# Comparison of Genetic Algorithms and Backpropogation as Methods for Training a MLP Network

**Trent Baker** b.trent5@gmail.com
*Department of Computer Science*
*Montana State University*
*Bozeman, MT 59717, USA*

**Logan Bonney** wakeup2early@gmail.com
*Department of Computer Science*
*Montana State University*
*Bozeman, MT 59717, USA*

**Bradley White** white.brad17@gmail.com
*Department of Computer Science*
*Montana State University*
*Bozeman, MT 59717, USA*

**Abstract**

In this paper the effectiveness of four methods for training a multilayer perceptron feed-forward neural network are examined. The algorithms implemented and compared are backpropagation, a genetic algorithm with real-valued chromosomes, an $(\mu + \lambda)$ evolution strategy, and differential evolution. Their performance was evaluated based on the mean squared error from five regression datasets. Each algorithm was tested on a variety of neural network structures and tested several times to tune algorithm specific parameters before the final evaluations. ES was found to be the most effective for most tests, however there were cases where other algorithms did approximate the function to a smaller margin of error, thus the no free lunch theorem still applies.

## 1. Introduction

### 1.1 Description of the Problem

The purpose of this experiment is to compare three evolutionary algorithms and backpropagation as methods for training a multilayer perceptron (MLP) feedforward neural network. The three evolutionary algorithms that are compared are: a genetic algorithm with real-valued chromosomes, $(\mu+\lambda)$-evolution strategy, and differential evolution. These algorithms are compared based on convergence rate and the accuracy of approximating a function from five chosen datasets from the UCI Machine Learning Repository (Lichman, 2013).

### 1.2 Hypotheses

The evolution strategy (ES) and differential evolution (DE) will perform better than the genetic algorithm (GA) because they are different attempts to improve the GA. Furthermore, GA is essentially searching the problem space randomly, where ES and DE take a more directed approach towards finding the correct solution. DE should be a good match for some problems, while struggling to excel at others. Since DE relies on differentials, it does not explore beyond the scope of its starting population. ES on the other hand should be a good fit for problems with a dataset that has relatively few local minima. Due to the controlled mutation of GA, it may be difficult to leave local minima. Therefore, GA will be the most consistent and the most middling of the three new methods. As for backpropagation (BP), it should be surpassed by all of the evolutionary methods. The evolutionary algorithms, besides DE, explore the problem space more than BP can, but DE is expected to converge much faster than the others, making up for its less diverse method of recombination.

## 2. Algorithms

This section provides a high level overview of the algorithms being implemented and compared.

### 2.1 Backpropagation

Backpropogation works by calculating the gradient with respect to a node, based on the error of the network for a training example and the input and output of the node. This allows backpropogation to reduce error quickly relative to evolutionary algorithms that are based on semi-random changes. Unfortunately, since it simply follows the gradient it is more prone to getting trapped in local minima (Zainuddin and Pauline, 2008). Backprogation was used to smooth out the error and provide a more consistent hike towards the gradient. This does increase the chances of getting caught in local minima but that can be combated by adding momentum to the weight update rule (Zainuddin and Pauline, 2008).

### 2.2 Genetic Algorithm

The genetic algorithm is a simple process composed of three steps, crossover, mutation and selection. Some form of elitism is used to preferentially select parents that have low error values. This ensures that the population gradually gets better. After a group of parents are selected, they are combined with each other using multi-point crossover to create a new population of offspring. Each new individuals has a chance to mutate which may change some weights positively or negatively slightly. This complete process is a generation, and repeats until a certain threshold or the population has converged (Engelbrecht, 2007).

Parents are chosen from the best individuals in the existing population, bacause of this it is likely that the population will be better overall after adding the new offspring. This elitism, applied every generation is what allows the GA to learn. There are two tunable parameters, crossover rate and mutation rate, both of which can drastically affect convergence rate and error. Increasing the crossover rate will lead to the to children being made up of smaller slices of more parents. This could be a positive or negative outcome depending on the natural shape and dependence of the data. The mutation rate affects how often values are modified, though not the magnitude of the modification. Increasing the mutation rate can lead to a a network that has a rougher learning curve because of the constant introduction of new genetic patterns.

