Comparison of Multiple Evolutionary Approaches to Backpropagation on Training Neural Networks

Clint Cooper CLINTGCOOPER@GMAIL.COM
Department of Computer Science
Montana State University
Bozeman, MT 59715, USA

Emily Rohrbough EMILYROHRBOUGH@YAHOO.COM
Department of Computer Science
Montana State University
Bozeman, MT 59715, USA

Leah Thompson LEAH.THOMPSON@MSU.MONTANA.EDU
Department of Computer Science
Montana State University
Bozeman, MT 59715, USA

Abstract

This paper analyzes four approaches to training a feedforward neural network: (1) backpropagation, (2) genetic algorithms (GA), (3), $(\mu + \lambda)$-evolution strategies (ES), and (4) differential evolution (DE). The three evolutionary approaches, GA, ES, and DE, are used to compare evolutionary approaches to the more traditional backpropagation method. All four training methods were applied to a multilayer perceptron (MLP) neural network and rate of convergence and accuracy were measured. The MLP neural network was used for classification over ten data sets from the UCI Machine Learning Repository (Bache and Lichman, 2015). The ten data sets were used as training and testing data for the neural network with each of the four training methods. This resulted in forty total tests performed for evaluation. It was predicted that the differential evolution method would outperform all other approaches in terms of accuracy, followed by backpropagation, ES, and GA respectively. However, all evolutionary approaches are expected to outperform backpropagation in terms of rate of convergence. The results showed that there was no true winner among the four algorithms, however, some algorithms had better performance or convergence rates over different datasets. Details and statistical analyses are discussed in the results section.

Keywords: Evolutionary Algorithms, Multilayer Perceptron Neural Networks, backpropagation, Evolutionary Strategy, Differential Evolution, Genetic Algorithm

1. Introduction

Evolutionary algorithms (EAs) are weak methods inspired by Darwin's theory of evolution by natural selection. A practical use for these algorithms can be applied to neural networks to find optimal solutions to problems such as function approximation or classification (Moriarty et al., 1999). EAs do not exploit domain knowledge (Sheppard, 2015a), but utilize the knowledge of a population of potential solutions by applying selection, operators, and replacement to search for better solutions (Moriarty et al., 1999). EAs select fit parents from the population to use various mutation and cross product operations to produce offspring that are evaluated and then inserted into the existing population through replacement methods (Sheppard, 2015a). The solutions that remain will ideally be the best, i.e., survival of the fittest. The three evolutionary approaches this project focused on were evolution strategy, differential evolution, and genetic algorithm (Sheppard, 2015b). These algorithms were used to train the weights of a pre-defined multilayer perceptron (MLP) neural network (Sheppard, 2015b). The aim of the project was to compare the convergence rate and performance of each algorithm against a traditional backpropagation weight training algorithm (Sheppard, 2015b).

2. Problem Statement and Hypothesis

A standard multilayer perceptron neural network implementation utilizes the backpropagation algorithm to update the weights between each node in the network. The backpropagation algorithm is known to be slow to converge, although it is effective when provided with enough training data, which inspired this project (Sheppard, 2015a). This project explored the use of evolutionary approaches for training a pre-defined neural network by implementing the genetic algorithm, differential evolution, and evolution strategy approaches. The convergence and performance of each algorithm was compared against backpropagation to determine the success of the evolutionary training method.

Given the nature of these four algorithms, the algorithms are expected to train the MLP network with similar performance and convergence rates. In terms of performance, we expect differential evolution to be the most accurate and outperform the other three algorithms due to its ability to explore the search space more efficiently than the other methods. Backpropagation is expected to perform second best because the gradient pushes towards the correct answer similar to a hill climber approach. The evolution strategy will come in behind backpropagation, but will outperform the genetic algorithm as it uses a self-adaptive mutation method, where the genetic algorithm has no self-adaptation. Both of these methods use a "random walk" approach through the search space, which may adversely affect the overall performance in comparison to backpropagation and differential evolution.

