Neural Network Training Via Evolutionary Methods

Stuart Dilts

Jacob Senecal

Neil Walton

Abstract

We examine the differences in the accuracy and convergence of different methods used for training neural networks on both regression and classification problems. Specifically, we investigate genetic algorithms, differential evolution, $\mu+\lambda$ evolutionary strategies (ES), and backpropagation. In all experiments, the network structure is held constant and only the weights of the network are learned. After parameter tuning the accuracy and convergence rate was tested on the various training methods and datasets. While no one algorithm was most accurate over all datasets, backpropagation performed as well or better than the evolutionary methods on regression tasks, but was less consistent on classification problems. Overall, $\mu + \lambda$ ES converged in the fewest generations, but all three evolutionary methods tended to converge in fewer generations (iterations) than backpropagation.

Keywords: Genetic algorithm, evolutionary strategy, differential evolution, neural network

1. Introduction

The key to creating a multi-layer perceptron (MLP) neural network that is capable of classification or regression is to adjust the weights of the connections in the network in response to the accuracy (or lack thereof) of the network in classifying or predicting the values of training data. A common way to train network weights is through backpropagation, where the weights are updated using a gradient descent based method which determines the weight updates based directly on the error measure at the output of the network. Backpropagation is a consistent method in terms of its operation, and it generally produces good results. However, as backpropagation uses gradient descent it may become stuck in a local optimum, leaving the possibility that a more optimal solution may exist.

Evolutionary algorithms can be used as an alternative to training MLP network weights. Evolutionary algorithms incorporate a degree of randomness in their search through the solution space which reduces the risk of becoming stuck in a local optimum compared to using backpropagation as the MLP training method. As a result, it is possible that an evolutionary approach to training an MLP network's weights may result in a better solution. In some cases evolutionary algorithms are also used to explore different network topologies, but in this study the network topology is fixed and only network weights are updated.

To investigate the differences in accuracy and convergence rate achieved through evolutionary training methods and backpropagation, a genetic algorithm, a differential evolution strategy, and a $\mu + \lambda$ evolution strategy are implemented. These algorithms are then tested on their ability to train a MLP network to perform classification and regression tasks on datasets obtained from the University of California Irvine machine learning repository.

2. Description of Algorithms

The three evolutionary approaches used in this study all have commonalties with one another, and share the same general strategy in training an MLP network's weights. The strategy is to start with a population of MLP networks with randomly initialized weights, then the algorithms update these weights through some combination of a mutation operator, a crossover operator, and a selection operator. An "individual" is a set of weights that represent an MLP network. All of tested evolutionary algorithms represent a network as a single vector of weights. Weights that originate from any given node are located consecutively in the vector to maintain the association between a set of weights and their node.

All the evolutionary algorithms make use of the same fitness function, which is the mean squared error for regression problems, and percent incorrectly classified instances for classification problems. However, they all have key differences in which operators are applied, the order in which mutation, crossover, and selection take place, and the details of implementation of each operator, that makes the three evolutionary algorithms distinct from one another.

2.1 Genetic Algorithm

The genetic algorithm makes use of mutation, crossover, and selection to evolve a population towards the goal state, which in this case is a set of network weights that will perform accurate classification and regression.

The `evolve` method first selects parents from the current population through a tournament selection process. A tournament selection works by randomly selecting between 2 and k individuals, where $k = |population|$ to compete in the tournament. Out of the selected individuals, the individual with the highest fitness is chosen to produce a descendant in the next generation. The number of individuals that compete in each tournament is an important parameter. If $|population|$ individuals were used in each tournament then only the fittest individual would move on to produce children, while if a small number of individuals compete in a tournament this increases the likelihood of less fit individuals moving on. This process continues until $|population|$ individuals have been selected to receive descendants in the next generation. In this study out of a population size of 50, 15 individuals are selected to compete in the tournaments. Using 15 randomly selected individuals out of a total population size of 50 provides a chance that some highly fit individuals will be selected to produce children in the next generation, but it also leaves the chance that some less fit individuals will also move on, which helps to maintain diversity in the population.

After parent selection, `crossover` and `mutation` occur. If a randomly generated number between $[0, 1)$ is below the user specified crossover probability, typically 0.7 to 0.9, then the current parents are used to produce two children through one point crossover. These children are then returned to the `evolve` method where they are passed to the `mutate` method.