### 2.3 Evolution Strategies

Evolution strategies (ES) is similar to the GA, but executes the primary steps slightly differently and in a different order. Specifically, a $(\mu + \lambda)$-evolution strategy is used; where $mu$ signifies the population size, $\lambda$ is the number of children, and the plus notation implies that selection is performed from the combination of population and offspring. In $(\mu + \lambda)$-ES, crossover involves the entire population and produces at least as many children as there are parents, specifically $1 \leq \mu \leq \lambda < \infty$. Additionally, each chromosome has a strategy parameter vector appended, where each attribute for an individual has a specific strategy parameter associated with it, denoted $\sigma$. Crossover is performed on the strategy parameters during reproduction as well. The strategy parameters are initialized randomly and then updated as follows:

$$\sigma'_{ij} = \sigma_{ij} e^{\tau' N(0,1) + \tau N(0,1)}$$

where $\tau' = \frac{1}{\sqrt{2n_x}}$, $\tau = \frac{1}{\sqrt{2\sqrt{n_x}}}$, and $n_x$ is the size of the population (Engelbrecht, 2007). Each child will be mutated after creation, thus ES has no tunable mutation rate, by updating the strategy parameters with the preceding equations, then applying each one to their respective weight. Specifically, $\Delta x_{ij} = \sigma_{ij} N(0,1)$ is used to apply the strategy parameters. Finally, elitist selection is performed from the population and all children to determine the best population for the next generation; with the population size remaining constant throughout the algorithm. To summarize the ES algorithm creates $\lambda$ children via crossover from the entire population, updates the strategy parameters for each child and applies them, then the next population is chosen from the best individuals $(\mu+\lambda)$ and the process repeats.

ES can be tuned both by the crossover rate in the same way as GA, and by the number of children generated. Since the next population is chosen from the sum of the parents and the children, decreasing the number of children generated could lead to decreased genetic diversity and therefore premature convergence. Alternatively, a number of children that is too high may make it hard to converge at all with no genetic consistency from generation to generation.

### 2.4 Differential Evolution

Differential evolution performs operations in a different order than the other algorithms, it first does mutation, then crossover on the mutated vectors. For mutation, DE uses the rule $trial_i = ind_i + \beta(ind_{I2} - ind_{i3})$ for each individual, i, where $i \neq i2 \neq i3$ (Engelbrecht, 2007). The trial vector for each individual is saved in an array called trial vectors. The trial vectors are then crossed over with the parent by choosing random features from the trial vector to replace the features in the parent, the results of this is a child vector. For replacement, a parent and its child are each evaluated and whichever performs better is kept in the population. DE needs to start with individuals that are spread out in the search space to get the best results because mutation is based off of differences between individuals in the population.

## 3. Experiment Approach

### 3.1 Method

Each algorithm was run multiple times tweaking the tunable parameters each time. This determined the effects of each in turn through examining the graphs produced of mean squared error for each generation. The properties of interest were convergence rate, accuracy, and computation time. Furthermore, the structure of the network, specifically the number of hidden layers and hidden nodes, was considered a tunable parameter and their effects were observed on the algorithms. This process allowed for the optimization of the tunable parameters discussed next. Once the parameters where chosen each algorithm was run on each dataset, using the same neural network structure for each. This was repeated multiple times with different network structures to observe any differences.

