Machine Learning Project 3: Neural Networks

Nicholas Stone nicholas.stone2@student.montana.edu

Matteo Björnsson matteo.bjornsson@student.montana.edu

Abstract

In this paper we present an experiment analyzing the performance of shallow feedforward neural networks on six data sets. 10 fold cross validation is used to evaluate each algorithm using 0/1 loss and F1 score for classification and Mean Absolute Error and Mean Squared Error for regression. We propose that, in general, Neural Network will perform well on classification problems, data sets with an even class distribution and many samples, and data sets with few preprocessing needs. We suggest that neural networks will perform better with a higher number of hidden layers. We did not find any clear support of these hypotheses with our experimental results. We discuss the possible factors affecting performance. No real conclusions can be drawn here without further testing.

Keywords: Feedforward Neural Network, Backpropagation

1. Introduction

Neural networks are supervised learning model where the network is composed of an input and output layer, each with a certain number of nodes, connected through a number of hidden layers. A given input is fed through the neural network in a "forward pass" and a given output is generated. This output represents the network regression estimate or classification guess which is dependent upon the type of data that is being learned. In this paper we implement a neural network capable of learning both regression and classification data sets. We vary the number of hidden layers in the network from 0-2 and compare results from three classification data sets and three regression data sets using ten fold cross validation. We hypothesize scenarios where the neural network would perform well and where the neural network would not perform well. Relevant background information is reviewed in Section 2, experimental hypotheses in Section 3, and experimental design in section 4. We conclude with our results and discussion in Section 5.

2. Background Information

In this experiment we implement a shallow feed forward neural network trained via backpropagation. Here the term shallow indicates the networks only have zero, one, or two hidden layers. In a general sense, feed forward neural networks are universal function approximators. The functions these networks are able to approximate depend on how many hidden layers are used by the model. A neural network with no hidden layers, otherwise

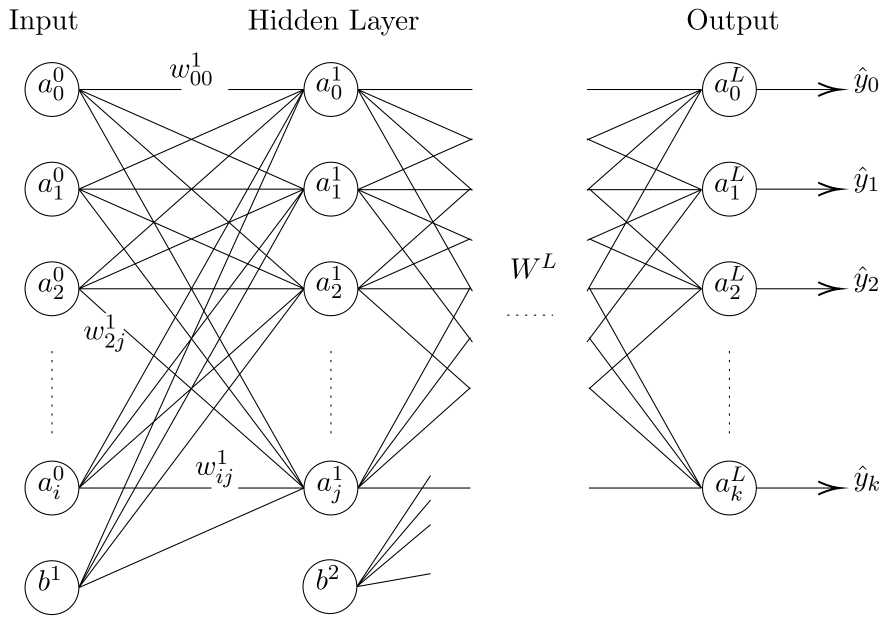


Figure 1: *Structure and notation for a generic neural network. The weights for any given layer are summarized here as a matrix,* W^l. *Additionally, the first layer activation values,* a^0, *are the inputs to the network and are not "activated."*

known as a single layer neural network, or a perceptron, can approximate linear functions. This works well for data sets that are linearly separable. A network using only one hidden layer is capable of approximating any continuous function, allowing for much more complex division of the data set while training the model. Finally, a neural network with two or more hidden layers can approximate functions that are not continuous. A neural network consists of a number of nodes in each layer, where the nodes of one layer and the next are completely connected with edges representing the "weight" between the two nodes. Additionally, every layer has a bias value connected to each node. This structure is illustrated in Figure 1, which also presents the notation used for each network component in the equations below.

