An Inquiry Into the Multi Layer Perceptron

Seth Bassetti SETH.BASSETTI@STUDENT.MONTANA.EDU
School of Computing
Montana State University
Bozeman, MT, USA

Ben Holmgren BENJAMIN.HOLMGREN1@STUDENT.MONTANA.EDU
School of Computing
Montana State University
Bozeman, MT, USA

Wes Robbins WESLEY.ROBBINS@STUDENT.MONTANA.EDU
School of Computing
Montana State University
Bozeman, MT, USA

Editor: Seth Bassett, Ben Holmgren, and Wes Robbins

Abstract

This paper describes our group's findings when implementing a feedforward multi layer perceptron neural network with back propagation, and running the model on various sets of data. More specifically, we show how our model performs on a set of breast cancer data, on data pertaining to different kinds of glass, on data related to different kinds of soybeans, predicting the age of abalone, performance values for various computer hardware, and on forest fire data. The performance of our model on each respective set of data is evaluated in the context of both classification and regression. For datasets where classification was performed, performance was evaluated in terms of 0/1 loss and average cross entropy. For datasets where regression was performed, mean squared error and mean absolute error were the chosen metrics to evaluate our model. We provided hypotheses for each specific dataset in terms of their overall performance in our model. Namely, we hypothesized the specific performance intervals and rough convergence times for each dataset, and our hypotheses were generally upheld. For classification and regression data, we chose a specific performance value for our hypotheses, and decided to uphold our hypotheses if they fell within 10% of the hypothesized value. In terms of convergence time, we specified a predicted change in convergence time by switching the number of hidden layers for a given problem, and confirmed our hypothesis if the convergence time found experimentally was within 10% of the predicted value.

Keywords: Neural Network, Feedforward, Back Propagation, Multi Layer Perceptron

1. Problem Statement

Utilizing six datasets- each from unique and differing settings, we implemented a feedforward neural network as an attempt to provide insights into the performance of this kind of neural network with respect to differing kinds of data. The model worked on these varying data sets with the aim of conducting supervised learning, that is- accurately guessing the class, or roughly the underlying function value in the case of regression, corresponding to each entry in the data given other examples of each respective class the model is attempting to guess. More rigorously, we utilized a multilayer perceptron to carry out learning on regression and categorization data. In particular, three of the datasets are used to perform regression (Abalone, Computer Hardware, and Forest Fires), and three are used to perform classification (Glass, Soybean, Breast Cancer). Each of the six datasets we use in the project have a variable number of classes and either discrete or real valued attributes. As the data sets are each representative of pretty drastically different situations, we generated completely seperate hypotheses for each kind of data with regards to its performance in our model.

For the abalone data, we hypothesized that mean squared error and mean absolute error would reflect a high degree of efficacy in predicting the age of abalone. Explicitly, we presumed that mean squared error would be roughly a value of 20, and mean absolute error would be roughly a value of 2. The rationale behind these values being that the data provided with each entry in the dataset would seem to be highly correlative with the age of abalone (things like the size, weight, and rings on each creature). We thought that our model would be able to generally predict age successfully within 2 years, which is generally reflected in these hypothesized error values.

For the computer hardware data, we thought that, as with the abalone data, we would expect attribute values to be highly correlative to the classes of hardware. Namely, we presumed that mean absolute error would be roughly a value of 80- as each individual class generally spanned a range of about 100 PRP, with a few smaller outliers. This would seem to imply then a mean squared error of roughly 6400.

For the forest fire data, we thought that predicting classes would be a bit trickier, since here we are attempting to predict the burned area of the forest. Just from briefly examining the data, it seems that the area burned can range anywhere from 0 to roughly 1000 hectares. However, most area values are relatively low, most are under 50. A high performing model in this scenario would likely produce a mean absolute error around 10. We had relatively low expectations for our model on this particular dataset, as the attribute data doesn't seem to be particularly revealing for class values. We predict a mean absolute error around 30 and a mean squared error around 1000, but are more or less shooting from the hip with this hypothesis.

Moving on to the classification data, we presumed that our model would perform reasonably well comparatively on the glass data set, which had 7 total classes. Guessing randomly would provide a roughly 14% rate of accuracy, so we decided that any accuracy percentage above 90% would mean a high performance for this data. Thus, we hypothesized a 0/1 loss value of roughly 0.20, and a cross entropy value smaller than 0.1.