### 3.2 Tuning

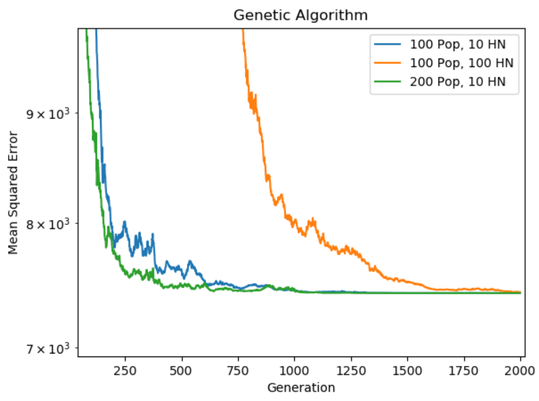
Throughout the tuning process, each evolutionary algorithm was ran at a large number (100,000) of generations to determine if there were very slow rates of convergence on any of them and after hours of running it was determined there was very little change after the 10000th generation, thus this parameter was used for all tests during experimentation. Next, various sizes of population and the structure of the neural network were compared using GA, shown in Figure 1a. The tuning result shows that an increase in hidden nodes smooths the convergence rate, avoiding local minima, however the computation time significantly increased so it was not practical. Also, increasing the population size allowed for a slight increase in accuracy however the computation time increased so, for all experiment tests a population size of 100 was used because it provided a balance between computation time, convergence rate, and accuracy. The structure of the network was still varied since accuracy of the network increases as a function of the amount of hidden nodes and layers. Next, algorithm specific parameters were examined using a small dataset.

#### 3.2.1 BP

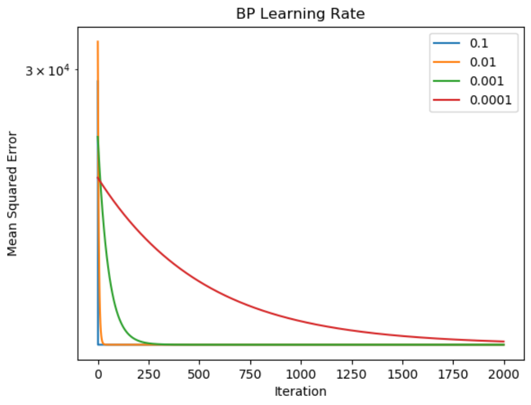
For BP, the tunable parameters are the number of iterations and the learning rate. An iteration consists of doing forward propagation, calculating error and then doing backpropagation to update the weights. The iterations were tested as high as 1 million, but few changes were observed even past the 50,000th iteration. Since the iterations are quite fast, 100,000 iterations was chosen for all tests. Learning rate was tested at four levels, shown in Figure 1b, which determined that setting the rate size too small resulted in a very large step size and possibly premature convergence, and setting it too high resulted in a loss of accuracy. Therefore, a learning rate of 0.001 was used for all tests because it appeared to be the middle group between both extremes.

#### 3.2.2 GA

The GA has two specific tunable parameters, crossover and mutation rate. Crossover rates were tested at three levels, shown in Figure 2a, and it was determined that there was little effect on the accuracy and convergence rate, thus the crossover rate was set to 0.5 to help reduce computations. Mutation rate however showed significant differences in the



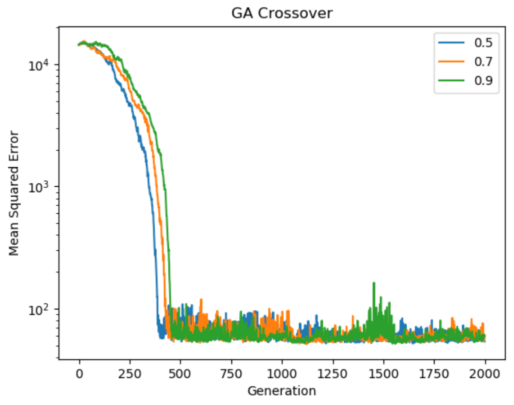
(a) Affects of population size and number of hidden nodes on GA



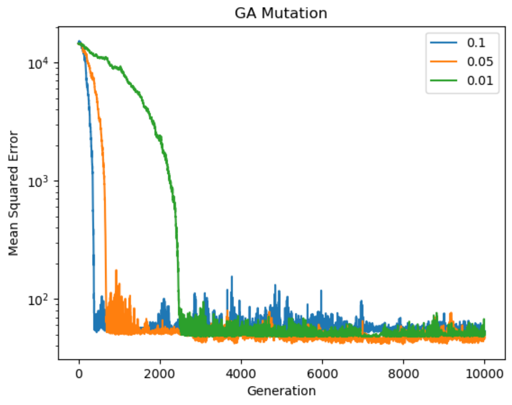
(b) Affects of learning rate on BP

Figure 1

convergence rate, observed in Figure 2b. A rate of 0.01 appeared to slow the rate down by a large margin for no benefit on accuracy, while a rate of 0.05 had more accuracy for more generations so this rate was chosen for experimentation.