Mutation proceeds by moving through each gene (weight) in an individual, mutating the gene if a randomly generated number in the range $[0, 1)$ falls below the mutation probability, typically 0.1 to 0.3. The magnitude of the mutation is determined by sampling randomly from a normal distribution, $N(0,1)$.

After mutation "fittest" replacement occurs, meaning the old population is replaced by the fittest individuals from the pooled old and new generation. This process proceeds for a user defined maximum number of generations, or until convergence is reached.

2.2 $\mu + \lambda$ Evolution Strategy

Evolutionary strategies focus on numerical optimization problems, and therefore work with individuals that are represented as arrays of real-valued numbers (Kruse et al., 2016). A distinctive feature of evolutionary strategies, is that they not only try to evolve the population to a more optimal state, they also try to improve the evolution operators, (e.g. mutation). Evolutionary strategies are also distinct in that they typically rely on mutation, often eschewing crossover entirely.

The implementation of the evolutionary strategy in this study relies exclusively on mutation, and it uses a local self-adaptive Gaussian mutation strategy, which associates a parameter σ, with each gene within an individual. The parameter σ is used as the standard deviation when a mutation value is sampled randomly from a normal distribution.

The idea behind self-adaptive Gaussian mutation is that individuals with "bad" mutations sizes will tend to produce "bad" offspring and will be filtered out of the population. While "good" mutation step sizes will spread within the population. A large step size can also be though of as favoring exploration of the search space, while a small step size would correspond to local optimization or exploitation. While the genes within an individual are updated with the mutation step size, the mutation step size itself is updated through a random sampling from a Gaussian distribution,

$$u \leftarrow N(0,1)$$

$$\sigma_x \leftarrow \sigma_x * \exp(u/\sqrt{length(chromosome)}$$

The other key feature of the $\mu+\lambda$ evolution strategy is that only the best individuals are used to create the next generation. In the evolutionary strategy, μ represents the number of individuals in the parent generation, and λ represents the number of children that were created in the mutation process. The $\mu + \lambda$ notation denotes that the parents and children are pooled and the best individuals are selected from this pool to form the subsequent generation.

2.3 Differential Evolution

The tools used in differential evolution are all ones which we have already encountered thus far — selection, mutation, crossover, and replacement (in that order) — however, the implementation of each varies from those of both the genetic algorithm and the evolutionary strategy. Selection has no real frills or innovations, the crossover operator bears a strong resemblance to the uniform crossover often employed by genetic algorithms, replacement is a simple one-versus-one comparison, but it is in the mutation operator where the largest differences and points of interest lie.

In a single round of evolution, we begin with the selection of the parents that will generate the offspring of the next generation. Two common methods for selecting parents in differential evolution are picking the k fittest parents, or selecting the parents at random from the population. The two methods tend not to lead to significant differences in the accuracy of the algorithm, so for simplicity's sake, we choose the parents at random.

Next, and most interestingly, we have the mutation operator. In differential evolution, mutation does not occur through the sampling of a normal distribution and changing individual genes by the sampled amount. In differential evolution, a mutated trial vector is created by combining three randomly selected members of the population, according to the equation: $u = x_1 + \beta(x_2 - x_3)$, where u is the trial vector, the x_is are the randomly selected individuals, and β is the scaling factor, where $0 \leq \beta \leq 2$. Increasing β increases the variation caused by the mutation by scaling up the difference between x_2 and x_3, and setting $\beta = 0$ would result in no mutation of the individual x_1. A common rule of thumb is to set $\beta = 0.5$, which we follow for the baseline parameters of our differential evolution experiments.

After the trial vector has been created, it is crossed-over with the parent selected during the selection phase. The two most commonly used methods for crossover in differential evolution are exponential crossover and binomial crossover. While exponential crossover is the method selected in the original formulation of the differential evolution algorithm, binomial crossover is found more commonly in subsequent applications and is the method we employ (Zaharie, 2007).