Additionally, in terms of convergence, the three EAs are expected to converge in fewer generations than the traditional backpropagation training method. The backpropagation algorithm is consistently accurate when given enough iterations to train the network. The evolutionary algorithms, however, explore the search space by exploiting knowledge of previously known good solutions, which allows the algorithms to move towards the correct answer quicker. Of the three EAs, differential evolution is expect to converge quickest, followed by evolution strategy. Genetic algorithm will converge the slowest of the three as it uses no dynamic learning or self-adaptation methods. This prediction is based on search methodology implemented for each algorithm, discussed in Section 4.

3. Algorithms Implemented

The multilayer perceptron (MLP) feedforward neural network developed for Project 2 was reused allowing for the training algorithm to be specified at run-time. Backpropagation, evolution strategy, differential evolution, and genetic algorithm are four training algorithms that could be specified for the MLP neural network. A brief description of an MLP and backpropagation are given below, followed by a detailed description of each evolutionary algorithm.

3.1 Multilayer Feedforward Neural Networks

A multilayer feedforward neural network that uses the backpropagation learning function is a biologically inspired algorithm that consists of a directed graph organized as input, hidden, and output layers (Mitchell, 1997)(Kurt Hornik and White, 1989). The number of hidden layers and nodes per hidden layer are design parameters that are set at run-time. Each node is connected to all other nodes in the next layer with an associated weight which is used to evaluate the input using an activation function. For this project, the sigmoid activation function was used (Sheppard, 2015a). During the training session, once the final layer is reached, the squared error of the output is determined against the expected output. If the error is not within a specified threshold, the neural network will use a training method to update the weight for every node, starting in the output layer.

3.2 Backpropagation

MLP networks commonly implement gradient descent backpropagation to determine the updated weight (Sheppard, 2015a). The backpropagation learning function is also referred to as stochastic gradient descent learning (Engelbrecht, 2007). It takes the output layer values obtained during the feedforward pass through the neural network, determines the squared error of the output, and performs backward propagation on the hidden layer if the error exceeds a specific threshold (Sheppard, 2015a). The squared error on the output layer is the summation of the delta errors of each output node, determined by comparing the calculated outputs, o_x, to the desired outputs, d_x(Kurt Hornik and White, 1989):

$$Err(X) = \sum_{x}^{X} (d_x - o_x)^2.$$

If the error threshold is exceeded, backward propagation is necessary and gradient decent is applied to the delta errors to update the weights of each node in the network (Sheppard, 2015a). Through backpropagation, the calculated errors from the output layer are sent towards the input layer. The hidden layer's weights are simultaneously updated based on the propagated error signal and a user-defined momentum value, M, ranging from 0 to 1. (Kurt Hornik and White, 1989). The weights of the output layers are updated as follows (Sheppard, 2015a):

$$w_{ji} \leftarrow w_{ji} + (1-M)\eta\Delta w_{ji} + M(w_{ji} - w'_{ji})$$

where w'_{ji} is the previous output layer weight and Δw_{ji} depends on the derivative of the sigmoid activation function:

$$\Delta w_{ji} = x_{ji}(d_j - o_j)o_j(1-o_j).$$

The hidden layers are updated as:

$$w_{kj} \leftarrow w_{kj} + (1-M)\eta\Delta w_{kj} + M(w_{kj} - w'_{kj})$$

where w'_{kj} is the previous hidden layer weight and Δw_{kj} is:

$$\Delta w_{kj} = x_{kj}o_j(1-o_j)\sum_{k\in ds(k)} w_{kj}o_k(1-o_k).$$

3.3 Genetic Algorithm

Genetic algorithm (GA) is the most commonly used evolutionary algorithm (Sheppard, 2015a) that was first proposed by Alex Fraser in 1957 (Fraser, 1960). The genetic algorithm strictly follows the EA processes of selecting parent individuals from a population to apply mutation and crossover operators to explore the search space by creating offspring that will be integrated into the population.

A population, P, of constant size, NP, is randomly initialized where the interval -0.5 to 0.5 was used in correspondence to the weight range of the MLP network. Each individual is then evaluated to determine the fitness. The fitness function for an individual in the population, x_i, that was used for evaluation was relative error:

$$Fitness(x_i) = \sum_{j=0}^{|D|} \frac{|d_{x_{i,j}} - o_{x_{i,j}}|}{d_{x_{i,j}}}.$$

where $|D|$ is the number of input-output pairs. Once an initial population is created and evaluated, parent selection, creation of offspring, and replacement are followed to create new generations of fitter solutions (Sheppard, 2015a).