(a) Affects of crossover rate on GA

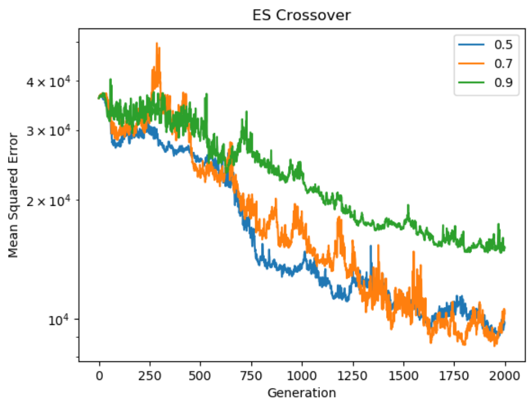


(b) Affects of mutation rate on GA

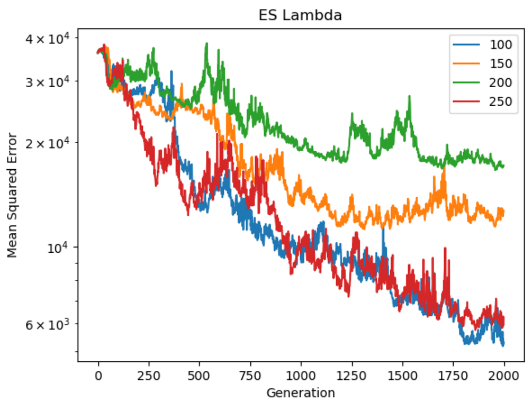
Figure 2

### 3.2.3 ES

ES has two tunable parameters: crossover rate and $\lambda$, the number of children produced in each generation. The crossover rate, in Figure 3a, was observed to shown faster convergence and better accuracy set to 0.5, which can be concluded to happen because the better attributes from a chosen parent are more likely to persist throughout generations. Next, the effect of $\lambda$ is shown in Figure 3b, with interesting results. It is worth noting that the population size was kept constant for these tests at 100. The smallest and largest $\lambda$ converged at approximately the same rate, and to the same accuracy while intermittent values were slower to converge and at less accuracy. It is unknown why it exhibited this behavior, and it was determined that since a large $\lambda$ produced an exponential growth in computation time that setting $\lambda = 100$ was appropriate.



(a) Affects of crossover rate on ES



(b) Affects of number of children on ES

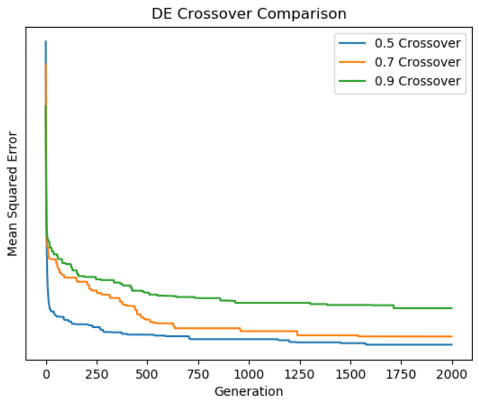
Figure 3

### 3.2.4 DE

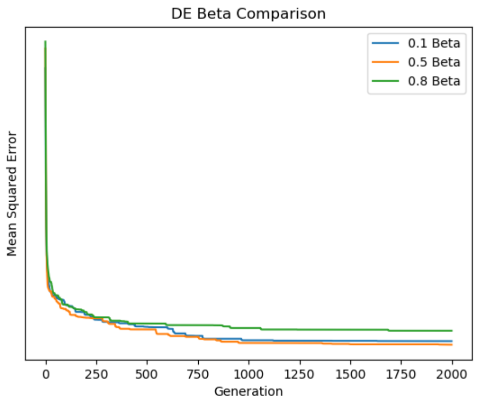
Like the other evolutionary algorithms, DE has two tunable parameters, one of which is also crossover rate and the other is $\beta$, a modifier for mutation. Crossover rate was shown to have relatively the same performance in DE as the other algorithms, shown in Figure 4a. Once again, a crossover rate of 0.5 was chosen for convergence and also consistency. Having all crossover rates set to 0.5 avoids any confounding variables within the analysis of the algorithms. Lastly, $\beta$ was examined in Figure 4b. Interestingly, a large $\beta$ did not result in higher accuracy, which was hypothesized earlier even though the population would have more diversity. For the testing, $\beta = 0.5$ was chosen because it showed the best performance, which aligns with (Engelbrecht, 2007).