In terms of the small soybean data, the feature values would seem to be highly related to each of the soybean classes, and the data set only featured four classes, meaning that randomly guessing should have a 25% success rate. As such, we hypothesized a roughly 95% accuracy rating, implying a 0/1 loss value of roughly 0.05. We hypothesized also a cross entropy value 0.001 on average.

Lastly, for the breast cancer data, we expected to find that our model also performed quite well. In terms of specific values, the breast cancer categorized only two classes (malignant and benign cancer), so random guessing would provide correct answers in theory 50%

of the time. Meaning that a high performing model we'd expect to be accurate certainly over 95% of the time, and we presumed it would be correct 98% of the time. Implying a 0/1 loss value of roughly 0.02. In terms of entropy, this was harder for us to predict. Since there were only two classes, a higher amount of entropy may be expected than in other data sets, so we predicted an average cross entropy value of roughly 1.

Wholistically, we expect to find that the deeper the network, the better the performance of our model after full hypertuning has taken place.

We also provide a hypothesis with respect to the amount of time each data set would take to converge in our model, which are provided with the knowledge of the number of hypertuned hidden nodes used in each experiment. Though not a perfect way to measure convergence time, we choose to simply time each run of our model on each differing dataset, and record these times. Perhaps more interesting than providing specific hypotheses for each individual dataset with respect to their exact times, we provide a hypothesis for each problem with regards to the factors with which runtime will change when increasing layers. For the abalone data, as we experimentally determined that the optimal number of hidden nodes was 5 and 6 for 1 hidden layer and 2 hidden layers respectively, we hypothesize a roughly linear increase in runtime as hidden layers increase. For the machine data, as the optimal number of hidden nodes was hypertuned to be 6 and 3 for 1 and 2 hidden layers respectively, we expect no considerable increase in runtime as layers are added. For the forest fire data, we expect to find no considerable increases in runtime as layers are added, since the number of hidden layers was hypertuned to be 4 and 2 for 1 and 2 hidden layers respectively. For the soybean data, where the hypertuned number of hidden nodes was determined to be 9 and 6, we expect no statistically significant change in time complexity. Similarly, for the glass data, as the hypertuned number of hidden nodes was determined to be 10 and 8, we expect to find linear increases in convergence time as a second hidden layer is added. Lastly, for the breast cancer data, as the hypertuned hidden nodes total 8 and 5 for the cases of 1 and 2 hidden layers respectively, we hypothesize no statistically significant changes in runtime. Rigorously, we define linear increases as the continuation of the constant factor which increases when the transition is made from 0 to 1 hidden layers. For any predicted convergence time complexity change, we confirm our hypothesis if the experimental value is within 10% in either direction of the value resulting from our hypotheses.

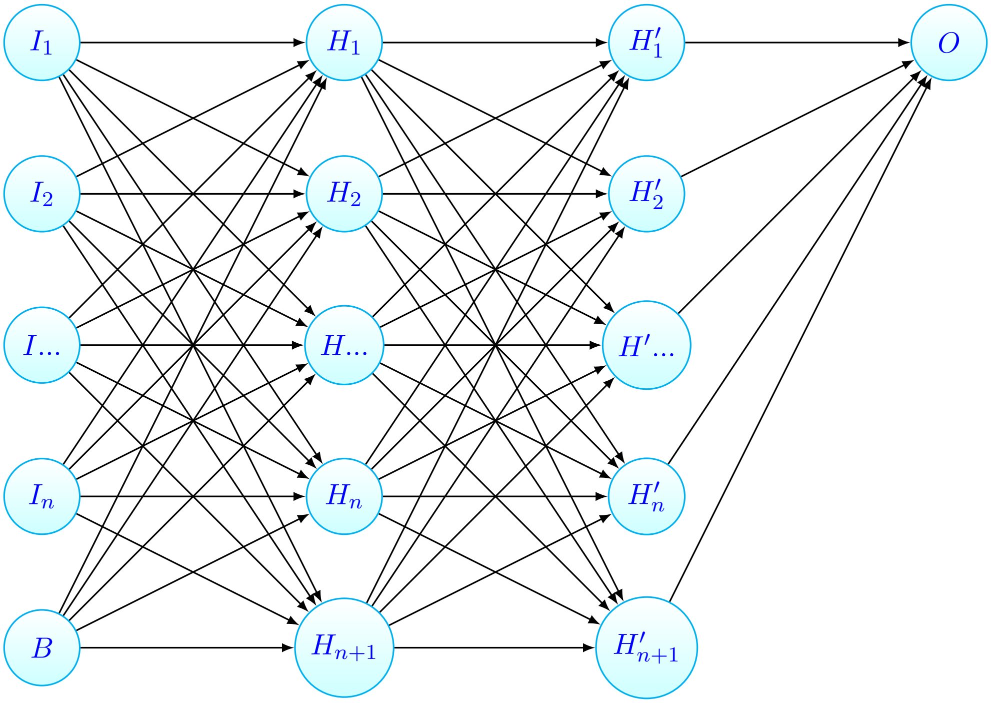
For each of our rigorously value defined hypotheses, we decided to overturn our findings if the experimentally found value was not within 10% of the original hypothesized error value in the positive direction, and if our experimental values were within this threshold or below the proposed value, we would confirm the hypothesis.

