Performance of Feedforward Neural Network with Differing Hidden Layers (CS605.449 – Project 6)

By Max Robinson

Abstract

In this experiment the performance of different feedforward neural networks with differing numbers of hidden layers are compared against one another on five datasets. Specifically, neural networks with zero, one, and two hidden layers were compared to each other. The models for each data set were built and tested on five datasets, Breast Cancer, Glass, Iris, Soybean, and Vote from the UC Irvine Machine Learning repository. The results of the comparison showed that the networks with zero hidden layers performed the best on the Breast Cancer, Iris, and Vote datasets. The networks with one hidden layer performed the best on the Glass and Soybean data. At no point did the two hidden layer networks outperform any of the other networks. The conclusion drawn from the experiment is that generalized tuning can hurt performance of more complex neural networks.

Problem

The problem being investigated in this experiment is how well a Feedforward Neural Networks with differing numbers of hidden layers performs on five different datasets. The Feed Forward Neural Networks compared are networks with zero, one, and two hidden layers. The five datasets used are the Breast Cancer, Glass, Iris, Soybean, and Vote datasets from the UC Irvine Machine Learning Repository.

For this experiment, I hypothesize that the neural networks with two hidden layers will perform the best when compared to networks with zero or one hidden layer. Networks with zero hidden layers perform closely to typical logistic regression. However, Logistic regression is a linear discriminant and is unable to learn non-linearity in the data. I believe that this data does have non-linearity in it.

A network with one hidden layer can be used as a universal approximator, given a sufficient number of nodes in the layer [1]. This is advantageous over logistic regression and gets away from the problem of data being not linearly separable. However, I believe it will be difficult to find the correct number of nodes to put in the single hidden layer to gain this universal approximation. A network with two hidden layers has the same properties however and provides a chance to find the correct number of nodes in layer two combined with layer one to gain this universal approximation ability. This characteristic of a neural network with two hidden layers gives the most flexibility in learning.

Since it is unknown if the datasets being used have discontinuities in them, I believe that the two hidden layers neural networks will perform the best since it can learn any function and is more likely to have closer to the correct number of nodes to provide this gain in accuracy.

Algorithm Implementation

Feedforward Neural Network

Forward Pass

A feedforward neural network is a structure for a learning algorithm that is uses connected nodes in a directed manner to solve the problems of classification or regression. Specifically feedforward neural networks are typically multi-layered. This means that there are layers of nodes with non-linear activations functions that are connected to another layer of nodes. When connecting nodes from layer to layer, every node from a layer is connected to every other node in the next laye r. Each connection from one node to another has a weight w_{ji}. A bias node with the value of 1 is also introduced at each layer, but no nodes connect to this bias, as it serves only to help serve as a threshold value [3].

For a node, a non-linear activation function is used to determine and constrain the val ue of the inputs before passing on the value. It is important that this activation function be non-linear as this is what allows a network to learn non-learn models. For this experiment the non-linear activation function used was the logistic function.

To calculate the value of a given node in the network, a linear combinations of the input values multiplied by the weight on the connection is taken as follows [3].

$$x_j = \sum_i y_i w_{ji}$$

where x_j is the jth unit or node and y_i is the ith input into j and w_{ji} is the weight between unit i and j. This linear combination is then put through the nodes activation function which for this experiment is the logistic function [3].

$$y_j = \frac{1}{1 + e^{-x_j}}$$

Here y_j is the output of node j, with input value of x_j as described above. Again, notice that the activation function is non-linear.

To get the total error for the network, which is how different the output of the network is from the expected output of the network is described using a form of squared error. Formally E can be described as follows [3].

$$E = \frac{1}{2} \sum_c \sum_j \left(y_{j,c} - d_{j,c}\right)^2$$

Here, c is an index over the dataset, and j is an index over the output units. For example if there could be two output nodes of the network. y is the output of the network for a given output node, and d is the desired state for that output node. This sum gives the total error of the st ate on a forward pass through the network.

The output of a forward pass through the network is also what is used in this experiment to classify a data point once a network has been trained.