Forward Pass The process of propagating inputs through a forward feed neural network is straightforward. At each layer l, for each neuron j in that layer, the net input is calculated from the sum of the node outputs of the last layer, a_i, times the weight between each node in the last layer and the current node for which we are calculating, w_{ij}. This net input is then run through an activation function, such as a logistic function. Equation (1) describes this process.

$$a_k^l = \varphi\left(\mathrm{net}_j\right), \text{ where } \mathrm{net}_j = \sum_i w_{ij}^l a_i^{l-1} + b^l \tag{1}$$

$$\varphi(z)_{\text{sigmoid}} = \frac{1}{1+e^{-z}} \tag{2}$$

$$\varphi(z_j)_{\text{softmax } l,j} = \frac{e^{z_j}}{\sum_{i \in l} e^{z_i}} \tag{3}$$

Here a_k^l represents the activation output for the kth node in layer l. This process of summing weighted inputs and activating the value is repeated for each layer. Any differentiable, non-linear activation function can be used, though some are more useful than others. Usually the activation function is chosen for performance reasons, model stability, or to ensure the outputs at the last layer are in a desired format. We use two activation functions in this paper: sigmoid (2) and softmax (3). Note that the softmax activation function uses the net inputs for every node in the layer to calculate the output for any given node.

Backpropagation The forward pass is used to estimate the output for new test samples on an already trained network. Training a neural network is synonymous with learning the appropriate weights for the network. To do this we must use a supervised learning process called backpropagation. This process updates the weights in the model to minimize the error observed from any given estimation. The loss function that determines the error of a forward pass forms a surface that is traversed using gradient descent. Equations (4) and (5) are Mean Squared Error and Cross Entropy Error, respectively, the two loss functions used in this paper. The weights of the network are adjusted by Equation (6), where the change in weight is determined by the partial derivative of the loss function with respect to the weight times some learning rate, η. The update to the bias component is defined in Equation (7). The process of gradient descent incrementally traverses the surface of the loss function in the direction of minimal values. This process is susceptible to identifying local minima instead of global minima. To combat this problem we can introduce a "momentum" term $\alpha\Delta w_{ij}^{t-1}$ and $\alpha\Delta b_j^{t-1}$ to Eequations (10) and (11) that retain an element of each previous backpropagation pass to help overcome small dips in the error surface.

$$L(\hat{y}, t)_{MSE} = \frac{1}{2}(\hat{y} - t)^2 \tag{4}$$

$$L(\hat{y}, t)_{CE} = -\sum_k t_k \, log(\hat{y}_k) \tag{5}$$

$$\Delta w_{ij} = \eta \left(\frac{\partial L(\hat{y}, t)}{\partial w_{ij}} \right) \tag{6}$$

$$\Delta b_j = \eta \left(\frac{\partial L(\hat{y}, t)}{\partial b_j} \right) \tag{7}$$

$$\frac{\partial L(\hat{y}, t)}{\partial w_{ij}} = a_j \delta_j \tag{8}$$

$$\delta_j = \begin{cases} \frac{\partial L(\hat{y},t)}{\partial a_j} \frac{\partial \varphi(\text{net}_j)}{\partial \text{net}_j} & \text{where } j \text{ is an output neuron} \\ \left(\sum_{k \in (l+1)} w_{jk} \delta_k \right) \frac{\partial \varphi(\text{net}_j)}{\partial \text{net}_j} & \text{where } j \text{ is an inner neuron} \end{cases} \tag{9}$$