2. Methods

In order to test our hypotheses, we ran our model using classification on the Glass, Breast Cancer, and Small Soybean data sets, and we ran our model using regression on the Abalone, Forest Fire, and Machine data. In order to successfully interpret the data, all feature values which were not originally real valued were converted to real number values. The only data set with missing attribute values was the Breast cancer data set, which included '?' values in place of missing data. We chose to assign each of these missing attribute values with the mean value found for that specific attribute in the data. We acknowledge that our solution in this case reveals potential vulnerabilities in our model, but we justify our choice by the large number of entries in the breast cancer data set, along with the presence of a wealth of attributes to delineate only two classes. Otherwise, preprocessing occurred by binning categorical data and assigning bins to real numbers. Furthermore, we used z-score normalization for each numeric attribute value so that the values wouldn't require any special handling. z-score normalization is defined as follows:

$$z = (x - \mu)/\sigma \tag{1}$$

Where a value x is subtracted by the mean μ of a sample and then divided by that sample's standard deviation σ. After all of the datasets had been preprocessed, we implemented the multi layer perceptron, which on a high level takes the form of the following graph:



Where inputs are denoted I_1 through I_n, a bias node is denoted B, the hidden nodes are denoted H_1 through H_{n+1}, and the output node is denoted O. Note that our model implementation was carried out for the cases with both one and two hidden layers, though it is capable of many more- as many as the user specifies. Furthermore, in many cases, we want to train a network with multiple outputs, and though not included in the above figure this implementation of multinets is included in our model. Importantly, each edge between nodes corresponds to a specific weight. These weights aren't included on the diagram for the sake of space, but they are critical for the functioning of the network. Initially, in the first forward run through the network, on a high level, weights are assigned randomly, and then the backpropagation step is what rigorously trains the model. Letting H_j designate each of the weights coming from the hidden nodes into the output, and $w_{k,j}$ denote the weights coming from each of the input nodes, all of the H_j are computed as follows:

$$H_j = \sum_{k=1}^{n} I_k * w_{k,j} + b_n \tag{2}$$

After each H_j is computed in its raw form, it is also passed through an activation function before being sent to the next node in the network. For weights being sent to a hidden node, we use the hyperbolic tangent. Then after all of the hidden nodes are passed, for regression data, we used a simple linear activation function, and for classification data we used the common softmax function when sending weights to the output. Eventually, once all of the outputs are computed on the first pass, back propagation is utilized by computing the derivative of output nodes with respect to the weights. The gradient is then computed moving further backwards through the network, until the process reaches the input nodes. This is computed in the following equations:

$$\Delta w_{ji} = -\eta \frac{\partial Err}{\partial w_{ji}} \text{ such that } \eta \in (0, 1) \tag{3}$$

Where error is computed using squared error for one example at a time, letting d_k denote the target output for unit k and o_k denote the target output for unit k:

$$Err(w) = \frac{1}{2} \sum_{k \in outputs} (d_k - o_k)^2 \tag{4}$$

In practice, this ends up boiling down to the more simple equation where d_j denotes the target output for unit j and o_j denotes the target output for unit j.