## 4. Results

Four main tests were conducted, with four different neural network structures to determine how well each algorithm performed on each dataset, for a total of 80 tests. Three tests were conducted with one hidden layer, and 10, 20, or 100 hidden nodes, which are shown in Figures 5, 6, and 7 respectively. The last test was conducted with two hidden layers and 10 hidden nodes, Figure 8. Overall, it can be observed that ES almost always performed better, or equal to GA and BP. However there were a few cases when ES did diverge due to the randomness of individuals when they were initialized. GA performed average for most tests which was expected due to the random searching. DE only performed well for the network structure with 100 hidden nodes, meaning that the diverse search space significantly helped



(a) Affects of crossover rate on DE



(b) Affects of beta on DE

Figure 4

the algorithm. BP and DE had problems converging quickly, which could have been from the small amount of datapoints which were in most datasets, the most was 1503 in the airfoil dataset. Lastly, BP performed about the same as the GA, average, except for test with using the airfoil dataset. This can be attributed to the large amount of noise which is present in that dataset.

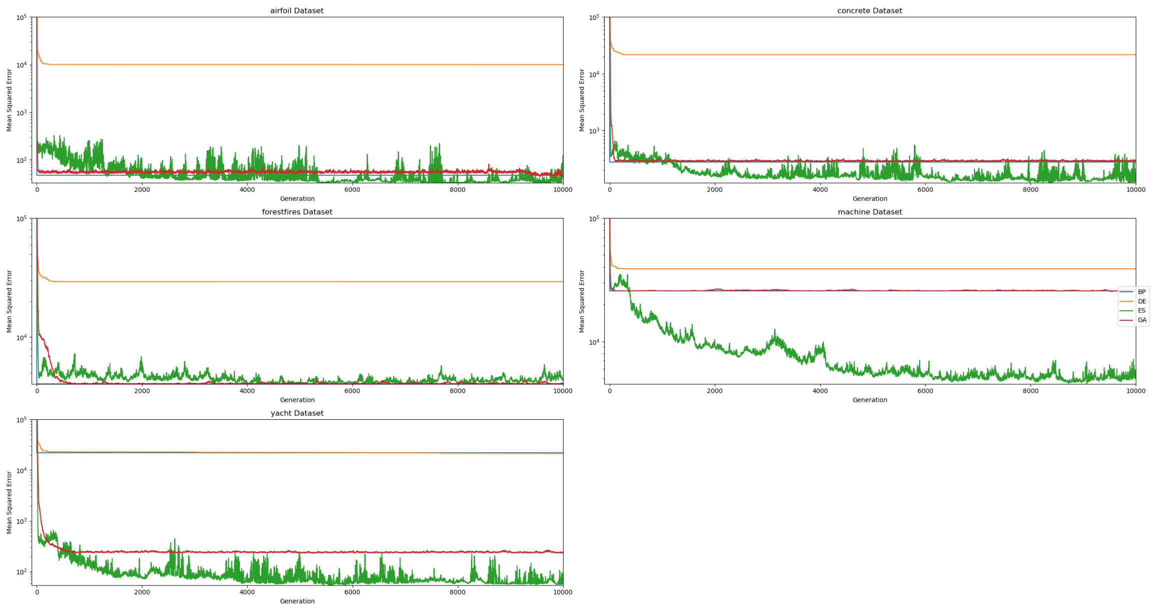


Figure 5: Mean Squared Error vs Generations for each algorithm on each dataset with one hidden layers and ten hidden nodes

