 0.6167 | 0.0020 | 0.6716 | 0.0020 |
| 0.1 | 0.5 | 0.9444 | 0.0133 | 0.9423 | 0.0142 |
| 0.01 | 0.5 | 0.8111 | 0.0043 | 0.8306 | 0.0048 |
| 0.001 | 0.5 | 0.6500 | 0.0085 | 0.6382 | 0.0082 |
| 0.0001 | 0.5 | 0.6111 | 0.0021 | 0.6438 | 0.0023 |
| 0.1 | 0.9 | 0.9167 | 0.0236 | 0.9423 | 0.0244 |
| 0.01 | 0.9 | 0.9056 | 0.0053 | 0.9250 | 0.0056 |
| 0.001 | 0.9 | 0.6556 | 0.0123 | 0.6622 | 0.0097 |
| 0.0001 | 0.9 | 0.6833 | 0.0033 | 0.6455 | 0.0034 |

| 1 Hidden Layer with $\eta = 0.001$ and $\alpha =$ OFF | | | | |
|---|---|---|---|---|
| Variable | Tuning | | Testing | |
| $LS$ | 0/1 | $ACR$ | 0/1 | $ACR$ |
| 6 | 0.6111 | 0.0070 | 0.6269 | 0.0055 |
| 5 | 0.6111 | 0.0057 | 0.6250 | 0.0071 |
| 4 | 0.6056 | 0.0050 | 0.6288 | 0.0060 |

| 2 Hidden Layers with $\eta = 0.001$ and $\alpha =$ OFF | | | | | |
|---|---|---|---|---|---|
| Variables | | Tuning | | Testing | |
| $L1$ $LS$ | $L2$ $LS$ | 0/1 | $ACR$ | 0/1 | $ACR$ |
| 6 | 6 | 0.6167 | 0.0065 | 0.6346 | 0.0062 |
| 6 | 4 | 0.6000 | 0.0061 | 0.6308 | 0.0051 |
| 5 | 5 | 0.5944 | 0.0064 | 0.6269 | 0.0065 |
| 5 | 3 | 0.6222 | 0.0053 | 0.6365 | 0.0046 |
| 4 | 4 | 0.6111 | 0.0056 | 0.6365 | 0.0053 |
| 4 | 2 | 0.6056 | 0.0038 | 0.6365 | 0.0037 |

Figure 3: Glass Identification Results Tables

### 4.4 Computer Hardware (Regression)

| 0 Hidden Layers | | | | | |
|---|---|---|---|---|---|
| Variables | | Tuning | | Testing | |
| $\eta$ | $\alpha$ | $MSE$ | $ACR$ | $MSE$ | $ACR$ |
| 0.1 | OFF | 0.1737 | 0.0114 | 0.1183 | 0.0093 |
| 0.01 | OFF | 0.2037 | 0.0053 | 0.0949 | 0.0085 |
| 0.001 | OFF | 0.1480 | 0.0400 | 0.1431 | 0.0533 |
| 0.0001 | OFF | 0.1234 | 0.0159 | 0.0809 | 0.0149 |
| 0.1 | 0.5 | 0.1459 | 0.0105 | 0.1890 | 0.0182 |
| 0.01 | 0.5 | 0.1061 | 0.0051 | 0.1943 | 0.0047 |
| 0.001 | 0.5 | 0.2349 | 0.0332 | 0.1415 | 0.0507 |
| 0.0001 | 0.5 | 0.1695 | 0.0231 | 0.1047 | 0.0231 |
| 0.1 | 0.9 | 0.1450 | 0.0091 | 0.2384 | 0.0119 |
| 0.01 | 0.9 | 0.1391 | 0.0046 | 0.1486 | 0.0091 |
| 0.001 | 0.9 | 0.1842 | 0.0423 | 0.1778 | 0.0613 |
| 0.0001 | 0.9 | 0.1497 | 0.0187 | 0.1460 | 0.0166 |

| 1 Hidden Layer with $\eta = 0.01$ and $\alpha =$ OFF | | | | |
|---|---|---|---|---|
| Variable | Tuning | | Testing | |
| $LS$ | $MSE$ | $ACR$ | $MSE$ | $ACR$ |
| 4 | 0.0883 | 0.0733 | 0.0523 | 0.0917 |
| 3 | 0.0967 | 0.0848 | 0.0581 | 0.0577 |
| 2 | 0.0959 | 0.1401 | 0.0636 | 0.1261 |

| 2 Hidden Layers with $\eta = 0.01$ and $\alpha =$ OFF | | | | | |
|---|---|---|---|---|---|
| Variables | | Tuning | | Testing | |
| $L1$ $LS$ | $L2$ $LS$ | $MSE$ | $ACR$ | $MSE$ | $ACR$ |
| 4 | 4 | 0.0853 | 0.1332 | 0.0552 | 0.1467 |
| 4 | 2 | 0.0871 | 0.1721 | 0.0567 | 0.0490 |
| 3 | 3 | 0.0845 | 0.1849 | 0.0550 | 0.1563 |
| 3 | 1 | 0.0862 | 0.1842 | 0.0577 | 0.1106 |
| 2 | 2 | 0.0871 | 0.2136 | 0.0573 | 0.1025 |
| 2 | 1 | 0.0861 | 0.1819 | 0.0569 | 0.1967 |