As with all crossover variations, binomial crossover selects some genes from the parent and some genes from the trial vector to create an offspring. The rate at which genes are selected from each source is dictated by a crossover probability, p_r. To begin, we select j^*, where j^* is an index between $[1,\ n_x]$, where n_x is the number of genes in a chromosome. Then, for each index, $j \in [1, n_x]$, generate $p = U(0,1)$. If $p < p_r$ or $j = j^*$, select the jth gene from the trial vector, otherwise choose the jth gene from the parent. By selecting the index, j^*, we ensure that at least one gene is crossed-over during this step. Increasing p_r tends to lead to faster convergence, but setting it too high runs the risk of premature convergence. A common range for p_r in the literature is $0.3 \leq p_r \leq 0.9$, where setting $p_r = 0.5$ results in uniform crossover (Gamperle et al., 2002). We select $p_r = 0.35$ to fall within this recommended range, but at the lower end of the range to mitigate the risk of premature convergence.

Once the parent has been randomly selected and the offspring has been generated through mutation and binomial crossover, a simple comparison of fitness is made between the parent and the offspring, with the fitter of the two returning to the population, and the less fit being culled.

3. Hypotheses

The hypotheses in this study focus on the effect of varying parameters within each algorithm, and the overall accuracy and convergence rate of the algorithms.

1. H_0: Within the differential evolution algorithm, adjusting β, a parameter used in the process of creating new individuals, will have no effect on accuracy.

H_a: Adjusting the parameter β will have an effect on accuracy of the differential evolution algorithm.

2. H_0: Adjusting the initial range of σ, the parameter or parameter vector used for mutation in the genetic algorithm, and the $(\mu + \lambda)$ evolution strategy, will have no effect on accuracy of the algorithms.
   H_a: Adjusting the parameter σ will have an effect on accuracy of the genetic algorithm, and the $(\mu + \lambda)$ evolution strategy.

3. H_0: Adjusting the population size used in the genetic algorithm, $(\mu + \lambda)$ evolution strategy, and the differential evolution algorithm, will have no affect on accuracy of the algorithms.
   H_a: Adjusting the population size used in the genetic algorithm, $(\mu + \lambda)$ evolution strategy, and the differential evolution algorithm, will have an affect on accuracy of the algorithms.

4. H_0: There is no difference in the accuracy achieved in classification and function approximation tasks by the genetic algorithm, $(\mu + \lambda)$ evolution strategy, differential evolution algorithm, and backpropagation.
   H_a: There will be a difference in the accuracy achieved in classification and function approximation tasks by the genetic algorithm, $(\mu + \lambda)$ evolution strategy, differential evolution algorithm, and backpropagation.

5. H_0: There is no difference in the convergence rate observed in classification and function approximation tasks by the genetic algorithm, $(\mu + \lambda)$ evolution strategy, differential evolution algorithm, and backpropagation.
   H_a: There will be a difference in the convergence rate observed in classification and function approximation tasks by the genetic algorithm, $(\mu + \lambda)$ evolution strategy, differential evolution algorithm, and backpropagation.

4. Experimental Design

Our main experimental goals are to determine the relative accuracy and convergence rate of multi-layer perceptron networks whose weights are trained using a genetic algorithm, a $(\mu + \lambda)$ evolution strategy, differential evolution, and backpropagation. Accordingly, our experimental design focuses on determining the appropriate parameters for each of the algorithms to get the best performance out of each, thereby allowing a fair comparison between the algorithms. The experimental design is also focused on tracking appropriate error measures throughout the experimental runs, so we can make comparisons between convergence rate and accuracy for each of the algorithms.

The multi-layer perceptron (MLP) network structure (i.e. number of layers, nodes per layer) will be held constant for all algorithms and all tests. The structure will be based on the results of the previous project where the effect of varying number of hidden layers and nodes per layer was investigated. The results of that study pointed to using 30 nodes per layer, and two hidden layers, as a network structure that provides consistently accurate results. The parameters used in the backpropagation experiments (i.e. learning rate), will also be based on the results of the previous project.

Both the parameter tuning tests, and the final tests to assess relative accuracy and convergence of the different algorithms, will be conducted on five different datasets from the University of California Irvine (UCI) machine learning repository. These datasets are; Abalone, Wine, Letter-Recognition, Yacht Hydrodynamics, and Concrete Slump. We have selected datasets that have a varied number of features and instances. These datasets also provide both regression and classification problems. The variation in the datasets was chosen deliberately to avoid a situation where one algorithm outperforms the others due to the structure of a single dataset being advantageous to that particular algorithm.