Back Propagation

To train a feed forward neural network a technique called back propagation is used, which was developed by Rumelhart, Hinton, and Williams [3]. Back propagation is used to incrementally update all of the weights in the neural network based on the error of the output of the network. This technique uses Gradient Descent by following the partial derivative of the error function with respect to the inputs and outputs to optimize the network. This update is done in two parts, first calculating the update rule for the output layer, and then calculating the weight updates for the hidden layers. The update rule for the output layer is described as follows

$$\Delta w_{ji} = \alpha \left(d_j - y_j\right) y_j \left(1 - y_j\right) x_{ji}$$

Here, alpha is the learning rate, d_j is the correct output for node j and y_j is the estimated output of the network for the jth output node, and x_{ji} was the input value of the node before the activation function was applied. This Δw_{ji} is then applied to all nodes j connect to output node i. This follows gradient descent because the first and second parts of the equation are the deriviates of the error function with respect to the outputs and the outputs with respect the activation function respectively [3].

For hidden layers, a different gradient is used since the error is not directly observable. To update the hidden layers of nodes, the error for a hidden node must first be calculated by figuring out the contribution that node had to the error of every output node to which it is connected. After this delta error is found, they are summed together to describe the total error of that node. This is described by the following equation.

$$\sum_{k \,\epsilon\, downstream(j)} \delta_k w_{kj}$$

This sum is then combined with the derivative of the nodes activation function. This gives the total derivative update described by

$$\delta_j = \left(y_j\right)\left(1 - y_j\right) \sum_{k \,\epsilon\, downstream(j)} \delta_k w_{kj}$$

The update rule for weights for hidden layers then becomes

$$\Delta w_{ji} = \alpha \delta_j x_{ji}$$

This update rule applies to all hidden nodes and weights. These updates are applied to the network following the typical gradient descent pattern. Once the network has finished training, the forward propagation part of the process of the network is used to get estimated outputs.

Gradient Descent

Gradient Descent is a method for function minimization that was originally proposed by Cauchy [2]. For this experiment we specifically used steepest descent which is often synonymous with gradient descent [2]. The idea behind gradient descent is that if one takes a step along the negative gradient of a function, $f(x)$, then the value of the function should decrease. Continued steps along the negative gradient of the function should continue to decrease the function value until a local or global minimum is reached.

For this experiment, the function that describes the gradient descent are as follows, which are described above.

$$\Delta w_{ji} = \alpha\,(d_j - y_j)y_j(1 - y_j)x_{ji} \text{ and } \Delta w_{ji} = \alpha\delta_j x_{ji}$$

In these update equation, α determines the size of the step that is taken along the gradient. The typically method used for steepest descent does a search for alpha as follows $\alpha_k = argmin_\alpha f(x_k + \alpha d_k)$ where $d_k = -\nabla f(x_k)$ [2]. What this describes is finding alpha such that the $f(x_k)$ along the negative gradient is the smallest value possible, thus finding the maximal step length to take along the gradient.

For this experiment, instead of doing this search for α_k a fixed value of α is used for the step size to take along the negative gradient. For this experiment $\alpha = .1$.

For gradient descent, the network continues to be updated until some convergence criterion is met. For this experiment, convergence is met once the average error of the network stopped changing by less than some value ϵ. The other stopping criteria for the network was if the average error of the network increased from the previous error. This meant that a local minimum was close and that the network was close enough. This was done in the interest of time. For this experiment $\epsilon = .00001$.

This value constitutes as a meta parameter for the gradient descent algorithm and can be tune d according to the results and error rates. For this experiment, epsilon was chosen after anecdotal experimentation across a subset of the test datasets.

Scoring

Error Rate

Error rate is used in this experiment to calculate how many misclassifications occurred when compared to the total number of data points classified. Error rate can be described as

$$errorRate = \frac{\#\ of\ misclassified\ test\ instances}{total\ number\ of\ test\ instances}$$

Experimental Approach

The approach for this experiment was two fold. The first part was done through data preproce ssing, and the second part was done through how the experiments were run. The metric used for comparison of the three different neural networks in this experiment was error rate. This is simply the number of incorrect predictions divided by the total number of items in the test set.

Data Pre-processing

Each data set was pre-processed to turn all of the features into binary features, 0's and 1's. The process for each data set was a little different, since they all had different ranges of values and types of values.