In the GA, individuals in the population are selected to act as parents. There are several selection methods such as elitism, fitness proportionate, rank based, tournament, and more. The tournament-based selection is typically preferred because it helps avoid the loss of diversity as children are generated

(Sheppard, 2015a). Tournament selection randomly selects a number of unique participants from the population and then orders the subset by their fitness. From this ordered subset, the top victors were used as the parents, referred to as the parent collection. Both number of participants and number of victors are tunable parameters.

The parent collection then implements the EA's operators of crossover and mutation. The crossover method implemented in the GA was the uniform crossover technique where each element in the offspring is from one parent or the other, determined by a randomly generated number in the range of 0 to 1 and comparing it to the crossover rate, CR. This ensures that a large collection of children is generated and that no parent reproduces asexually. The children can be referred to as the children collection.

Once the children collection is created, a uniform mutation is applied on each child by potentially altering each element in in the genome. This is applied by randomly generating a number between 0 and 1 and comparing it to the mutation rate, MR, such that if this random number is less than MR, a random value between -0.5 and 0.5 is added to the current element's value.

Once all the children have been generated and mutated, a steady-state population replacement selects the fittest children and replaces (kills) the least-fittest individuals in the population. This process is continued until an individual is found within the specified threshold or a maximum number of generations is reached.

3.4 Differential Evolution

Differential evolution (DE) is a simple, population based, stochastic function (Stron) that uses real valued parameter vectors to optimize functions without using the gradient (Fleetwood, 2004)(Arunachalam, 2008). The DE algorithm follows the same evolutionary algorithm procedures of selection, operations and replacement that were inspired by natural selection to search for a better solution for each member of the population (Sheppard, 2015a). However, the DE procedures are executed in the following order: initialization, evaluation, mutation, recombination, selection (Fleetwood, 2004).

First, the population, P, consisting of NP real-valued vectors are randomly instantiated, where NP is a user-defined population size (Sheppard, 2015a). The vectors in the population were randomly assigned within the range -0.5 to 0.5 with respect to the weight range in the MLP neural network. Once P is initialized, each vector, $x_i \in P$, is evaluated using a fitness function to assign a fitness score or optimization cost to determine the effectiveness of the solution (Sheppard, 2015a). The fitness function utilized in training the MLP network was the relative error, such that the error was minimized to find the best solution:

$$Fitness(x_i) = \sum_{j=0}^{|D|} \frac{|d_{x_{i,j}} - o_{x_{i,j}}|}{d_{x_{i,j}}}.$$

where $|D|$ is the number of input-output pairs.

In differential evolution, each vector in the population explores the search space by combining its own positions and other existing vectors in the population (Arunachalam, 2008). As a new position is explored, its new fitness score is compared against its previous position's score to determine which position provides a better candidate solution vector (Sheppard, 2015a). These selection, mutation, and recombination processes are repeated for each vector in the population (Sheppard, 2015a). This is continued until the maximum number of generations is reached or a sufficient vector is found that has a fitness within the specified neural network threshold.

The selection, mutation and recombination methodology utilized in the DE algorithm are defined as DE strategies (Arunachalam, 2008). DE strategies are represented by the $DE/x/y/z$ notation, where x represents how the population vectors are selected, y is the number of differential vectors used in mutation, and z is the type of crossover performed (Arunachalam, 2008). This means each vector selected from the population is used in the mutation and recombination methods. The mutation and recombination used in the differential evolution algorithm are what makes the differential algorithm unique.

Each vector in the population undergoes mutation to expand the search space (Fleetwood, 2004). It requires a donor vector, x_i, defined by x in the DE notation. Additionally, the y in the DE notation determines if one or two weighted difference vector(s) will be used to perturb the population (Stron)(Arunachalam, 2008). The standard mutation strategy utilizes one difference vector, so two vectors in the population that are not the donor vector are randomly selected as x_j and x_k. Once the difference vector is created, it is multiplied by beta mutation rate, β, to create the weighted difference vector. This vector is then subtracted from the donor vector, resulting in a trial vector, u_i, defined as (Fleetwood, 2004):

$$u_i = x_i + \beta(x_j - x_k)$$

If two difference vectors are required, four vectors are randomly selected, multiplied by the beta mutation rate and then subtracted from the donor vector (Sheppard, 2015a). The standard mutation method of one difference vector was used in the DE implementation.