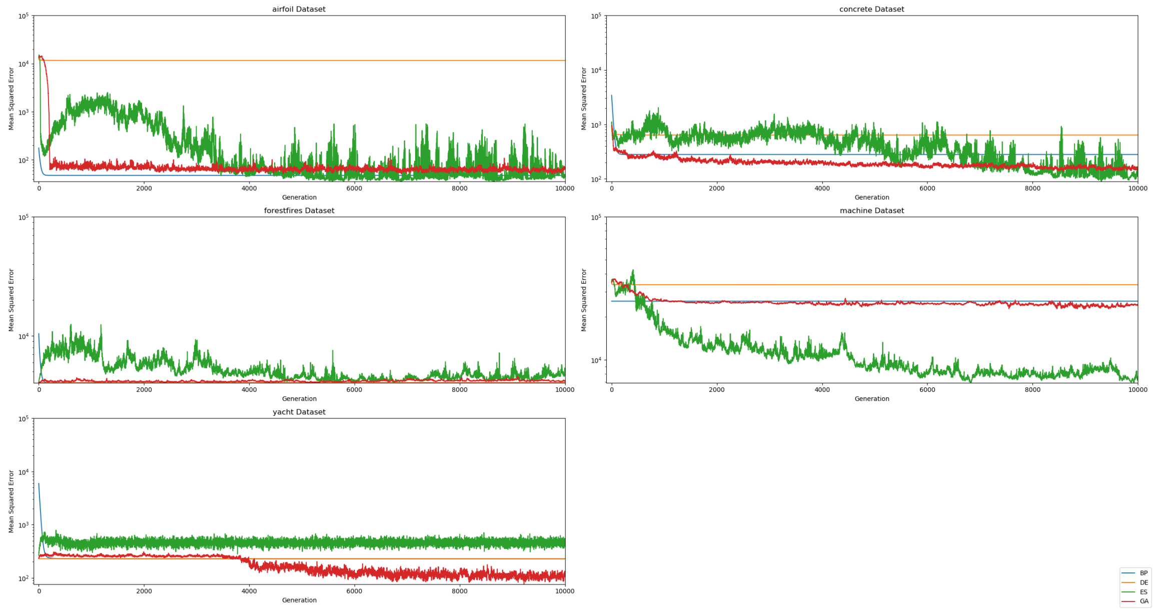


Figure 6: Mean Squared Error vs Generations for each algorithm on each dataset with one hidden layers and 20 hidden nodes