Each algorithm requires parameter tuning that will be conducted before the final experimental tests that assess the accuracy and convergence of all algorithms. The mutation rate and crossover rate in the genetic algorithm, and the number of parents and offspring in the $(\mu + \lambda)$ evolution strategy will be varied over a typical range of values. For the mutation rate this range of values will be 0.01 to 0.3, and the crossover rate will be varied over 0.6 to 1. For the evolutionary algorithm the ratio of parents to offspring will be $\mu : \lambda = 1 : 7$. The value ranges for these parameters have been selected according to the guidelines set by (Kruse et al., 2016). For the differential evolution algorithm, the DE/rand/1/bin variant will be used across all experiments, and β, the mutation scaling factor, will be varied over the range 0.1 to 0.9. The total population size for all algorithms will then be varied to determine an acceptable range of population sizes to use in the final tests. The initial population sizes we investigate will be based on research by (Rylander and Gotshall, 2002). However, we expect ideal population sizes will vary based on the properties of a given dataset.

Once parameter tuning is completed, the experiments to test the hypotheses will be conducted. Each of the experiments will be performed using ten fold cross-validation on each of the five UCI datasets. This will provide sufficient results to form a statistically significant conclusion for each of the hypotheses. Also, each hypothesis concerns how accuracy and convergence rate is affected when certain algorithm parameters are varied, or the hypothesis requires that the final convergence rate and accuracy of each algorithm be determined. To assess convergence rate and accuracy, the root mean squared error and 0-1 loss are used. The root mean squared error is an acceptable error measure for regression, while 0-1 loss is used for classification problems. Each dataset will have one fold that is used as a validation set that will be used to assess the accuracy or fitness of individuals (MLP network weights) in a population as training of the MLP network weights proceeds. The validation set will be used to assess root mean squared error or 0-1 loss after each generation, in order to produce plots of convergence rate. Average root mean squared error or 0-1 loss over the ten fold cross-validation tests, and five datasets will be used as the final measure of accuracy.

For each test that assesses the effect of a parameter on convergence rate or accuracy, all other parameters will be held constant while the parameter of interest is varied. The final hypotheses comparing convergence rate and accuracy of the algorithms versus one another will be conducted using the best performing parameter values from the previous tests. An analysis of variance (ANOVA) statistical hypothesis test will then be used to determine if there is a statistically significant difference between the accuracy and convergence rate of the tested algorithms.

5. Results

The results of the experiments are divided by dataset and hypothesis. The effect of varying key parameters in the evolutionary algorithms are shown first, followed by the final accuracy and error convergence plots achieved after parameter tuning was completed.

5.1 Parameter Variation

Within the genetic algorithm the mutation step size is determined by $N(0, \sigma)$. The result of varying σ on an example dataset is shown in the following table. Using an ANOVA test the difference in means is found to be statistically significant with a p-value of 0.037 and a significance level of 0.05. $\sigma = 1$ was selected as the final value to use in the subsequent accuracy and convergence tests.

Table 1: Genetic Algorithm σ Variation

	$\sigma = 0.5$	$\sigma = 1$	$\sigma = 2$
Mean RMSE	8.75	5.66	8.32
Std Dev	2.53	1.04	1.44

For differential evolution, the magnitude of the mutation is scaled by β. The following table illustrates the impact of varying β on the accuracy attained by the differential evolution algorithm on the abalone dataset. For this particular dataset, a p-value of 0.54 was calculated, which is not significant at a significance level of 0.05. None of the other four datasets had significant results at the significance level of 0.05.

Table 2: Differential Evolution β Variation

	$\beta = 0.1$	$\beta = 0.5$	$\beta = 0.9$
Mean RMSE	78.8%	77.6%	80.0%
Std Dev	3.1%	2.1%	5.6%

The effect of varying the population size on an example dataset is shown in the following tables. Differential evolution shows the effect of varying population size on the concrete slump dataset, the genetic algorithm shows the effect of varying the population size on the yacht hydrodynamics dataset, and the $\mu + \lambda$ evolution strategy shows the effect on the Wine dataset.