$$\Delta w_{ij}^t = \eta \delta_j a_j + \alpha \Delta w_{ij}^{t-1} \tag{10}$$

$$\Delta b_j^t = \eta\delta_j + \alpha\Delta b_j^{t-1} \tag{11}$$

Calculating this partial derivative is at first very intimidating because every node is connected to every node in successive layers; there are many ways a change in a single weight can affect the output of the loss function. Thankfully, we can calculate this derivative layerwise, starting at the output layer and work our way back to the input layer using the Delta Rule, described in Equations (8) and (9). The delta of each inner layer neuron, δ_j, can be calculated using the weights of the downstream layer, w_{jk}, the delta of the downstream layer, δ_k, and the derivative of the activation function at that layer. The delta of the output layer, δ_j^L, is calculated with only the partial derivative of the loss function with respect to the activation values, and the partial derivative of the activation function with respect to the net input. This is generally easier as those functions are chosen for their simple gradient.

This process of forward pass then backpropagation is repeated over and over until the error from the forward pass converges. At this point the model is trained and can be used on test data.

Datasets This experiment tests shallow feedforward neural networks with backpropagation on six data sets. Three of these are regression problems while the remaining three are classification problems. Overall, these data sets are not very large, though the Abalone data set is significantly larger than the rest. Table 1 summarizes details about each data set.

Table 1: Data set summary

Data set	Character	#Sample	#Attr.	Estimate	Notes
Fire	Mixed	517	12	Regress.	Difficult task, output trends toward 0.
Abalone	Mixed	4178	8	Regress.	Relevant features.
Machine	Mixed	209	7	Regress.	Some irrelevant features?
Cancer	Categorical	699	9	Class.	Class split 65/35. Some missing attr.
Soybean	Categorical	47	35	Class.	One class has twice the occurence.
Glass	Real	214	9	Class.	Very uneven distribution of classes.

3. Hypotheses

Neural networks are trained until the error of the forward pass converges on some value. The speed with which this occurs is important because it dictates the cost of running the algorithm and therefore the usefulness of the model. We suspect that data sets with fewer attributes and fewer samples will converge faster. This is simply because there are fewer computations per training epoch. In general, convergence will happen quickly for all data sets because, compared to some data sets used to train deep neural networks with thousands of features or more, these data sets are small with few attributes. Therefore, the Soybean data set will reach convergence the fastest and the Cancer and Fire data sets will take the longest before reaching convergence.

Regardless of how long it takes to converge, we anticipate that the Cancer and Abalone data sets will perform the best overall because of how many samples are available to train the model. Additionally, the Cancer data set has only two possible class values, which we feel will also result in good performance. We suspect that the Soybean data set may suffer from overfitting due to the small number of samples.

Due to the pre-processing and data manipulation some data sets have been changed in nature in order to be able to run on the neural network algorithm. As a result, we expect these data sets may perform poorly compared to data sets that were relatively untouched. We expect that Machine, Forest Fire, and Abalone may be affected by this, whereas Cancer, Soybean and Glass data sets will perform better in this respect.

Finally, we expect that, overall, the neural network will perform better with more hidden layers because a neural network with more hidden layers can approximate more complex functions. Additionally, we expect the networks will perform better on classification problems where a uniform amount of each class is available to be used in the training, as opposed to regression. We suspect it is harder for the network to determine to the exact decimal what the proper regression estimate should be, as opposed to simply choosing between one of several values.

4. Experimental Design

4.1 Preprocessing

The first pre-processing step is to remove features from the data sets that do not have any major influence on the output variable. This is seen only in the Machine, Cancer and Glass data sets with the removal of a sample index and the name of the model and manufacturer of each machine product. Before normalizing all of the data, additional processing was done to data sets that had categorical features. These features were mapped to a integer value if they were not already represented by one. For example, with the Forest Fire data set, the days of the week and month names were converted to an equivalent integer value from 0-6 and 0-11, respectively. This was done because the network is represented by a linear combination of weights and input values, requiring that the inputs can be used in arithmetic operations.

