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. For all algorithms, a population of 40 individuals was used- as this seemed a sizable enough population to experience serious, meaningful evolutionary changes without drastically harming the runtime.

1. Particle Swarm Optimization:

   Particle Swarm Optimization has hyper parameters commonly noted as ω, swarmsize, c_1, c_2. ω affects the strength of the intertia term when calculating velocity. The value of ω is between the range 0 and 1. A lower ω value allows for more dynamic movement of particles with in the swarm allowing for finding a solution for quickly. However, by having a small interia term the alogorithm is at a higher risk of getting stuck in local optimum. After tuning we found $\omega = .8$ to be the optimum value that prevented getting stuck in local optima, but did not overpower the cognitive and social terms. The hyperparameters c_1, c_2 act as coefficents for the social and cognitive terms respectively. We found the optimally tuned values of these to be dependant on the topology used for the swarm. In a gBEST smarm we found it optimal values to be $c_1 = 1.2, c_2 = .6$. In this case we have c_1 equal to twice the value to c_2 in order to comboat selection pressure applied by high fitness of the social term in a gBEST topology. When using an lBEST topology we tuned $c_1 = .8, c_2 = .8$. The swarm size hyperparamter represents the number of particles in the swarm. We tuned this value to 10 because we found it was sufficiently large, while still being small enough to keep runtime relatively low.

2. Differential Evolution:

   Differential Evolution is especially customizable. Namely, in our implementation, we iterate over all vectors in a population $\mathbb{P}$, and call each vector the targed vector, denoted $\mathbb{X}_j$. Then we also choose three other distinct vectors $\{\mathbb{X}^1, \mathbb{X}^2, \mathbb{X}^3\} \in \mathbb{P}$. From these vectors, we compute a trial vector $\mathbb{U}_j$ as follows:

$$\mathbb{U}_j := \mathbb{X}^1 + \beta(\mathbb{X}^2 - \mathbb{X}^3) | \beta \in [0, 2] \tag{4}$$

The above constitutes the mutation operation for differential evolution. Below, we now show the crossover operation as follows:

$$\mathbb{X}'_j = \begin{cases} \mathbb{X}_{ji} \text{ if } rand(0,1) \leq Pr \\ \mathbb{U}_j \text{ else} \end{cases} \qquad Pr = 0.5 \tag{5}$$

Where $\mathbb{X}'_j$ denotes the crossed over $\mathbb{X}_j$ and Pr is a tunable parameter for a crossover probability. We chose to stick to 0.5 for this value, as we seemed to get the best performance with this setting in trials.

We then check to see if the target vector has higher fitness than the crossed over vector. If so, we just keep the original target vector. Otherwise, we replace the target vector with the vector $\mathbb{X}'_j$ which came from the combination of our mutation and crossover operations. As a result, DE has a tendency to be a bit elitist. In the notation common to the field, we chose to use the implementation denoted "*DE/curr/2/Bin*" Where we conducted DE choosing the current organism while iterating through the population, with a total of 2 difference vectors in the mutation, and where we used binomial crossoveri.

3. Genetic Algorithm: In our genetic algorithm implementation, we iteratively choose pairs in the population randomly, comparing their fitness. Whichever member of the pair has greater fitness, we add to the selected population. We then conduct the crossover operation, doing so uniformly, and on another randomly designated pair, iteratively over the entire population. More rigorously, honoring some crossover rate $\mathbb{C}$ value in $[0, 1]$, we choose to do gene crossover whenever a random variable on the same interval is less than $\mathbb{C}$. This is done by randomly assigning two children genes (weights) corresponding to a random mix of the weights of the original pair (parents). If the random variable is not less than $\mathbb{C}$, we just keep the randomly chosen pair the same for the next generation of the genetic algorithm. Finally, mutation occurs uniformly based on random number generation within some range, to be added or subtracted to a given gene, based on a mutation rate. We found experimentally that retaining a mutation rate and a mutation amount of roughly 0.2 seemed to add in the perfect amount of random genetic diversity into the population, without introducing too much randomness to outweigh evolutionary processes. However, for the genetic algorithm we found it optimal to use a much higher frequency of crossover. We rationalize this decision with the rationale that adding in crossover for selected individuals introduces genetic diversity- but only from the already determined most fit individuals. Which is extremely beneficial for a population in the long term.

3. Results

We now provide our results when running our various implementations of genetic algorithsm on the named sets of data. We compare these results alongside other genetic algorithms, as well as against standard back propagation of our multi layer perceptron. We conduct all evolutionary algorithms on a population of 40 individuals. Here we record the final fitness value provided by each of our algorithms, which is an average fitness value across the entire population in that generation.

We begin by providing the performance of our model on the breast cancer categorization data. Our results with respect to runtime and fitness measured in cross entropy are as follows:

Metric	GA	DE	PSO	Back Propagation
Fitness (entropy)	0.118	0.208	0.205	0.129
Convergence(sec)	2.914	2.899	59	N.A.

Next, we provide our findings on the glass data:

Metric	GA	DE	PSO	Back Propagation
Fitness (entropy)	1.225	1.729	1.177	
Convergence(sec)	1.140	0.983	260	N.A.

For the final set of categorical data, we investigate the small soybean data:

Metric	GA	DE	PSO	Back Propagation
Fitness (entropy)	0.741	1.184	0.630	1.133
Convergence(sec)	0.237	0.242	160	N.A.

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.

5. Summary