Table 3: Diff Evolution

Pop Size	Mean RMSE	Std Dev
7	5.17	1.16
35	5.31	1.11
70	5.83	1.93

Table 4: Genetic Algorithm

Pop Size	Mean RMSE	Std Dev
15	15.88	0.97
50	5.67	1.04
75	6.83	1.61

Table 5: $\mu + \lambda$ ES

Pop Size	Mean RMSE	Std Dev
100	33%	9%
200	30%	9.6%
300	35%	9.6%

For the genetic algorithm with a p-value of 0.00001 and a significance level of 0.05, there is a statistically significant difference between the accuracy achieved with a population size of 15 and the larger population sizes for the genetic algorithm. A smaller population size also performed poorly on the other datasets. A population size of 50 was chosen for the subsequent accuracy and convergence tests.

For varying the population size for differential evolution on the concrete slump dataset, a p-value of 0.57 was calculated, which is not significant at the 0.05 level. None of the other four datasets produced significant results at the significance level of 0.05.

Varying the population size used in $\mu + \lambda$ evolution strategy had a p-value of .813564 for the wine dataset. Changing the size of the population with this algorithm did not significantly effect the overall accuracy as measured by RMSE. Because of this, the population size $\mu = 100$ was used for the rest the tests used in this report.

5.2 Regression and Classification Accuracy

An ANOVA test is used for each dataset to assess if the differences in mean accuracy between algorithms are statistically significant.

Table 6: Yacht Hydrodynamics - Regression

Training Algorithm	Mean RMSE	Min RMSE	Max RMSE	Std Dev
Backpropagation	1.06	0.41	2.38	0.62
Genetic Algorithm	5.77	4.74	7.35	0.93
$\mu + \lambda$ ES	15.11	9.49	20.79	3.65
Differential Evolution	13.93	10.31	21.66	3.40

With a p-value of < 0.00001 and a significance level of 0.05 there is a statistically significant difference in accuracy among the training algorithms.

Table 7: Concrete Slump - Regression

Training Algorithm	Mean RMSE	Min RMSE	Max RMSE	Std Dev
Backpropagation	2.84	1.97	4.99	0.86
Genetic Algorithm	5.27	4.13	5.84	0.63
$\mu + \lambda$ ES	5.38	1.87	25.58	5.05
Differential Evolution	5.17	3.84	7.711	1.16

With a p-value of 0.00002 and a significance level of 0.05 there is a statistically significant difference in the accuracy among the training algorithms.

Table 8: Letter Recognition - Classification

Training Algorithm	Percent Incorrect	Min Incorrect	Max Incorrect	Std Dev
Backpropagation	28%	5%	35%	8.49%
Genetic Algorithm	78%	70%	86%	5.95%
$\mu + \lambda$ ES	85%	76%	92%	35%
Differential Evolution	91%	87%	94%	3.0%

With a p-value of < 0.00001 and a significance level of 0.05 there is a statistically significant difference in accuracy among the training algorithms.

Table 9: Abalone - Classification

Training Algorithm	Percent Incorrect	Min Incorrect	Max Incorrect	Std Dev
Backpropagation	71%	69%	76%	2.12%
Genetic Algorithm	76%	73%	80%	2.58%
$\mu + \lambda$ ES	77%	73%	79%	2.05%
Differential Evolution	77.4%	72%	83%	4.8%

With a p-value of 0.007 and a significance level of 0.05 there is a statistically significant difference in accuracy among the training algorithms.

Table 10: Wine - Classification

Training Algorithm	Percent Incorrect	Min Incorrect	Max Incorrect	Std Dev
Backpropagation	36%	11%	56%	12.58%
Genetic Algorithm	46%	28%	50%	6.73%
$\mu + \lambda$ ES	33%	22%	44%	8.5%
Differential Evolution	32.5%	16.7%	44.4%	9.6%

With a p-value of 0.79 and a significance level of 0.05 there is no statistically significant difference in the accuracy among the training algorithms.