Every data point that had missing attributes was randomly assigned a new value between the minimum and maximum values from the feature space.

After all of the preprocessing steps are executed as described above, the values of each feature are normalized to a value between zero and one. This was done to all data sets. This is an important step to prevent one feature from dramatically "outweighing" another feature in the network. Finally, all of the regression target variables are also normalized between zero and one so a sigmoid activation function can be used to estimate regression output values.

4.2 Hyperparameter Tuning and Experimental Setup

Hyperparameter Tuning We found the neural networks to be very sensitive to input parameters, especially learning rate. The number of batches, learning rate, and number of epochs the model is trained for all seemed to impact each other. Additionally, we tuned for an appropriate number of nodes per hidden layer. The tuning process was not just about finding the most performant parameters, but also finding the best performing parameters

Parameter	Tested Values
Learning Rate	.1, .01, .001, .0001, .00001, .000001
Epochs	5,000, 10,000, 50,000, 500,000
Number of Batches	2, 5, 10, 20, 50
Number of Nodes Per Hidden Layer	From 2 to #{features}, stepped by $^{\text{input size}}/_4$

Table 2: *Hyperparameter values tuned for each data set.*

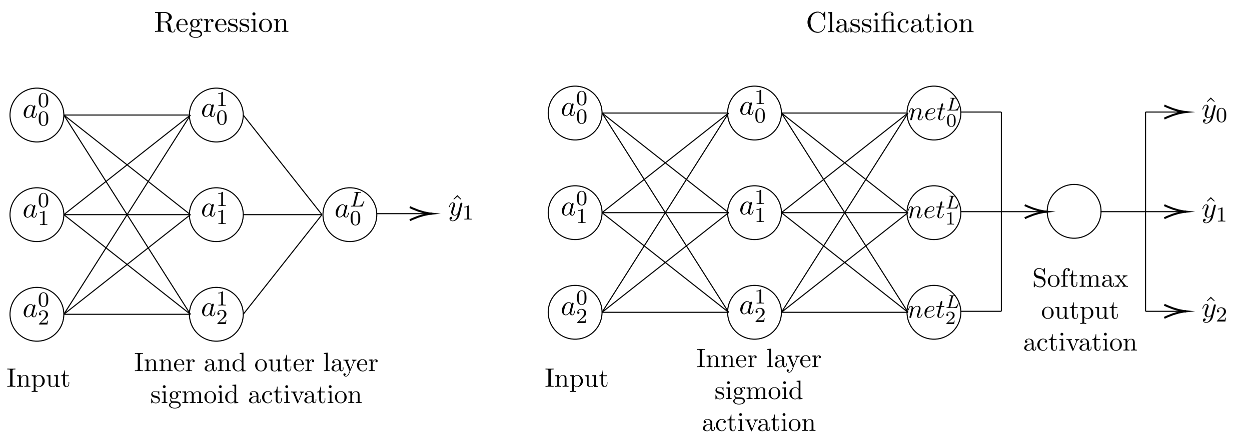


Figure 2: *Single hidden layer network architectures for regression and classification.*

that reduced computational cost. We found that while the number of nodes per hidden layer did not dramatically affect the performance of the model, sometimes we were able to reduce the number of nodes and thus reduce the number of calculations.

We tuned hyperparameters using a random ten percent of the data set as testing data and the remainder as training data. We tested each permutation of the values of the four parameters listed in Table 2 on each data set. We selected the parameters that gave the best results in the fewest forward passes (epochs times number of batches).