There are two recombination methods used in the DE algorithm, binomial crossover and exponential crossover, defined by z in the DE strategy (Sheppard, 2015a). The donor vector, x_i, and the trial vector, u_i, are required and are crossed against one another using a crossover rate, CR, to create the offspring vector, x_i' (Ali et al., 2009). In binomial crossover, each element of the two vectors are considered for the offspring vector by generating a number between 0 and 1 and comparing it against CR (Sheppard, 2015a). However, to guarantee at least one element of the mutated trial vector is carried forward, one element of u_i is randomly chosen for the offspring vector before the remaining elements are considered (Fleetwood, 2004). For exponential crossover, the first element for the offspring vector is randomly chosen from the trial vector and elements are continuously added from the trial vector until the generated number between 0 and 1 is less than CR Sheppard (2015a). Once this condition is met, the offspring vector receives the remaining elements from the donor vector. Stron and Price stated both recombination methods are equally as effective (Stron), thus binomial crossover was used in the implemented differential algorithm for this paper. The overall DE strategy defined for this project was DE/current/1/bin.

3.5 $(\mu + \lambda)$-Evolution Strategy

It has been rationalized that if a biological process can be optimized by evolution, with evolution being a biological process itself, then evolution can optimize itself (Engelbrecht, 2007). In other words, the evolution of evolution, which is the idea behind the evolution strategy (ES). Each individual in an ES population is represented by its current genetic makeup, which consists of a vector of real values, and a vector of strategy parameters. The strategy parameters are used to model the behavior of an individual within their given environment. Each individual is defined as:

$$X_i(t) = (x_i(t), \sigma_i(t))$$

where $x_i(t)$ is the vector of real values randomly initialized in a range of -0.5 to 0.5. $\sigma_i(t)$ is the vector of strategy parameters, initialized as standard deviations of $\chi(t)$ (Engelbrecht, 2007).

For the $(\mu + \lambda)$-ES algorithm, $(\mu + \lambda)$ refers to the ES generating λ offspring from μ parents, where $1 \leq \mu \leq \lambda \leq \infty$. This method implements elitism to guarantee that the fittest parents survive through to the next generation (Engelbrecht, 2007). The initialization of the vector of real values, the tournament-based parent selection, and evaluation are implemented as described for the GA.

Crossover in the ES is applied to the vectors of strategy parameters and the vectors of real values. Local crossover was implemented for this project, where ρ parents was set to two. This means two randomly selected parents were used to create a single offspring by using discrete recombination where the actual values of the parents are used to construct the offspring (Engelbrecht, 2007). A crossover rate is used similarly to the GA where a random value between 0 and 1 is selected and compared to the crossover rate.

Mutation in an individual is then applied to improve the fitness of the individual. This is accomplished through self-adaptive updating of the strategy parameters with an elliptic distribution. This updating of the strategy parameters is done using the following lognormal equation:

$$\sigma'_{ij}(t) = \sigma_{ij}(t)e^{\tau' N(0,1)+\tau N_j(0,1)}$$

where $N(0,1)$ is a (0, 1) normally distributed random scalar and τ and τ' are the learning parameters, also referred to as rates of self-adaptation, where

$$\tau' = \frac{1}{\sqrt{2n_x}}, \quad \tau = \frac{1}{\sqrt{2\sqrt{n_x}}}$$

and n_x is the number of deviation parameters used. Each of the real values is then mutated as:

$$x_{ij} = x_{ij} + \sigma_{ij} * N(0,1)$$

A mutation rate is used to compare a randomly selected value between 0 and 1 to the mutation rate. If the random value is less than the mutation rate, the mutation procedure described above will be applied.

The procedure of selection, crossover, mutation, and evaluation is repeated until the maximum number of generations is reached or a sufficient vector of real values is found that has a fitness within the specified neural network threshold.

