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

An Inquiry Into Evolutionary Algorithms & 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 evolutionary algorithms on a feedforward multi layer perceptron with back propagation- comparing evolutionary training to our implementation of back propagation. 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 average cross entropy. For datasets where regression was performed, mean squared error was the chosen metric to evaluate our model. In these two differing settings, we used each respective evaluation of performance as our value indicative of fitness. 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 specified rough percentage changes in fitness from start to finish for our hypotheses with respect to each evolutionary model, 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 between different models 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 in an attempt to provide insights into the performance of evolutionary algorithms when applied to 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. We chose to evaluate our evolutionary algorithm's performance on each set of data in multiple ways. Namely, we focused on the convergence rate and % difference in loss when compared to back propagation (either positive or negative). Fitness is measured as an average loss throughout the entire neural network as data is fed in. Convergence rate is measured in the number of seconds it takes for our evolutionary algorithms to begin and then terminate. For each metric, if our experimental values are within 10% in either direction of the hypothesized values, we choose to confirm the hypothesis.

For the abalone data, we hypothesized that generally, our model would perform relatively well in this setting, and most members of the population would start with a relatively high fitness. As such, we thought that working more globally towards an optimum would be favorable for evolutionary algorithms, and particle swarm optimization would perform especially well. Rigorously, we hypothesized roughly a 10% decrease in fitness values (which corresponds to an improvement in fitness) when compared to back propagation for our genetic algorithm, roughly a 20% decrease for our differential evolution algorithm, and roughly an 30% increase for particle swarm optimization. We anticipate that our time of convergence will be roughly 3 seconds for GA and DE, and roughly 3.5 seconds for PSO.

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. Thus, we expected a focus on global optimums to be pertinent, and we expected PSO to find the most dramatic decrease in fitness values, of roughly 15%. We expected our genetic algorithm to represent a roughly 10% decrease in fitness values, and our differential evolution to be roughly the same as back propagation. In terms of runtime, we expected genetic algorithms to perform the fastest, then differential evolution, and finally particle swarm to perform the slowest, each of these with statistical significance. (At least 10% slower/faster than the other algorithm specified).

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. 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. As such, we thought that bias in the model towards elite members would be preferable. Differential evolution, with the largest bias towards the fittest members, we thought would be the most successful here. Specifically, we anticipated a 50% decrease in fitness values from differential evolution, a roughly 40% decrease from genetic algorithms, and a roughly 10% decrease from particle swarm optimization. Each model we expect to take a bit longer to converge, as we expect genetic algorithms and differential algorithms to converge roughly around 8 seconds, and particle swarm optimization roughly around 10. We 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. We thought that since there were a large amount of classes, it might make sense for particle swarm optimization to perform particularly well. We hypothesized a roughly 10% decrease from PSO, a roughly 10% decrease from GA, and a roughly 5% increase from DE. In terms of convergence time, we thought PSO might converge at roughly the same speed as GA, each at around 5 seconds. We thought that DE might converge a little slower, at roughly 7 seconds.

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. We thought that preferring the most elite members in a population here would be preferable, since there are relatively few classes to worry about, and moving towards them would probably happen the quickest favoring those with the highest fitness. As such, we thought DE would have the highest percent decrease, of roughly 15%, and GA would follow with roughly 5%, and PSO would bring up the rear with a 5% or so increase. We thought that because of these somewhat clear cut distinctions, DE and GA would converge the fastest, each at roughly 3 seconds. PSO we thought would converge at roughly 4 seconds.

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. In this kind of setting, we thought that the most elite members of the population would be similarly highly favored as in the soybean data. However, we also thought that normal backpropagation would be quite effective. Thus, we anticipated a 25% reduction in fitness (loss) values from DE, and roughly 10% from GA. We didn't think particle swarms would be particularly helpful here, and expected an increase of roughly 30% in fitness values from PSO when compared to back propagation. In terms of runtime, we also thought that DE would be really fast, roughly 2 seconds per run, with GA at roughly 3 seconds per run, and PSO at roughly 6 seconds.

Wholistically, we expect to find that the more complex of differentiation (different possible categories, huge variation in function values) a neural net sets out to do, the better more globally focused algorithms like PSO will do. We expect generally for DE to perform better when the delineation between different classes is more clear. Generally, we anticipate GA will be somewhere in between the two, but nearer to DE.

Note that we chose to measure the performance of evolutionary algorithms with respect to the percent change in their fitness values when comparing the outputs of two different methods. In doing so, we're able to categorize the performance of these algorithms in how effectively they may alter a population to be more fit after a fixed number of generations-based most specifically just on the holistic performance of the algorithms rather than the performance of our model on generic data.

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, with back propagation. As this implementation is not at the heart of this experiment, for details of the exact equations we used and our exact implementation of the MLP, see our programming assignment 3.

After the model is trained and makes its guesses, we determine model performance using 2 different loss functions- which we deem indicative of the evolutionary fitness of that particular neural net. 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:

$$crossentropy = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{k} t_{i,j} ln(p_{i,j}) \tag{2}$$

Cross entropy 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. We chose to use this loss function because we thought it would be more important in guaging the efficacy of our evolution- without too heavily overfitting data.

For regression data, we evaluated the performance of our model using 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:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{3}$$

Both metrics were used as fitness values for a given neural network in the context of evolutionary algorithms. After the neural network had been initialized and evaluated in this manner, we incorporated the Genetic algorithm, differential evolution algorithm, and particle swarm optimization. Though we won't include their pseudocode here, as the algorithms are well known and documented, we will provide our hypertuning of specific parameters within each algorithm, and justification for our hypertuning.

1. Particle Swarm Optimization:

   Particle Swarm Optimization has hyper parameters commonly noted as $\omega, swarmsize, c_1, c_2$.

2. Differential Evolution:

   Differential Evolution is especially customizable.

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.