Figure 4: Computer Hardware Results Tables

### 4.5 Abalone (Regression)

| 0 Hidden Layers | | | | | |
|---|---|---|---|---|---|
| Variables | | Tuning | | Testing | |
| $\eta$ | $\alpha$ | $MSE$ | $ACR$ | $MSE$ | $ACR$ |
| 0.1 | OFF | 0.2986 | 0.0110 | 0.3038 | 0.0182 |
| 0.01 | OFF | 0.1034 | 0.0254 | 0.1750 | 0.0260 |
| 0.001 | OFF | 0.0485 | 0.0111 | 0.1001 | 0.0139 |
| 0.0001 | OFF | 0.0982 | 0.2155 | 0.0626 | 0.2262 |
| 0.1 | 0.5 | 0.1437 | 0.0089 | 0.3794 | 0.0131 |
| 0.01 | 0.5 | 0.0579 | 0.0260 | 0.1814 | 0.0283 |
| 0.001 | 0.5 | 0.0685 | 0.0207 | 0.0715 | 0.0277 |
| 0.0001 | 0.5 | 0.1041 | 0.1615 | 0.0758 | 0.2138 |
| 0.1 | 0.9 | 0.1544 | 0.0130 | 0.3137 | 0.0118 |
| 0.01 | 0.9 | 0.0464 | 0.0257 | 0.1581 | 0.0200 |
| 0.001 | 0.9 | 0.0303 | 0.0152 | 0.0644 | 0.0154 |
| 0.0001 | 0.9 | 0.0719 | 0.1582 | 0.0519 | 0.2849 |

| 1 Hidden Layer with $\eta = 0.001$ and $\alpha =$ OFF | | | | |
|---|---|---|---|---|
| Variable | Tuning | | Testing | |
| $LS$ | $MSE$ | $ACR$ | $MSE$ | $ACR$ |
| 5 | 0.0406 | 0.0459 | 0.0421 | 0.0625 |
| 4 | 0.0403 | 0.0751 | 0.0329 | 0.0720 |
| 3 | 0.0477 | 0.0649 | 0.0281 | 0.0468 |

| 2 Hidden Layers with $\eta = 0.001$ and $\alpha =$ OFF | | | | | |
|---|---|---|---|---|---|
| Variables | | Tuning | | Testing | |
| $L1$ $LS$ | $L2$ $LS$ | $MSE$ | $ACR$ | $MSE$ | $ACR$ |
| 5 | 5 | 0.0373 | 0.0442 | 0.0282 | 0.1227 |
| 5 | 3 | 0.0459 | 0.1091 | 0.0251 | 0.2243 |
| 4 | 4 | 0.0411 | 0.0370 | 0.0271 | 0.1639 |
| 4 | 2 | 0.0520 | 0.1202 | 0.0240 | 0.1678 |
| 3 | 3 | 0.0464 | 0.1206 | 0.0248 | 0.1943 |
| 3 | 1 | 0.0574 | 0.1515 | 0.0232 | 0.2911 |

Figure 5: Abalone Results Tables

### 4.6 Forest Fires (Regression)

| 0 Hidden Layers | | | | | |
|---|---|---|---|---|---|
| Variables | | Tuning | | Testing | |
| $\eta$ | $\alpha$ | $MSE$ | $ACR$ | $MSE$ | $ACR$ |
| 0.1 | OFF | 0.1050 | 0.0059 | 0.1530 | 0.0031 |
| 0.01 | OFF | 0.0550 | 0.0280 | 0.0695 | 0.0556 |
| 0.001 | OFF | 0.0677 | 0.1717 | 0.0986 | 0.1480 |
| 0.0001 | OFF | 0.0747 | 0.0886 | 0.0577 | 0.1333 |
| 0.1 | 0.5 | 0.0976 | 0.0061 | 0.0881 | 0.0057 |
| 0.01 | 0.5 | 0.0667 | 0.0522 | 0.0665 | 0.0300 |
| 0.001 | 0.5 | 0.0602 | 0.1590 | 0.0945 | 0.1914 |
| 0.0001 | 0.5 | 0.0643 | 0.1421 | 0.0819 | 0.0937 |
| 0.1 | 0.9 | 0.0871 | 0.0049 | 0.0940 | 0.0067 |
| 0.01 | 0.9 | 0.0576 | 0.0826 | 0.1183 | 0.0323 |
| 0.001 | 0.9 | 0.0685 | 0.1534 | 0.0906 | 0.0623 |
| 0.0001 | 0.9 | 0.0805 | 0.1117 | 0.0646 | 0.1209 |