4. Experimental Approach

The four algorithms used to train the MLP neural network were tested on 10 data sets from the UCI Machine Learning Repository. The 10 data sets were chosen based on the following conditions: (1) must be a classification data set, (2) must have 6-12 attributes, (3) must have less than 6000 instances and (4) must have one classification per set of attributes. These constraints were placed on the choice of data sets in hopes to compile test results with similar performance and convergence rates due to the data set characteristics. The characteristics of each data set can be viewed in Table 1. Once these data sets were chosen, each set was normalized to numerical representation. Note: the Wilt and Indian Liver Patient data sets were visually verified to confirm there was no missing data; the four instances of missing data in the MONK's Problems data set were removed.

There was a total of 72 tests performed. 40 tests were run on each of the four training algorithms applied to each of the 10 data sets. An additional 32 tests were run for each algorithm where each data set, except MONK's Problems and Wine, were trimmed to 300 instances where the ratio of classes present was preserved. These additional 32 tests on trimmed data were necessary since after 36 hours of testing, only a few generations had completed when using the original data sets. This would not have completed within an acceptable time frame for the purpose of this project. For these tests, the original data for MONK's Problems and Wine were used because they already had less than 300 instances.

For each test, the number of iterations, the relative, least squares and loss squared errors, as well as the percent misclassified of the final neural network were collected for performance and convergence analysis. The number of iterations was used to analyze convergence, and the error evaluations and percent misclassification were used to analyze the performance of each test.

4.1 Tuning Process

The MLP neural network used was designed to handle flexible parameters by taking in the inputs, the outputs, an error threshold, and the hidden layer topology. The network also allowed specification in terms of the number of hidden layers used and the activation function each hidden node used. For this project, the error threshold was set to 10 percent, the sigmoid activation function was used for each hidden and output node, and the hidden layer topology consisted of two hidden layers, the first layer containing three nodes, and the second containing two nodes. These MLP network parameters were consistent for each test. The following sections specify the parameters used for each training algorithm implemented.

Data Set	Num Instances	Num Attributes	Num Classes
Abalone (Waugh, 1995)	4,177	8	29
Car Evaluation (Bohanec, 1997)	1,728	6	4
Contraceptive Method Choice (Lim, 1997)	1,473	9	3
Indian Liver Patient (Ramana, 2012)	583	10	2
MONK's Problems (Thurn, 1992)	124	6	2
Page Blocks (Malerba, 1995)	5,473	10	5
Tic-Tac-Toe Endgame (Aha, 1996)	958	9	2
Wilt (Johnson, 2013)	4,889	6	2
Wine (Forina and Aeberhard, 1991)	178	12	3
Yeast (Nakai, 1996)	1,484	8	10

Table 1: The details of the ten data sets chosen from the UCI Repository to test on each algorithm (Bache and Lichman, 2015).

4.1.1 Backpropagation

There are three tunable parameters that were used in the implemented backpropagation algorithm: maximum number of generations, minimum learning rate, LR, and a momentum rate, MR. The maximum number of generations (iterations) was set to to ensure convergence. Additionally, to assist with the speed of convergence, a dynamic learning rate was applied that required a minimum learning rate of .3 to ensure learning always occurs. Finally, a momentum rate of .5 was used to avoid being stuck in a local optima.

4.1.2 Genetic Algorithm

The genetic algorithm has six tunable parameters: a maximum number of generations, a population size NP, the number of tournament participants, the number of tournament victors, a mutation rate, MR, and a crossover rate, CR. The maximum number of generations was set to 1000 to ensure convergence. The population size was set to 20 so the population was large enough to be diverse, but small enough to ensure several generations could perform searches in a reasonable amount of time. 10 individuals of the population were randomly selected for the tournament participants and the top five victors were chosen to produce the children of the next population. A 0.25 mutation rate and a 0.75 crossover rate was applied.

4.1.3 Differential Evolution

There are three tunable parameters in the implemented differential evolution algorithm: a maximum number of generations, a population size, NP, the beta mutation scalar, β, and the crossover rate, CR. Like the GA, the maximum number of generations was set to 1000 to ensure convergence and the population size was set to 20. The beta mutation scalar was set to 0.6 to create a trial vector with a larger step to ensure the search space was explored quickly, however, the 0.4 crossover rate ensured the population maintained its genetic diversity.