5.3 Regression and Classification Error Convergence

Table 11: Iterations/Generations to Convergence

	Concrete		Yacht		Letter		Wine		Abalone	
	Mean	StDev	Mean	StDev	Mean	StDev	Mean	StDev	Mean	StDev
Backprop	733	204	816	205	1163	267	201	69	1195	504
Genetic Alg.	305	153	770	87	653	125	245	110	285	228
Diff Evol.	374	263	575	236	404	298	132	162	133	93
$\mu + \lambda$ ES	76	41	20	2	22	7	4	1	45	5

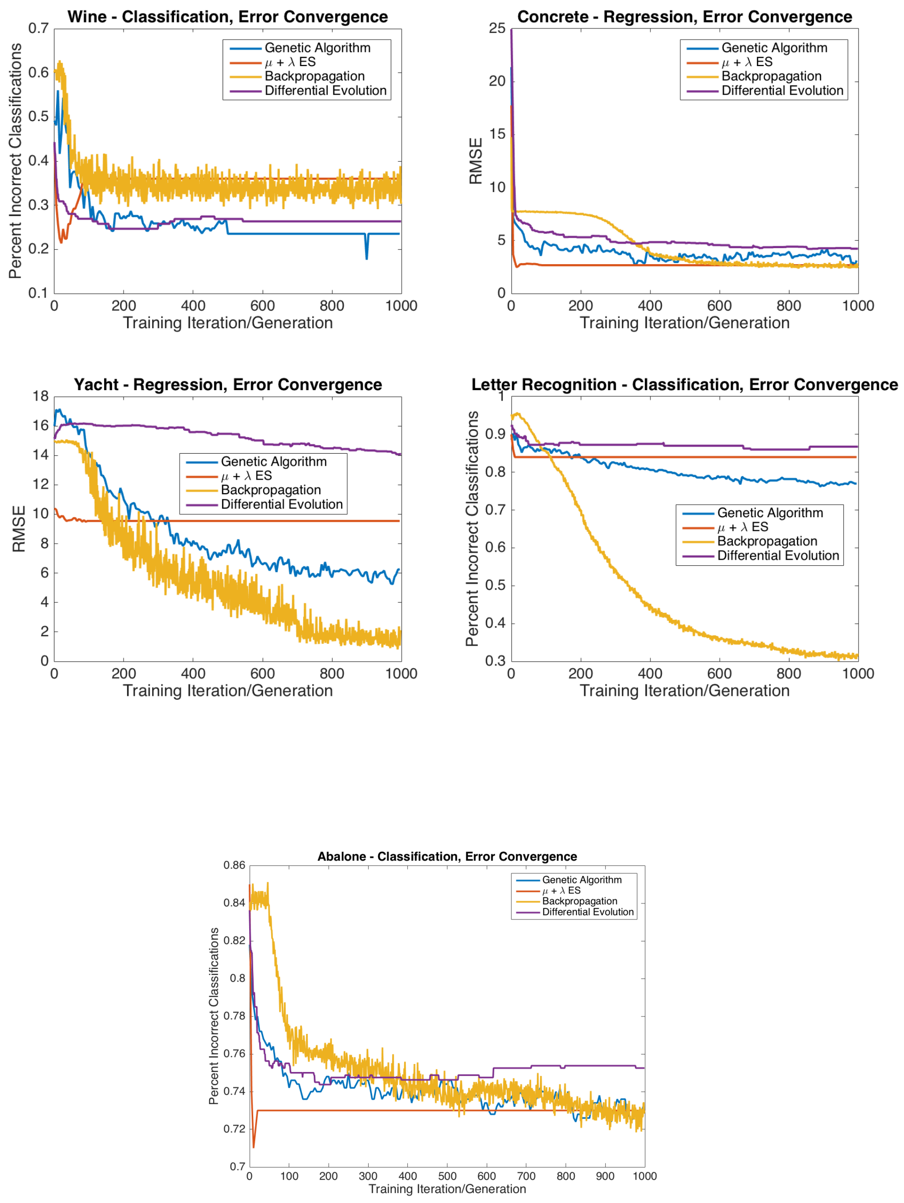

Figure 1: Error convergence for the various training algorithms. Plots are limited to 1000 training iterations/generations.

6. Discussion and Analysis

For the initial hypotheses, where the effect of varying various parameters is investigated, it is clear that varying the mutation parameter σ has an effect on the accuracy achieved by the genetic algorithm. This issue is discussed in (Kruse et al., 2016). It is likely that a mutation step size that is too small does not make large enough changes to the genes in an individual to make a meaningful move that would lead a population in a new direction within a search space. While a mutation step size that is too large, often reduces the fitness of "good" individuals, harming the efficiency of the search process. Thus, a balance must be found between large and small mutation step sizes. Essentially exploration and exploitation must be balanced properly. Since the $\mu + \lambda$ evolution strategy uses self-adaptive Gaussian mutation, poor mutation step sizes tend to be filter out of the population, and the effect of the initial value of σ is less pronounced.