For the Breast Cancer data set, the ID feature was removed from the set since it was a unique id and does not provide correlation with the data. The remaining features all have values between 1 and 10 so

nine new features were created for each one and a one's hot encoding was applied to each new feature. As a result, there were 90 total features, plus the class label per feature vector.

For the Glass data set, the unique id was removed. All values were floats, and had a max and min range for the data set. These values were then binned into 6 bins. A one's hot encoding was applied to which bin a given value fell into.

The iris dataset was also real numbers based and had a maximum and minim value for each feature. The same tactic of binning and one's hot encoding was used for this dataset. 6 bins were also used.

The soybean dataset was integer based, with a maximum value in the entire data set of 6. As such, each integer feature was given 6 binary features, and a one's hot encoding was applied. For missing features, a random value between 0 and six was generated and used in its place.

The house votes dataset was binary based, so all the values were transposed into 1's and 0's. For missing values, a random number 0 or 1 was generated for that place. This data set had the largest portion of missing data.

Cross Validation and Validation Set

For this experiment, cross validation was used to provide robust and averaged results across different distributions of the data. Five-fold cross validation was used for this experiment. This means that from the complete set of the data, five partitions are created. For each run of each neural network the five folds are rotated through, where four of the folds are merged to create training set of 80% of the data, and one fold is used as the test set, 20%. The partition that is used for the test set is rotated so that every partition is used as the test set once.

The data used for each fold in the cross validation are also stratified since all data sets were classification problems. This means that the distribution of the data for each fold mirrors that of the distribution of the data in the entire training set. If class A makes up 40% of the data in the entire data set, then class A will make up 40% of the data in each fold for cross validation.

Classification

To get classify the datasets, each algorithm was run on each data set with a one-vs-all classification approach with cross validation. As such, each data set was tested with each possible class as the positive class.

When classifying the results, the neural network with the architecture corresponding to that dataset was used for classification. For example, one neural network used for the Breast Cancer dataset with zero hidden nodes had 90 inputs and one output node. This architecture was used to do the classification for all data run with a neural network with zero hidden layers for the Breast Cancer dataset.

For data sets with more than two classes, this means that the positive class used was selected at run time, and one class was selected. All other records that were of a different class were then lumped into the negative class. Each possible class value for each dataset was used as the positive class. This allows error rate to be measured for each class individually for the data set.

Error rate for each algorithm was calculated for each fold. The error rates were then averaged across the number of folds, in this case five folds. The standard deviation between the folds was then also calculated.

Network Structure and Choosing Hidden Layer Nodes

Neural networks rely heavily upon the architecture of the network for how well it performs. As such, finding the appropriate structure of the network is important, especially for choosing the number of nodes to be in each hidden layer. The architectures also differ by dataset since each data set responds differently to different architectures.

The general architecture that was the same for all networks was the number of input nodes and the number of output nodes. The number of input nodes was always equal to the number of features in the dataset. There was always one output node. For the networks with zero hidden layers, the architectures were simply the number of input nodes to the one output node.

To find the appropriate number of nodes for the hidden layers however is not obvious. To determine the best number of hidden nodes to have in each hidden layer, a search over the number of nodes in the hidden layer was performed for each data set. This search was first done for having just one hidden layer.

The search done was simplistic. For each dataset, a random class for that dataset was chosen to use as the class to classify. A neural network was then trained with five, ten, fifteen, and twenty nodes in the hidden layer. The error rate for each network was recorded and then compared to the other networks. The network with the lowest error rate was chosen as the network to use for the experiments.

When searching for the number of nodes in the second hidden layer, two approaches could be taken. Search for the combination of first layer and second layer hidden nodes that work best, or bootstrap the search by using the best number of nodes found for a single hidden layer through the previous search and only search for the second hidden layer nodes. In the interest of time, the latter approach was taken.

Using the number of nodes from the first search as the first hidden layer, the search for the number of nodes in the second hidden layer was performed the same as the search for the nodes in the first hidden layer. Again, the networks with the lowest error rate were then used in the experiment.

Table 1 shows the overall network structure for each dataset with a given number of hidden nodes.