4.1.4 $(\mu + \lambda)$-Evolution Strategy

The tunable parameters in the ES implementation were the same as the GA for population size, mutation rate, crossover rate, and maximum number of generations with the addition of strategy parameters, Gaussian parameters, μ, and λ. The strategy parameters were initialized as standard deviations of the vector of real values and updated with a self-adaptive method using the log-normal equation discussed in section 3.5. μ parents was set to five and λ offspring was set to 20. The parameters within the random Gaussian were set as a range between 0 and 1 (Sheppard, 2015a).

5. Results

Although 72 tests were ran, of the original data sets, few were able to converge in the time allotted for testing. However, each test completed for the data sets that were trimmed to 300 instances and the exception of the MONK'S Problems and Wine data sets. Thus, statistical analysis was only performed on and considered for the trimmed tests and the MONK's Problems and Wine tests.

5.1 Performance

As stated in the experimental approach, relative, least squares and loss square errors, as well as the number of misclassified performance calculations were performed on trained MLP neural network for each data set. The performance of each network was defined as the accuracy of the classification. From this, we concluded:

1. Table 2 shows the relative error of each algorithm for each data set. Of these errors, DE was the most accurate for six of the ten data sets, while the GA was more accurate for three data sets, and the ES was only better for one data set. This error goes against our algorithm expectations of backpropogation producing the second most accurate solutions and the evolution strategy algorithm performing third best.

2. Least squares error results are shown in Table 3. Least squares error represents data fitness of the expected output versus the observed output given a model. The least squares information gathered for each data sets gives no clear indication of which algorithm performed better. However, this information does show that each algorithm generally was able to handle the same information in a similar fashion, such that their performance would be similar. Some algorithms performed better on specific datasets, but not with enough consistency to make any solid assumptions.

3. Table 4 displays the loss square error. It represents cost associated with each trained MLP neural network for each of the data sets. From the gathered data, BP generally has the loss, followed by DE. The GA was the next best performing algorithm, with ES almost always performing the worst of the four. These associated losses depict the hypothesized algorithm performance exactly.

4. The final performance measure calculated for classification of the MLP was the percent misclassified presented in Table 5. Since percent misclassified is a good measure for the performance of classification algorithms (Engelbrecht, 2007), we will focus on these results more so in the conclusion than previous error reporting.

Data Set	**BP**	**GA**	**ES**	**DE**
Abalone (Waugh, 1995)	36.24%	13.31%	17.17%	16.66%
Car Evaluation (Bohanec, 1997)	42.22%	11.16%	23.16%	18.79%
Contraceptive Method Choice (Lim, 1997)	55.68%	38.64%	32.49%	39.79%
Indian Liver Patient (Ramana, 2012)	32.1%	15.25%	11.86%	15.25%
MONK's Problems (Thurn, 1992)	32.59%	11.11%	20.87%	8.83%
Page Blocks (Malerba, 1995)	28.21%	8.14%	10.05%	6.62%
Tic-Tac-Toe Endgame (Aha, 1996)	33.65%	19.49%	19.49%	17.8%
Wilt (Johnson, 2013)	2.18%	11.72%	6.89%	7.67%
Wine (Forina and Aeberhard, 1991)	33.53%	8.94%	23.13%	17.57%
Yeast (Nakai, 1996)	58.81%	36.71%	41.55%	31.21%

Table 2: The relative error of each algorithm for each data set. Note: BP is backpropagation.

Data Set	**BP**	**GA**	**ES**	**DE**
Abalone (Waugh, 1995)	734.76	259.34	974.29	544.86
Car Evaluation (Bohanec, 1997)	28.11	26	68	44.53
Contraceptive Method Choice (Lim, 1997)	48.98	95.79	89	82
Indian Liver Patient (Ramana, 2012)	11.39	18	14	18
MONK's Problems (Thurn, 1992)	6.53	4.95	10.01	4.05
Page Blocks (Malerba, 1995)	30.69	44.62	35.4	36
Tic-Tac-Toe Endgame (Aha, 1996)	14.8	23	23	21
Wilt (Johnson, 2013)	0.99	3.92	2.05	2.15
Wine (Forina and Aeberhard, 1991)	14.65	8.05	16.2	10.93
Yeast (Nakai, 1996)	170.03	212.31	314.54	309.89

Table 3: The least squares error of each algorithm for each data set.