Similar considerations must be taken into account when tuning β for the differential evolution algorithm. If β is too low, the mutation between parents and offspring will be too small to effectively explore the search space, and setting β too high can result in wild, unhelpful mutations (Gamperle et al., 2002). We see this pattern emerge to some degree in our results, with the algorithm performing best on most datasets when β is set to 0.5, and accuracy decreasing when β is set to 0.1 or 0.9. The effect is not as pronounced in this study as it is elsewhere in the literature, and it is important to note that while we do observe the expected pattern, none of the results in this study concerning the tuning of β were found to be statistically significant.

The parameter tuning process also showed that varying population size can have an effect on the accuracy of an evolutionary algorithm. Particularly when a population size is small there is not enough diversity within the population to explore the search space sufficiently. A small population often leads to premature convergence, which is detrimental to the final accuracy achieved by an evolutionary training algorithm.

Moving on to the hypotheses regarding convergence and accuracy of the training algorithms, the performance of the various training algorithms varied significantly depending on the dataset being used in the test. This was expected, as the datasets present varied numbers of features and instances as well as regression and classification problems, all of which create large differences in the difficulty of learning a dataset. However, the results showed that backpropagation was the most consistent training algorithm in terms of both convergence and accuracy. Looking at all five tested datasets backpropagation consistently achieved similar or better accuracy compared to the evolutionary approaches. The consistency seen in the backpropagation training algorithm was expected as backpropagation doesn't incorporate any randomness compared to the evolutionary methods which do incorporate a degree of randomness in the mutation and crossover operators. Among the evolutionary approaches, the genetic algorithm was the most consistent in terms of accuracy, and often achieved the best accuracy.

Looking at the convergence hypothesis we did observe a difference in convergence rates. Interestingly, the evolutionary algorithms typically converged faster than backpropagation. This was unexpected, as the randomness present in mutation and crossover was expected to slow down the convergence process. Despite the randomness present in the training process of the evolutionary algorithms the improvements between generations of the evolutionary training methods are often large if a "good" mutation or crossover is found, while the improvement between iterations during backpropagation is limited by the learning rate, which is purposely set small to avoid instability in the training. The small learning rate is most likely the reason slower convergence was observed during backpropagation. We also believe that the $\mu + \lambda$ evolution strategy was experiencing premature convergence due to how rapidly it converged compared to the other algorithms.

7. Conclusion

Unsurprisingly, parameter selection and the dataset in question can have a large influence on the accuracy and convergence rate of backpropagation and the evolutionary algorithms. The degree of randomness present when training a MLP network using evolutionary methods generally leads to greater inconsistency in the training process compared to backpropagation which does not incorporate the randomness that is seen in the mutation and crossover operators of the evolutionary algorithms.

Adjusting the rate of mutation through σ and β is crucial for balancing the exploration/exploitation trade-off. If mutation is too extreme, the offspring can often land outside of the relevant search space, delaying convergence, or leading to genetic drift. If the mutation is too mild, the algorithm can get stuck in a local minimum and converge prematurely, which likely explains the steep drop, then immediate leveling out present in some of the $\mu + \lambda$ results.

Ultimately, maintaining diversity in the population for the genetic algorithms is key to effectively exploring the search space and avoiding premature convergence. Initializing the population to a large enough size and ensuring the magnitude of the mutations is large enough (controlled by σ and β) are two of the most important considerations when tuning evolutionary algorithms.

References

Roger Gamperle, Sybylle Muller, and Petros Koumoutsakos. A parameter study for differential evolution. *3rd WSEAS International Conference on Evolutionary Computation*, pages 4841–4846, 2002.

Rudolf Kruse, Christian Borgelt, Christian Braune, Sanaz Mostaghim, and Matthias Steinbrecher. *Computational intelligence: a methodological introduction*. Springer, 2016.

Rylander and Gotshall. Optimal population size and the genetic algorithm. *Population*, 100(400):900, 2002.

Daniela Zaharie. A comparative analysis of crossover variants in differential evolution. *International Multiconference on Computer Science and Information Technology*, pages 171–181, 2007.