Table 1 Architecture of Neural Networks per Dataset

Dataset	0 Hidden Layers	1 Hidden Layer	2 Hidden Layer
Breast Cancer	90 -> 1	90 -> 5 -> 1	90 -> 5 -> 20 -> 1
Glass	54 -> 1	54 -> 5 -> 1	54 -> 5 -> 5 -> 1
Iris	24 -> 1	24 -> 20 -> 1	24 -> 5 -> 15 -> 1
Soybean (small)	204 -> 1	204 -> 5 -> 1	204 -> 5 -> 5 -> 1
House-Votes	16 -> 1	16 -> 5 -> 1	16 -> 5 -> 5 -> 1

Results

After the running the experiment as described in the above sections, the average error rates for each dataset and each positive class can be viewed in Table 2. The table lays out each data set on each row, with sub rows describing which class was used as the positive class the data set. The average error rates for the Neural Networks with a different number of hidden layers are highlighted.

Table 2 Average Error Rate of Neural Networks with differing numbers of hidden layers per Dataset and Positive Class

Error Rate Table	Positive Class Used	0 Hidden Layers Average Error Rate	0 Hidden Layers SD	1 Hidden Layer Average Error Rate	1 Hidden Layer SD	2 Hidden Layer Average Error Rate	2 Hidden Layer SD
Breast Cancer	Benign	0.03768	0.00845	0.05217	0.01478	0.22029	0.14748
	Malignant	0.03913	0.01561	0.04638	0.00580	0.29710	0.29616
Glass	1	0.32195	0.06435	0.39024	0.07235	0.37073	0.06435
	2	0.35610	0.05022	0.33171	0.02487	0.37561	0.03308
	3	0.08780	0.01195	0.07317	0.0	0.07317	0.0
	5	0.06341	0.01951	0.04878	0.0	0.04878	0.0
	6	0.03415	0.01195	0.02439	0.0	0.02439	0.0
	7	0.04878	0.03779	0.07317	0.04363	0.12195	0.0
Iris	Iris Setosa	0.0	0.0	0.4	0.32660	0.33333	0.0
	Iris Versicolour	0.06667	0.04216	0.66667	0.0	0.33333	0.0
	Iris Virginica	0.06	0.05735	0.54667	0.24000	0.33333	0.0
Soybean (small)	D1	0.875	0.0	0.125	0.0	0.125	0.0
	D2	0.875	0.0	0.125	0.0	0.125	0.0
	D3	0.875	0.0	0.125	0.0	0.125	0.0
	D4	0.625	0.0	0.375	0.0	0.375	0.0
House-Votes	Democrat	0.04186	0.01740	0.04651	0.01040	0.10930	0.13770
	Republican	0.03953	0.01186	0.16279	0.22750	0.05349	0.02279

Table 3 Standard Deviation of NN with Differing numbers of hidden layers per Dataset and Positive Class

Error Rate Table	Positive Class Used	0 Hidden Layers SD	1 Hidden Layer SD	2 Hidden Layer SD
Breast Cancer	Benign	0.00845	0.01478	0.14748
	Malignant	0.01561	0.00580	0.29616
Glass	1	0.06435	0.07235	0.06435
	2	0.05022	0.02487	0.03308
	3	0.01195	0.0	0.0
	5	0.01951	0.0	0.0
	6	0.01195	0.0	0.0

	7	0.03779	0.04363	0.0
Iris	Iris Setosa	0.0	0.32660	0.0
	Iris Versicolour	0.04216	0.0	0.0
	Iris Virginica	0.05735	0.24000	0.0
Soybean (small)	D1	0.0	0.0	0.0
	D2	0.0	0.0	0.0
	D3	0.0	0.0	0.0
	D4	0.0	0.0	0.0
House-Votes	Democrat	0.01740	0.01040	0.13770
	Republican	0.01186	0.22750	0.02279

Behavior

The behavior of the three different architectures of neural networks and the corresponding error rates did not perform anywhere close to as expected.

When comparing the different neural networks, the results were all across the board. For some datasets the networks with zero hidden layers performed the best. Examples are the Breast Cancer, Iris, and Vote datasets. For other data subsets, one hidden layer performed better, such as in the Glass and Soybean datasets. On no dataset or class in a dataset did two hidden layers perform better on average than either of the other two networks. This was entirely unexpected.