Data Set	**BP**	**GA**	**ES**	**DE**
Abalone (Waugh, 1995)	12.27	4.54	17.15	9.39
Car Evaluation (Bohanec, 1997)	0.59	0.44	1.15	0.76
Contraceptive Method Choice (Lim, 1997)	0.83	1.71	1.51	1.39
Indian Liver Patient (Ramana, 2012)	0.25	0.31	0.24	0.31
MONK's Problems (Thurn, 1992)	0.71	0.21	0.42	0.17
Page Blocks (Malerba, 1995)	0.58	0.76	0.59	0.63
Tic-Tac-Toe Endgame (Aha, 1996)	0.42	0.39	0.39	0.36
Wilt (Johnson, 2013)	0.02	0.07	0.03	0.03
Wine (Forina and Aeberhard, 1991)	0.43	0.23	0.46	0.31
Yeast (Nakai, 1996)	2.83	3.9	5.37	5.29

Table 4: The loss squared error of each algorithm for each data set.

Data Set	**BP**	**GA**	**ES**	**DE**
Abalone (Waugh, 1995)	89.83%	76.27%	77.97%	74.58%
Car Evaluation (Bohanec, 1997)	27.12%	20.34%	40.68%	33.90%
Contraceptive Method Choice (Lim, 1997)	83.05%	59.32%	54.24%	62.71%
Indian Liver Patient (Ramana, 2012)	25.41%	30.51%	23.73%	30.51%
MONK's Problems (Thurn, 1992)	70.83%	20.83%	41.67%	16.67%
Page Blocks (Malerba, 1995)	13.56%	11.86%	15.25%	10.17%
Tic-Tac-Toe Endgame (Aha, 1996)	42.37%	38.98%	38.98%	35.59%
Wilt (Johnson, 2013)	1.69%	6.78%	3.39%	3.39%
Wine (Forina and Aeberhard, 1991)	42.86%	22.86%	45.71%	31.43%
Yeast (Nakai, 1996)	83.05%	64.41%	69.49%	50.85%

Table 5: The percent misclassified by each algorithm for each data set.

5.2 Convergence

Additionally, a convergence analysis of the average number of iterations required for each algorithm to properly train the MLP neural network for each data set was calculated. Table 6 shows the average number of iterations used for a convergence analysis. The convergence rates were dependent on maximum number of generations. Knowing this, we concluded:

1. The set maximum number of generations was not large enough for the algorithms to find a solution within the set threshold error.

2. For the data sets that did find adequate solutions within the maximum number of generations, it seems the evolution strategy algorithm would likely converge towards a solution the quickest of the four algorithms.

Data Set	**BP**	**GA**	**ES**	**DE**
Abalone (Waugh, 1995)	10,000	1,000	1,000	1,000
Car Evaluation (Bohanec, 1997)	10,000	1,000	1,000	1,000
Contraceptive Method Choice (Lim, 1997)	10,000	1,000	1,000	1,000
Indian Liver Patient (Ramana, 2012)	10,000	1,000	1,000	1,000
MONK's Problems (Thurn, 1992)	10,000	1,000	783	1,000
Page Blocks (Malerba, 1995)	10,000	6	20	11
Tic-Tac-Toe Endgame (Aha, 1996)	10,000	1,000	1,000	1,000
Wilt (Johnson, 2013)	1	3	1	4
Wine (Forina and Aeberhard, 1991)	10,000	1,000	841	1,000
Yeast (Nakai, 1996)	10,000	1,000	1,000	1,000

Table 6: The average number of iterations taken to reach convergence.

6. Conclusion

It can be concluded that the No Free Lunch Theorem (Sheppard, 2015a) continues to hold when comparing algorithms against one another. No single algorithm was able to always outperform the others, although each algorithm did perform better than the other algorithms for different data sets. Solely considering the percent misclassified, backpropagation was only able to outperform any of the evolutionary approaches for the Wilt data set. The DE had a lower percent misclassified over all other algorithms for the Abalone, MONK's Problems, Page Blocks, Tic-Tac-Toe Endgame, and Yeast data sets. This does not necessarily provide enough evidence to state that DE performed better overall since other algorithms had better performances in other data sets. Of the GA and ES algorithms, the GA typically performed better than than ES.