$$\Delta w_{ji} = \eta(d_j - o_j)o_j(1 - o_j)x_{ji} \tag{5}$$

After the model is trained and makes its guesses, we determine model performance using 4 different loss functions. These loss functions provide an effective method of determining model performance when certain features of the model are changed. Explicitly, letting N be the number of samples, k be the number of classes, $t_{i,j}$ being the value 1 if sample i is in class j and 0 otherwise, and $p_{i,j}$ being the predicted probability that sample i is in class j, the average cross entropy was computed as follows:

$$crossentropy = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{k} t_{i,j} ln(p_{i,j}) \tag{6}$$

The second loss function we introduced was the 0/1 loss function, which is the ratio of incorrect guesses to total guesses made on the testing data. Denoting the test set T, and letting the number of correct guesses made on the test set by the model be denoted g_c, the 0/1 loss is computed with the simple ratio:

$$1 - \frac{g_c}{|T|} \tag{7}$$

We chose both of the above loss functions with the motivation that both are metrics with importantly opposed qualities in the evaluation of a multi layer perceptron neural net.

Perhaps the most immediately logical evaluation of loss is 0/1 loss, which is a measurement of the ratio of incorrect guesses made in a test set. Importantly, this is indicative of the accuracy of our model, but is not particularly informative of overfitting. Cross entropy however is helpful when evaluating the performance of a model while also keeping in mind the potential of overfitting. By incorporating punishments for a choice being more 'surprising' in that the chances of a correct choice are relatively small, cross entropy is useful in capturing the performance of a model with greater depth than simply counting the number of correct solutions- which in turn provides a weariness for overfitting. Both metrics are important to have an effective model, so both loss functions were chosen in the evaluation of our model on various data.

For regression data, we evaluated the performance of our model using mean absolute error and mean squared error. Mean squared error is computed by simply finding the mean of the squares of the errors. Mathematically, letting Y be the vector of n observed values and $\hat{Y}$ be the vector of n predicted values, mean squared error MSE is computed as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{8}$$

Similarly, mean absolute error MAE is computed by finding the raw average errors by which the forecasts of a model differ from the ground truth.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i) \tag{9}$$

3. Results

By running our model on each of the 6 respective datasets, we were able to compare our hypotheses against experimental findings. Further, we were able to analyze the performance of our model with regards to the time it took to run on varying data, and also to fully view the optimal performance of our model in solving each respective problem after hypertuning had taken place. To begin, we reached the consensus that we would carry out our experimentation with a momentum factor of 0.5, which seemed to generally optimize our overall performance and runtime in the most wide array of problems being investigated in the model. We kept this value consistent throughout experimentation to focus specifically on hypertuning the optimal number of hidden nodes, which seemed generally to play the biggest role in optimizing the model overall. We also chose to maintain 5 epochs throughout the experimentation process, as it seemed to give us relatively steady results without drastically increasing the runtime or seeming to overfit the model. Hypertuning the optimal number of hidden nodes in each run of the algorithm was conducted by simply testing the average performance values from using 1 to the total number of attributes in a given set of data, and then choosing the number which gave us the greatest average performances. We acknowledge the assumption to not consider more hidden nodes than inputs is imperfect, but we believe it provides enough options to get at least close to running our model optimally.

To begin, we present the performance of our algorithm on each data set. Each table holds the values found on ten runs of our model with 0, 1, and 2 hidden layers, after hypertuning the number of nodes per layer.

First, we present our findings on the machine data. In the process of hypertuning the number of nodes in the hidden layer, we worked down from the largest value we tested which was 9. The results for 0, 1, and 2 hidden layers are as follows:

Metric	0 Hidden Layers	1 Hidden Layer	2 Hidden Layers
No. Hidden Nodes	N.A.	6	3
Mean Abs. Err.	2611.02	73.67	95.202
Mean Sq. Err.	3195073.89	19371.32	23266.021
Runtime(sec)	0.17	0.45	0.34

Next we present our findings when running the abalone data in our model. The Abalone data may have 8 inputs, so our hypertuning of hidden nodes started at 8 tested decreasing values until testing 1.