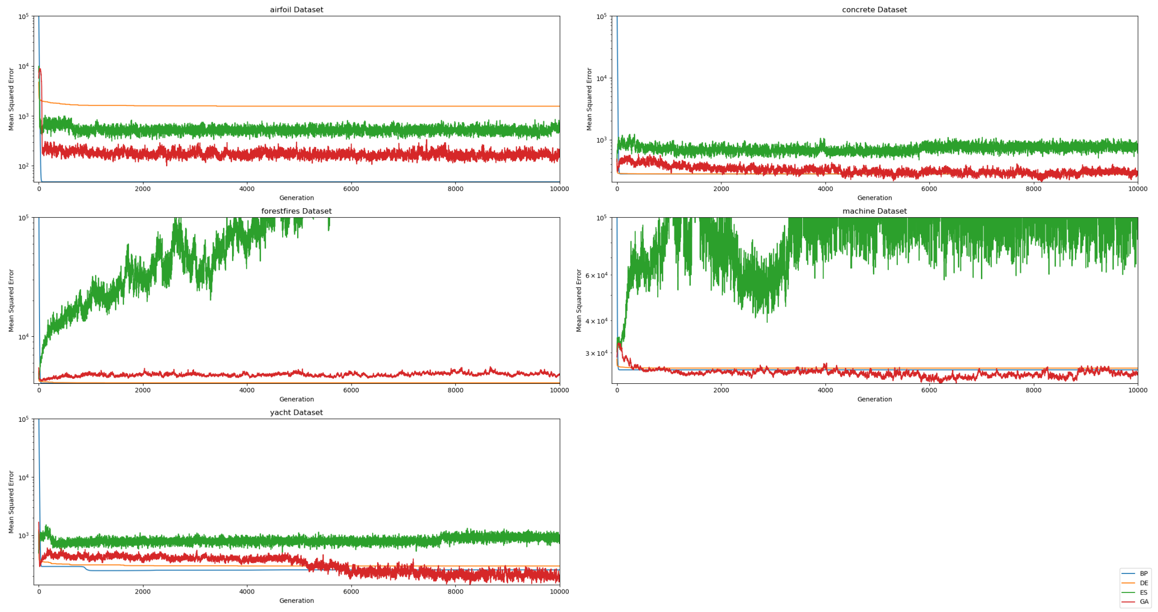


Figure 7: Mean Squared Error vs Generations for each algorithm on each dataset with one hidden layers and 100 hidden nodes

## 5. Conclusion and Future Work

In conclusion, for the datasets test, ES was the "best" algorithm, though further testing could be performed. It is possible that the chosen datasets were too similar and thus ES was observed to outperform the others. As usual, the no free lunch theorem applies since a dataset with significant noise did not show ES performing well. Future work could consist of $k$-fold cross validation performed to determine if the results were more conclusive and the testing could be expanded to large datasets or even classification problems. Additionally, some of the parameters which had shown much slower convergence rates could be run for extended periods of time to observe if they eventually become more accurate. Lastly, some methods presented to dynamically set parameters such as crossover and mutation rates, could be employed to observe the effects.

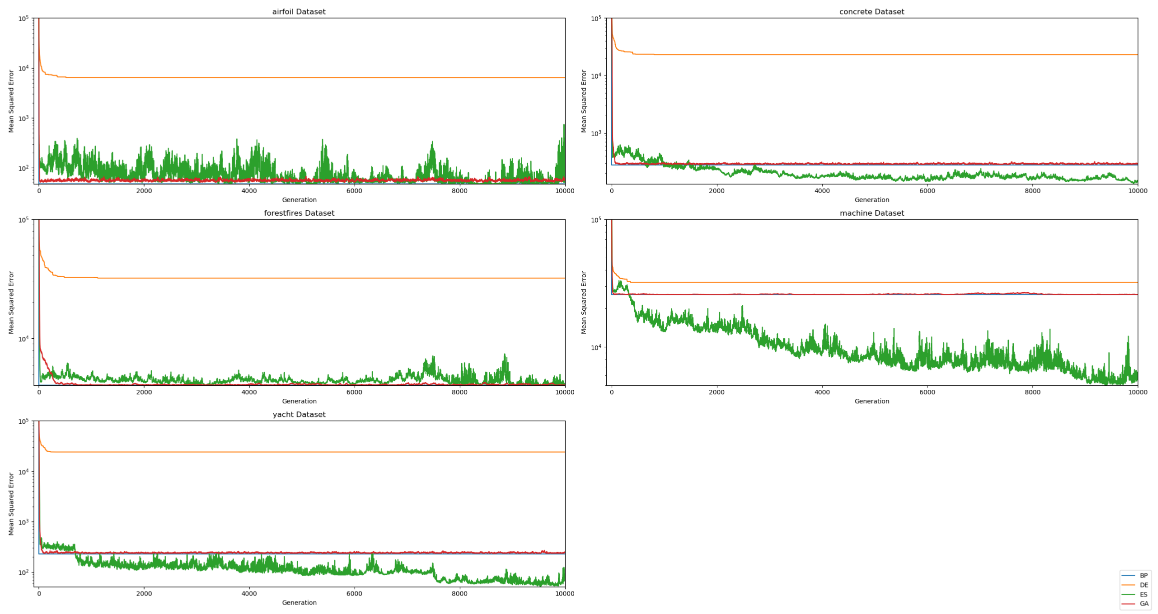


Figure 8: Mean Squared Error vs Generations for each algorithm on each dataset with two hidden layers and ten hidden nodes

## References

Andries P Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

Zarita Zainuddin and Ong Pauline. Function approximation using artificial neural networks. *WSEAS Transactions on Mathematics*, 7(6):333–338, 2008.