The other unexpected result of the experiments was the variation of standard deviations of the different experiments. For zero hidden layers, the standard deviations were all fairly small ranging between 0 and .64. One hidden layer networks had larger standard deviations with some standard deviations ranging up to .32, like on the Iris Setosa class. This means that there is a 60% chance that on any given run of a network performing with .32 less or more error rate than the average. For this example, that would mean that the error rate for 1 hidden layer for the Iris Setosa class could be as low as 8%, or as high as 72%.

Two hidden layer networks had similar standard deviation ranges. The most extreme example being the malignant class of the Breast Cancer dataset, where the average error was .297 and the standard deviation was .296. This shows that it is possible given a 60% chance that the error rate for this class is as low as .001 or .1% for classification, but on average it was much higher.

The other very interesting thing about the standard deviation were the number of standard deviations of zero. Both one hidden layer and two hidden layers had a significant number of standard deviations in the experiments that were zero, and two hidden layer networks tended to have even more. This would suggest that for each cross validation, all of the networks for each of the folds learned the exact same thing.

To explain some of these phenomenon's, I believe that the increased errors comes down to tuning parameters for the neural networks. A general approach was taken to constructing these neural networks, and assumptions where made about how the networks were constructed and trained across all of the datasets. The primary assumptions where the epsilon and alpha values, and the structure of the networks, including how the hidden layer searches were done.

For all experiments, the epsilon and alpha values were kept the same, epsilon was .00001 and alpha was .1. I think these had a huge impact on training when it came to increasing the number of hidden layers. As the number of hidden layers grew, both the time it took to train the algorithm and the amount that each network changed per back propagation was decreased. This meant that the stopping point for the network to finish learning occurred earlier on in the gradient descent because the error changed less. This could have caused decreased performance in the networks with multiple layers.

The structure of the network also played a role. The search for the number of hidden nodes per layer for both searches was done in a fairly simplistic manner due to time constraints. As a result, the number of hidden nodes chosen could have not generalized well, or could have been a "fluke" or some random chance that it was the best number of nodes when the search was run. In addition, only the number of nodes in increments of five were considered. This assumption could have been incorrect and the correct or optimal number of nodes could have been a number that was not tested during the search.

The wide variation of standard deviations I believe can be attributed in part to the random initialization of the weights. Each fold for cross validation re-initialized the weights of the network. As such, it is possible that some network initializations were much better than others and lead to a much more performant neural network than others, based on where on the gradient the gradient descent started. One bad initialization of a network could lead to a skewed average error rate.

To address the several zero standard deviations that occurred in the hidden layer networks, I believe that the networks, based on the initialization and the number of weights were more likely to find local minimums and get stuck there. This appeared to happen frequently for the two layer hidden network, and sometimes for the one layer hidden network.

Summary

Overall, the feedforward neural networks did not perform at all as expected. There was a trade off on which datasets zero hidden layers and one hidden layer performed best on. On the Glass and Soybean datasets, the one hidden layer networks performed better, while on the rest the zero hidden layer networks performed the best. The two hidden layer networks did not outperform either of the other networks on any of the dataset. There was a wide range of standard deviations with some being extremely high. This is attributed to the random start of the weights in the network and the possibility for different gradients being followed for different starts due to the large number of weights possible. The trade off in performance and lack of performance of the one and two hidden layer networks is attributed mostly to the generalized approach to tune the networks for each dataset and architecture, rather than tuning parameters specifically for each dataset and network architecture.

References

[1] Hornik, Kurt. "Approximation Capabilities of Multilayer Feedforward Networks." Neural Networks, vol. 4, no. 2, 1991, pp. 251–257., doi:10.1016/0893-6080(91)90009-t.
[2] Meza, J. C. (2010), Steepest descent. WIREs Comp Stat, 2: 719–722. doi:10.1002/wics.117
[3] Rumelhart, David E., et al. "Learning Representations by Back-Propagating Errors." Nature, vol. 323, no. 6088, 1986, pp. 533–536., doi:10.1038/323533a0.

Data Sources

Breast Cancer— https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29
Glass — https://archive.ics .uci.edu/ml/datasets/Glass+Identification
Iris — https://archive.ics.uci.edu/ml/datasets/Iris
Soybean (small) — https://archive.ics.uci.edu/ml/datasets/Soybean+%28Small%29
Vote — https://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records