Metric	0 Hidden Layers	1 Hidden Layer	2 Hidden Layers
No. Hidden Nodes	N.A.	5	6
Mean Abs. Err.	2.71	2.18	2.057
Mean Sq. Err.	15.92	8.438	7.7254
Runtime(sec)	3.77	9.22	12.68

And for the final regression data set, we present our experimental findings of the performance of our multi layer perceptron on the Forest Fire data. Interestingly, this was where we found the lowest optimal number of hidden nodes in general, specifically with the case of 2 hidden layers. We tried values under 10 for this data.

Metric	0 Hidden Layers	1 Hidden Layer	2 Hidden Layers
No. Hidden Nodes	N.A.	4	2
Mean Abs. Err.	3.195e+32	426.00	353.71
Mean Sq. Err.	1.84e+15	18.05	17.97
Runtime(sec)	0.49	1.13	0.81

Next, we present our findings with the Soybean data. Here our model performed quite reliably (albeit rather strangely), and plenty of values for hidden nodes could've been used with likely similar results- which showed strong performance for only one hidden layer.

Metric	0 Hidden Layers	1 Hidden Layer	2 Hidden Layers
No. Hidden Nodes	N.A.	9	6
Ave. Cross Entropy	0.00447	0.00589	0.3027
% Accuracy	100%	100%	35.5%
Runtime(sec)	0.146	0.309	0.24

Now we include our findings for the glass data. Particularly with 0 hidden layers, our findings were quite volatile with this data set, with cross entropy in particular ranging quite a lot. Hypertuning started with 9 hidden nodes and worked downward.

Metric	0 Hidden Layers	1 Hidden Layer	2 Hidden Layers
No. Hidden Nodes	N.A.	10	8
Ave. Cross Entropy	0.0172	0.0236	0.0327
% Accuracy	62.54%	66.67%	57.55%
Runtime(sec)	0.50	0.85	0.98

Next we provide our findings when testing our model on the Breast Cancer data. Hypertuning began with 10 hidden nodes and worked downward.

Metric	0 Hidden Layers	1 Hidden Layer	2 Hidden Layers
No. Hidden Nodes	N.A.	8	5
Ave. Cross Entropy	0.0000673	0.00048	0.0088
% Accuracy	96.65%	96.99%	97.14%
Runtime(sec)	0.89	2.32	1.93

4. Discussion

In this section, we mention our experimental findings as they pertain to our previously generated hypotheses, and provide some commentary on possible explanations of our findings- which were quite varied.

For the Abalone data, we predicred a rough mean squared error of 20, and a rough mean absolute error value of 2. With respect to our performance-related hypotheses, our hypothesis was upheld with regards to mean absolute error, but our best performing model (with 2 hidden layers and 6 hidden nodes) significantly outperformed our hypothesis, with an average mean squared error value of 7.7254. With regards to the convergence time of our model, we hypothesized a roughly linear increase from the model with 1 hidden layer vs 2 hidden layers. Experimentally, we found that convergence time increased when adding one hidden layer by a factor of 2.44- so to confirm our hypothesis we require a factor of increase when adding two hidden layers to the model within 10% in either direction of 2.44. In the end, we found an increasing factor of roughly 1.37, which was not large enough to confirm our hypothesis by our predefined standards. This trend in convergence time may be reflective of more of a logarithmic exponential behavior of the model as more hidden layers are introduced.

For the most difficult data set, the forest fire data, we had set our expectations fairly low for the performance of our model. We had predicted a mean absolute error value of roughly 30, and a mean squared error of roughly 1,000. We significantly outperformed our hypothesis, with a best average mean squared error of 353.71, and a best mean squared error value of 17.97. For this problem, it was noteworthy how massive of an impact adding even one layer made for the performance of the model. And in general, despite this being a difficult problem, we were pleased with how well our model performed here. With regards to convergence time, we expected to find no condsiderable changes in runtime when adding a new layer. In practice, we actually found a statistically significant decrease in runtime when transitioning from 1 to 2 hidden layers in the model. (From 1.13 to 0.81, a 28% reduction in convergence time).

For the final regression data set, the machine data, we found that our hypothesized average mean absolute error would be roughly 80, and mean squared error roughly 6400.