Experimental Setup In this experiment, the structure of the network changes depending on the data set. Figure 2 illustrates the differences between the classification and regression architectures. For the three regression problems we use the sigmoid function as the activation function at each node in a hidden layer. The output layer in the activation function has only one node and uses sigmoid function to estimate the regression value. We use a sigmoid activation function for regression output because we found the network to be very unstable when using a linear activation function for the output layer. This is why we normalize the data set target variables to values between zero and one, due to the fact that the sigmoid function is limited to this output range.

For the three classification problems we still use a sigmoid activation for all inner layers. The major difference between the regression networks and the classification networks is the output layer has as many nodes as there are possible ground truth class values. Each output node represents one of the possible class choices. Additionally, the output layer activation function is now the soft max function. This function requires knowledge of all output layer net inputs. This is contrary to the sigmoid function, which only needs to know the net input to the node it is activating. This behavior is essential because it guarantees that all of the output values represent a probability, the largest of which is used to choose the class estimate. The error function used in the regression data set is Mean Squared Error while the error function used in the classification neural networks is Cross Entropy. These loss functions are chosen specifically because they not only match the function of the model, but they also provide for very simple gradients that are used for backpropagation.

A neural network of zero, one, and two hidden layers is trained for each data set. For each number of hidden layers, the data set is cut into ten equal sized folds (randomly for regression, class-stratified for classification) for cross validation. Each of the ten folds is used to test the model after it is trained with the remaining nine folds. The model is then trained for as many forward passes as is required for convergence (determined by hyperparameter tuning). Once the model converges, the test data is propagated through the network in a final forward pass. The results of the forward pass are the network estimates. These estimates are then evaluated as discussed below.

4.3 Evaluation

Different loss functions are used to evaluate classification and regression data sets. The performance of the algorithm on categorical data sets is evaluated using 0/1 Loss and F1 score. Regression is evaluated using Mean Squared Error and Mean Absolute Error.

0/1 Loss is a measure of classification accuracy and only measures the percent of examples classified correctly. 0/1 score does not weight the consequence of misclassification, but such a measure is not required for this experiment. Furthermore, for each class the number of True Positive, True Negative, False Positive, and False Negative assignments are counted. From these counts the precision and recall are calculated on a per-class basis. The F1 score of each class is the harmonic mean of these two values. The total F1 score for the model is a weighted average of all of the per-class F1 scores.

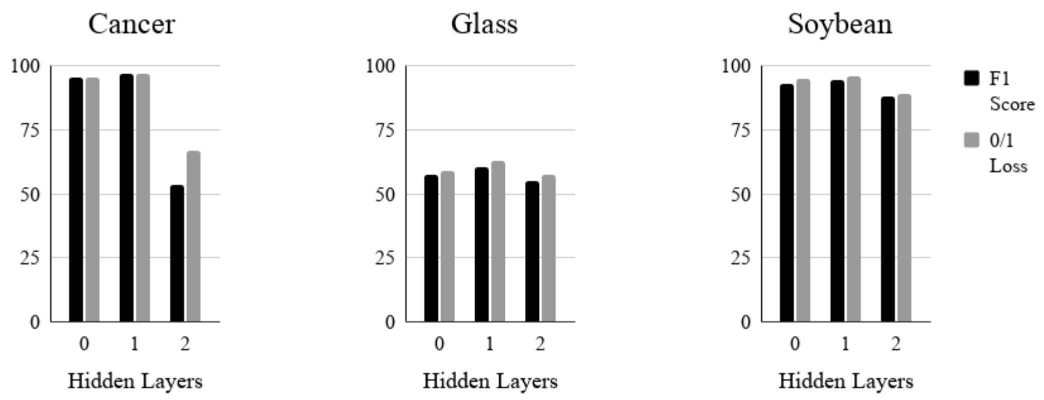
$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

The regression data sets were evaluated using Mean Absolute Error (MAE) and Mean Squared Error (MSE). These loss functions work similarly, with the Mean Absolute Error function taking the difference between the real regression value and the algorithm generated regression value and taking the absolute value of this result. The Mean Squared Error function works similarly by again taking the difference between the two values and squaring the result.