If comparing results by the type of data, the Wilt and Page Blocks data sets both had very low misclassified percentages over all algorithms. This is due to the fact that both data sets were dominated by one class, even though there were two data classes. The data sets with a high percent misclassified for all algorithms were Abalone, Contraceptive Method Choice, and Yeast. This is due to these data sets having a more even distribution of the classes, where no single class dominated the set.

From the performance and convergence analysis performed, we conclude that no algorithm performed significantly better or was able to converge within the specified number of generations. The set threshold of 10 percent contributed to the poor convergence rates and often led the algorithm to reach it's respective maximum number of generations. The threshold of 10 percent was set to ensure better classification when evaluated, however we had noticed in tuning our parameters, the MLP would typically get stuck at percent error greater than 10 and was unable to find weights for the MLP that would result in better classification. This is because the maximum number of generations had not been significantly increased.

7. Future Work

There are several things we could do to improve our MLP training results. This includes using a smaller error threshold, altering the tunable parameters, implementing non-static mutation and crossover rates and adjusting the selection methods used in the GA, ES, and DE. Longer testing times could be allowed for future runs.

References

D.W. Aha. UCI machine learning repository: Tic-tac-toe endgame data set, 1996. URL `http://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame`.

Musrrat Ali, Millie Pant, and Ajith Abraham. Simplex differential evolution. *Acta Polytechnica Hungarica*, (5):97–99, 2009.

Vasan Arunachalam. Water resources research report. Technical report, July 2008.

K. Bache and M. Lichman. UCI machine learning repository, 2015. URL `http://archive.ics.uci.edu/ml`.

M. Bohanec. UCI machine learning repository: Car evaluation data set, 1997. URL `http://archive.ics.uci.edu/ml/datasets/Car+Evaluation`.

Andries P. Engelbrecht. *Computational Intelligence: An Introduction.* John Wiley and Sons, Ltd, 2007.

Kelly Fleetwood. An introduction to differential evolution. Powerpoint, 2004. URL `http://www.maths.uq.edu.au/MASCOS/Multi-Agent04/Fleetwood.pdf`.

M. Forina and S. Aeberhard. UCI machine learning repository: Wine data set, 1991. URL `http://archive.ics.uci.edu/ml/datasets/Wine`.

Alex S Fraser. Simulation of genetic systems by automatic digital computers vi. epistasis. *Australian Journal of Biological Sciences*, 13(2):150–162, 1960.

B. Johnson. UCI machine learning repository: Wilt data set, 2013. URL `https://archive.ics.uci.edu/ml/datasets/Wilt`.

Maxwell Stinchcombe Kurt Hornik and Halber White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

T. Lim. UCI machine learning repository: Contraceptive method choice data set, 1997. URL `http://archive.ics.uci.edu/ml/datasets/Contraceptive+Method+Choice`.

D. Malerba. UCI machine learning repository: Page blocks classification data set, 1995. URL `https://archive.ics.uci.edu/ml/datasets/Page+Blocks+Classification`.

T.M. Mitchell. *Machine Learning.* McGraw-Hill Science, Engineering, and Math, 1997.

David E Moriarty, Alan C Schultz, and John J Grefenstette. Evolutionary algorithms for reinforcement learning. *J. Artif. Intell. Res. (JAIR)*, 11:241–276, 1999.

K. Nakai. UCI machine learning repository: Yeast data set, 1996. URL `http://archive.ics.uci.edu/ml/datasets/Yeast`.

Surendra Parsad Badu M. Venkateswarlu N. Ramana, B. UCI machine learning repository: Ilpd (indian liver patient dataset) data set, 2012. URL `https://archive.ics.uci.edu/ml/datasets/ILPD+(Indian+Liver+Patient+Dataset)`.

John Sheppard. Lecture notes in machine learning: Soft computing, 2015a.

John Sheppard. Csci 447 machine learning: Soft computing project #3 pdf, 2015b.

Rainer Stron. Differential evolution for continuous function optimization. URL `http://www1.icsi.berkeley.edu/ storn/code.html`.

S. Thurn. UCI machine learning repository: Monk's problems data set, 1992. URL `http://archive.ics.uci.edu/ml/datasets/MONK%27s+Problems`.

S. Waugh. UCI machine learning repository: Abalone data set, 1995. URL `http://http://archive.ics.uci.edu/ml/datasets/Abalone`.