In practice, we had mixed results. We were able to confirm our hypothesis with respect to mean absolute error, experimentally finding at the lowest a value of 73.67, which lies within 10% of our hypothesized value. However, our experimentally determined mean squared error was much higher than anticipated, lying at 19371.32. Interestingly, the best performance was found when only 1 hidden layer was used in the model- which came against the preconcieved bias towards complexity in models as a general benefit. With regards to converence time, we had expected to find no considerable change in runtime as we changed 1 hidden layer to 2. We found experimentally that there were statistically significant changes in convergence time, namely a decrease of about 24% in runtime as 2 hidden layers were added, again breaking preconceived expectations that increasing the complexity of a model might guarantee increased runtime. This was likely in large part due to the significant difference between the number of optimal hidden nodes used in 1 hidden layer vs in 2 hidden layers. 6 hidden nodes per layer were found to be optimal in the case of just 1 hidden layer, whereas only 3 hidden nodes were optimal in the case with 2 hidden layers.

In the classification data, we had incredibly varied results, and found that for 2/3 of the classification data sets, our model performed best with 1 hidden layer, and using 2 hidden layers was actually worse than no hidden layers- a result we never found in regression data.

To start, we analyze the glass data with respect to our hypothesis on the performance of our model. We hypothesized a 0/1 loss value of roughly 0.20, corresponding to guessing correctly 80% of the time, and a cross entropy value of roughly 0.1. In practice, we found that our 0/1 loss hypothesized value could not be upheld (finding experimentally a 0/1 loss on average in the best case of 0.33), but that we outperformed our expected cross entropy value, gaining in the lowest case an average cross entropy of 0.0172. Allowing us to partially reject our hypothesis. In terms of runtime, we had hypothesized a linear change in runtime when adding the second layer based on the number of hidden nodes, which didn't quite hold experimentally. (Adding 1 hidden layer increased convergence time by a factor of 1.7, adding 2 hidden layers increased convergence time by a factor of 1.15).

For the soybean data, we hypothesized a 0/1 loss value of roughly 0.05, and an average cross entropy value of 0.001. In practice, we performed better in the best case for our model, gaining a 0/1 loss value of approximately 0 and an average cross entropy of roughly 0.0236 for the model with 1 hidden layer. Interestingly, our performance dropped significantly when adding a second hidden layer, and in this case underperformed in terms of the hypothesis. With regards to convergence time, we had anticipated no significant change in the time it took to converge with 2 hidden layers as opposed to 1. In practice, we found that adding a 2nd hidden layer came with a statistically significant decrease in convergence time.

Lastly, for breast cancer, we hypothesized a 0/1 loss value of roughly 0.02, and an average cross entropy value of roughly 1. We found in the best case a value just shy of the 0/1 loss value (0.028), which was just slightly too far to be deemed close enough to the specified value to confirm the hypothesis. However, we found a cross entropy value significantly lower in all cases than what was hypothesized, leading to inconclusive results with regards to our initial thoughts towards the performance of our model on this problem. With respect to convergence time, we again had thought that the model would converge with no statistically significant difference in time between 1 and 2 hidden layers. Experimentally, we found that having 2 hidden layers allowed for a statistically significant decrease in convergence time.

5. Summary

In running our feedforward multi-layer perceptron on 6 diverse sets of data, we came away with a broader understanding of the performance of our model with respect to hypotheses both regarding performance and convergence time. Namely, we found quite mixed performances, and in general were not able to overwhelmingly uphold the majority of our hypotheses. Broadly speaking, we had expected that more hidden layers would mean improved performance of our model, which in numerous examples proved to be untrue. However, with respect to performance, our hypotheses were more generally upheld in problems of regression than in those of classification. Alternatively, when examining our hypotheses regarding convergence time, we found quite reliably that our expected results were overturned. Generally, our hypotheses seemed to reflect a bias towards greater computational expense in adding additional hidden layers, which proved not to be the case in numerous examples. To our surprise, adding additional hidden layers actually regularly contributed to decreases in the amount of time it took for our algorithm to converge. Further investigating even deeper neural networks is certainly an exciting area of future work. All in all, our findings are revealing of the wide possibilities- and limitations- of networks such as the multi-layer perceptron, and refining such models will undoubtedly be immediately relevant for the forseeable future in the world of machine learning.