Journal of Machine Learning Research 1 (2020) 1-48 Submitted 4/00; Published 10/00

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 5% in either direction of the hypothesized value. In terms of convergence time, we specified a number of iterations predicted iterations and again 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 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 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 5. 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

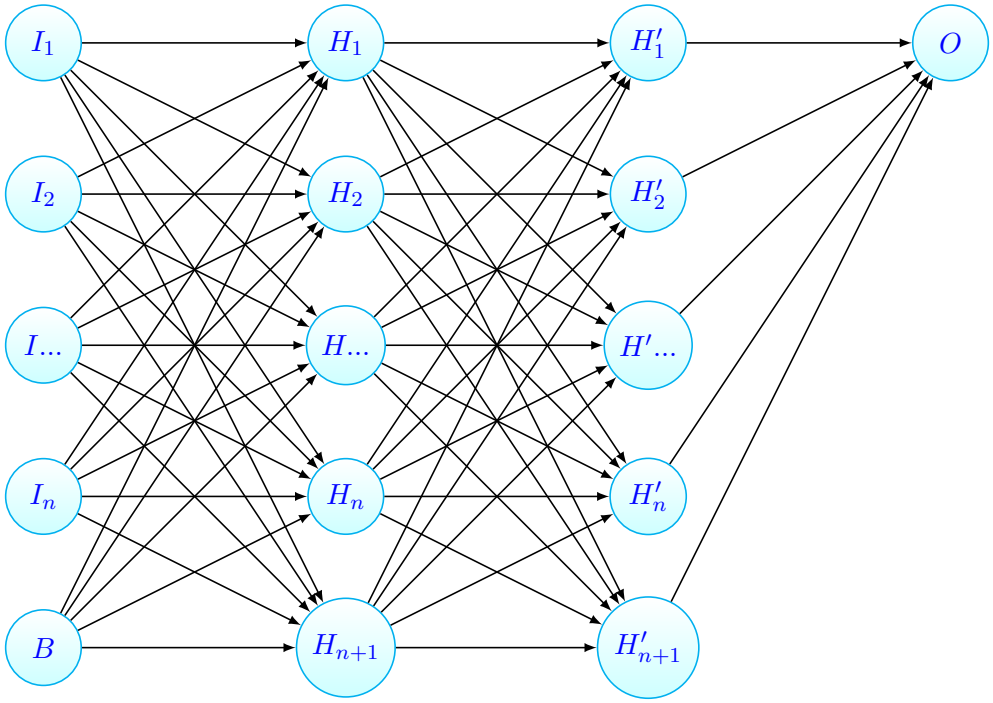
loss value of roughly 2. In terms of entropy, this was harder for us to predict. Since there were only two classes, a higher amount of entropy may be expected than in other data sets, so we predicted an average cross entropy value of roughly 1.

We also provide a hypothesis with respect to the amount of time each data set would take to converge in our model. 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. We hypothesized that the number of different feature variables in a data set would correlate strongly with the runtime of the model, since our multi layer perceptron is much more combinatorially expensive as more feature values are introduced. Thus, we predicted that our glass data would run fairly slowly, as we thought the total of 10 attribute values would cause weights to converge rather slowly, implied by our hypothesized value of roughly 5 seconds per run. We predicted too that the machine data, with its 10 attributes, would also converge rather slowly, similarly falling around 5 seconds per run. The soybean data, with the most attributes (35), we thought would take the longest to converge, falling at roughly 10 seconds per run. The forest fire data, with 13 attributes, we hypothesized would run in roughly 6 seconds. The breast cancer data, with only 8 feature values, we expected to run in only 3 seconds. Similarly, the abalone data we hypothesized would run also in roughly 3 seconds, with its 8 attributes. As a general rule of thumb, we expected the number of attributes to correlate with how long it would take the weights to converge in each dataset, as the number of attributes are what is feeding into the network to contribute to weights in the first place.

For every single one of our hypotheses, we decided to overturn our findings if their value was not within 10% of the original hypothesized error value. That is, 5% in either the positive or negative direction of what we predicted.

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 simply coerced all missing data to 'NaN' values, effectively eliminating them from being considered in our model. All that this meant was that these particular feature values didn't end up being used in the model. We acknowledge that our solution in this case reveals potential vulnerabilities in our model, but we justify our choice to exclude this data 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. 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 more. 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:

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

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:

$$\Delta w_{ji} = -\eta \frac{\partial Err}{\partial w_{ji}} \text{ such that } \eta \in (0, 1) \tag{2}$$

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:

$$Err(w) = \frac{1}{2} \sum_{k \in outputs} (d_k - o_k)^2 \tag{3}$$

In practice, this ends up boiling down to the more simple equation

$$\Delta w_{ji} = \eta(d_j - o_j)o_j(1 - o_j)x_{ji} \tag{4}$$

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:

$$crossentropy = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{k} t_{i,j} ln(p_{i,j}) \tag{5}$$

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:

$$1 - \frac{g_c}{|T|} \tag{6}$$

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 percent 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:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{7}$$

Similarly, mean percent error MPE is computed by finding the average percentage errors by which the forecasts of a model differ from the ground truth. Letting a_t be the ground truth value of a prediction, f_t be the actual value of the prediction, and n being the number of different times a forecast is made, this is rigorously computed as follows:

$$MPE = \frac{100}{n}\sum_{t=1} n\frac{a_t - f_t}{a_t} \tag{8}$$

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 most effectively hypertune many of the parameters within our model.

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. The number of nodes per layer was tuned by beginning with an equivalent number of nodes to the number of inputs, and then reducing this number until the performance average didn't improve. We acknowledge the assumption to not consider more hidden nodes than inputs is imperfect, but for practical purposes related to mostly to time complexity it is the approach we chose. We also found generally the best performances using roughly 0.5 as a momentum factor, so this value was included for momentum throughout our experimentation.

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 8. 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.	8	6
Mean Abs. Err.	3.195e+32	87.546	95.202
Mean Sq. Err.	2.453e+59	24797.948	23266.021
Runtime(sec)	6.643	22.380	23.110

Next, we present our findings with the Soybean data. Importantly, due to the large number of input variables for any given node in the model under Soybean data, (and because roughly half of these inputs could be considered redundant to some degree) we limited our hypertuning to only have 10 total hidden nodes, as performance was effectively universally strong on this data set and changes in performance were incredibly minute.

Metric	0 Hidden Layers	1 Hidden Layer	2 Hidden Layers
No. Hidden Nodes	N.A.	7	9
Ave. Cross Entropy	0.000128	0.000105	0.0000846
% Accuracy	100%	100%	100%
Runtime(sec)	6.043	11.012	13.819

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 wildly from roughly 0.04 to 1.7e-08. Hypertuning started with 9 hidden nodes and worked downward.

Metric	0 Hidden Layers	1 Hidden Layer	2 Hidden Layers
No. Hidden Nodes	N.A.	8	9
Ave. Cross Entropy	0.0172	0.0236	0.0327
% Accuracy	62.54%	67.11%	60.95%
Runtime(sec)	19.26	29.056	39.19

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.	5	8
Ave. Cross Entropy	0.0000673	0.00048	0.0088
% Accuracy	96.65%	96.99%	97.14%
Runtime(sec)	35.71	65.57	115.15

4. Discussion

Todo

5. Summary

Todo