5. Results and Discussion



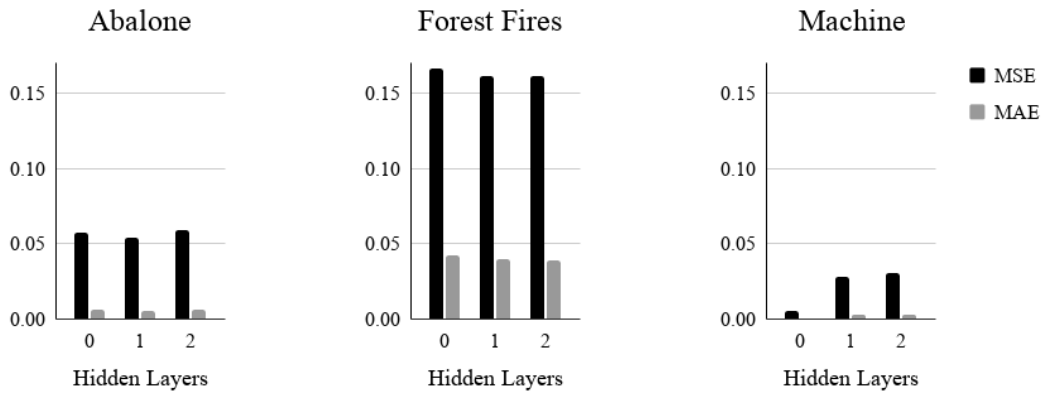
	Cancer		Glass		Soybean	
	F1	0/1	F1	0/1	F1	0/1
Zero	95.43	95.42	57.30	58.87	93.17	95.00
One	96.89	96.85	60.27	62.99	94.40	96.00
Two	53.41	66.67	55.05	57.40	87.97	89.00

Figure 3: *Experimental results for the classification data sets. The performance (F1 and 0/1 score) is charted for the zero, one, and two hidden layer neural networks trained on the data sets. These loss scores are a measure out of 100.*

Our experimental results are summarized in Figures 3 and 4. Some notes on training convergence are illustrated in Figure 5. Cancer and Soybean performed very well, though the performance of the model on the Cancer data set was inconsistent. The Glass data set performed very poorly. For regression, the Machine data set performed the best, while the Abalone data set also performed well. The Forest Fire data set performed the worst for the regression problems.

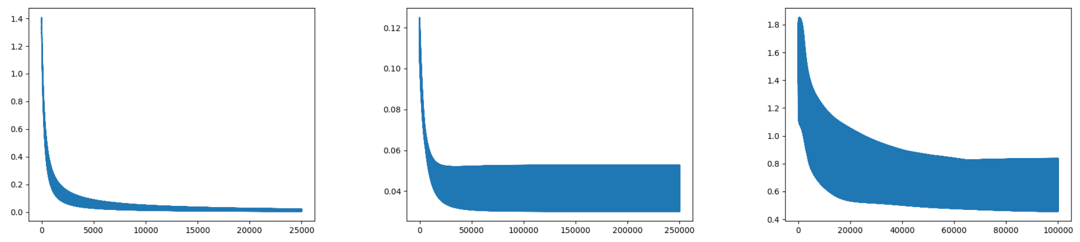
First of all, there is no clear support from our results that neural networks perform better with more hidden layers. In fact, from these initial results, it could even be possible that networks with fewer hidden layers perform better. We think that this must be very dependent on the data set being tested, as is obvious from the No Free Lunch theorem. Some problems are definitely not linearly separable and therefore would benefit from a hidden layer.