| 1 Hidden Layer with $\eta = 0.001$ and $\alpha =$ OFF | | | | |
|---|---|---|---|---|
| Variable | Tuning | | Testing | |
| $LS$ | $MSE$ | $ACR$ | $MSE$ | $ACR$ |
| 6 | 0.0272 | 0.3059 | 0.0251 | 0.3439 |
| 5 | 0.0266 | 0.2195 | 0.0261 | 0.2931 |
| 4 | 0.0272 | 0.2651 | 0.0248 | 0.3110 |

| 2 Hidden Layers with $\eta = 0.001$ and $\alpha =$ OFF | | | | | |
|---|---|---|---|---|---|
| Variables | | Tuning | | Testing | |
| $L1$ $LS$ | $L2$ $LS$ | $MSE$ | $ACR$ | $MSE$ | $ACR$ |
| 6 | 6 | 0.0252 | 0.3284 | 0.0238 | 0.3323 |
| 6 | 4 | 0.0251 | 0.3385 | 0.0239 | 0.3167 |
| 5 | 5 | 0.0251 | 0.2777 | 0.0240 | 0.3653 |
| 5 | 3 | 0.0255 | 0.3445 | 0.0239 | 0.2917 |
| 4 | 4 | 0.0252 | 0.2601 | 0.0242 | 0.3061 |
| 4 | 2 | 0.0253 | 0.2251 | 0.0241 | 0.3359 |

Figure 6: Forest Fires Results Tables

### 4.7 Discussion

Our initial hypothesis that the abalone, breast cancer identification, and forest fires data sets would perform well was correct. We think it was because they were large data sets, so our model was able to accurately analyze patterns and wasn't overly distracted by outliers or noise. The smaller glass identification, soybean identification, and computer hardware data sets likely were greatly disrupted by outliers and noise due to their size and produced

poor results as a consequence. Our hypothesis that generally, models with a higher average convergence rate will perform better, was typically correct for regression data sets, but it turned out to be much more complex than that. We were unable to find any patterns or draw any general conclusions around convergence rate and performance on classification data sets. Models with a higher convergence rate did generally perform better on regression data sets. However, convergence rate appeared to be tied far more closely to number of hidden layers (it increased as number of hidden layers increased). Basically, within a zero hidden layer model, convergence rate was low and within a two hidden layer model, convergence rate was high regardless of how well the hyperparameters at that layer performed. Our initial hypothesis was that models with two layers would perform best and that our models with no hidden layers would perform the worst was generally correct. It is interesting to note though that the improvement from zero to one layers is much more significant than the improvement from one to two layers. This is likely because the data only passes through a linear or softmax output layer activation when there are zero hidden layers, but passes through a logarithmic sigmoid activation and this output activation with one hidden layer. The difference between one and two hidden layers is just another logarithmic sigmoid activation, so it makes sense that adding a whole new activation that introduces nonlinearity does much more than just adding a second layer with this same activation.

### 4.8 Limitations

While our experiment produced many valuable results, it also had some limitations. It is important to note that our inputs to our Neural Network algorithm were somewhat arbitrarily chosen by us. This is important because it means that the learning rate, momentum coefficient, and size of the hidden layers were not necessarily the most ideal values to give us the most accurate model. If we had more time to tune more values, we may have been able to make our model perform better. We also only looked at six datasets, and, although diverse, they do not cover anywhere near all possible types of data sets. So, by processing such a small scope of the information that can be processed, our findings likely do not apply to all possible data sets.

## 5 Summary

Through this experiment, we gleaned valuable insight into the appropriate uses of Feed-Forward Neural Networks with Backpropagation. The algorithm performed best on large data sets, and we noted the highest performance on these data sets when the model made use of two hidden layers. Convergence rate also seems to be tied to the number of hidden layers among regression data sets and is ambiguous among classification data sets. We found that adding hidden layers to the network model does add the ability to understand additional relationships, but adding more and more hidden layers is exponentially less important in finding the correct output than the first hidden layer. Our hypotheses were generally correct. However, our experiment wasn't perfect; it was limited by our somewhat arbitrary tuning choices and our lack of data set variety. Further research is needed on a wider range of tuning values for these data sets as well as testing on even more unique data sets.

## 6 Appendix

Max Hymer:
- Paper (review/editing and combined work listed below)
- Code (none)
- 30% effort

Michael Downs:
- Paper (abstract, software design, review/editing and combined work listed below)
- Code (all)
- Hyperparameter tuning
- 70% effort

Combined Work:
- Video commentary and recording
- Paper (introduction, experimental approach, results, summary)