The hypotheses that were formed prior to generating experimental results were mostly incorrect, though some of our initial hypothesis turned out to be true. We anticipated that all data sets would converge quickly, and that data sets with the fewest samples and attributes would converge the fastest. Of the problems, Soybean converged the fastest. This data set clearly had the fewest samples, which seems to support our theory, though it does have a larger number of attributes. We found a large variation in convergence over the data sets. Almost all the data sets converged the fastest when there was only one hidden layer. Data sets like Forest Fires took many forward passes to converge, as did the



	Abalone		Forest Fires		Machine	
	MSE	MAE	MSE	MAE	MSE	MAE
One	0.05722	0.00612	0.16673	0.04200	0.00522	0.00023
Two	0.05406	0.00552	0.16177	0.03936	0.02819	0.00264
Three	0.05870	0.00649	0.16163	0.03930	0.03026	0.00268

Figure 4: *Experimental results for the regression data sets. These loss scores (Mean Squared Error and Mean Absolute Error) are a measure of error of the regression estimate compared to the known value.*



(a) Soybean convergence. Zero hidden layers, 25,000 forward passes.

(b) Forest Fires convergence. One hidden layer, 250,000 forward passes.

(c) Glass convergence. One hidden layer, 100,000 forward passes.

Figure 5: *A sample of convergence plots from different data sets and number of hidden layers. The vertical axis is the error for each forward pass, and the horizontal axis is the number of forward passes. The width of the line represents a large variation in error outputs across the batches in each epoch.*

Cancer data set. One reason the Cancer data set performed so poorly with two hidden layers is because it never seemed to converge in this case. Machine converged fairly quickly and cleanly, whereas Glass did converge, but with a large margin of error. More extensive hyperparameter tuning, including smaller learning rates and more forward passes, would be required to truly understand the cause of poor convergence. Examples of convergence plots can be found in Figure 5.

It is difficult to compare the regression and classification problems directly because they use different metrics to measure performance. However, what we do observe is that the models seemed to maintain a similar performance with differing number of hidden layers on the regression problems, whereas in the Cancer data set the performance dropped dramatically with two hidden layers.

Our overall prediction that Cancer and Abalone would perform the best out of all of the data sets was somewhat incorrect. Evident in Figure 3, the cancer data set performed well except when the network had two hidden layers. While it performed the best with one and two hidden layers among the classification problems, due to the fact that Soybean performed consistently across all number of hidden layers, we feel that we cannot cleanly conclude Cancer performed the best. The experiment showed similar finding with the Abalone data set. Overall the data set performed well. However, the Machine data set performed better than Abalone for every neural network. Our hypothesis that Fire and glass would perform the worst was correct. We suspect that this is due to the fact that these are difficult regression and classification problems. These two data sets have performed poorly in both previous experiments with a Naive Bayes classifier and with a K-Nearest Neighbor algorithm.

At the end of the experiment the soybean data set yielded results that were unexpected in the initial hypothesis. Since soybean has such a small number of samples and previous classification algorithms performed poorly on this data set we expected the neural network to produce the same results. However the soybean data set did exceptionally well including when the neural network used 2 hidden layers. One possible reason this data set did well is because it had the largest number of attributes and therefore the most initial nodes.

From the experiment there is no direct indication that more modified data sets performed worse than data sets that were relatively unchanged. One of the more interesting results from the experiments is that most data sets did equally well with zero hidden layers as one. This could indicate that each of the classification problems are linearly separable as a neural networks with no hidden layers can only approximate linear functions. However, it could also be because the zero-hidden-layer networks simply learned and converged faster that they performed just as well. We would need further study to make such conclusions.

6. Conclusion

This paper presented a study of a neural network that was capable of estimating both regression and classification data sets. The algorithm was tested on six data sets, the performance of which was evaluated using 10-fold cross validation. In comparing our hypotheses to experimental results we found no clear conclusions regarding the effect of the number of hidden layers on performance. We did predict that the Fire data set would not perform well due to having its data processed and transformed multiple time, but we also predicted that these algorithms would perform well generally on regression, which was not generally the case here. Extensive testing with a better hyperparameter optimization method could prove insightful and possibly explain the results for the Cancer data set.