Neural Networks Trained by Population Based Algorithms

Andrew Kirby andy-kirby@live.com
Kevin Browder browderkevin54@gmail.com
Nathan Stouffer nathanstouffer1999@gmail.com
Eric Kempf erickempf123@gmail.com

Abstract

Multilayer Perceptron networks are supervised learning models that can be used for classification and regression. These networks are commonly trained using Backpropagation, however, Population Based Algorithms also may be used to train these networks. Each training method requires careful tuning and no algorithm consistently outperforms the others. Selection of training method likely depends on the nature of the model and information to learn. Additionally, Population Based Algorithms take more time to train than Backpropagation.

1. Problem Statement

Given data sets, the task is to compare the effectiveness of a feed forward Multilayer Perceptron (MLP) trained by Population Based Algorithms (PBAs) and Backpropagation. Specifically, there are three PBAs: the Genetic Algorithm (GA), Differential Evolution (DE), and Particle Swarm Optimization (PSO). Each training algorithm will be implemented and tested on the following six data sets.

The abalone data set contains features paired with the age of an abalone (the class). The 28 ages are closely related, potentially making classification difficult. The car data set contains attributes corresponding to 4 possible conditions of a car while the segmentation data set details 6 distinct subjects of outdoor photographs. The rest of the data sets are regression. The forest fires set contains the area burned by a forest fire. Most acreages are 0 with a few large values that might give the model trouble. The machine data set predicts the performance of a CPU, theres is an even distribution across its regression values. Wine quality contains the score of a wine with associated attributes. Several of the attributes are correlated, making regression more difficult (Dua and Graff, 2017).

Backpropagation was implemented and discussed in a previous project so only its results are displayed in this paper. Population Based Algorithms operate on an exploration-exploitation framework. The general idea is that members of the population begin by exploring the search space and, as a run continues, gradually shift to exploiting the most fit areas of the search space. In general, PBAs are considered to be methods for global optimization and do not get "caught" in local minima.

1.1 Hypothesis

Population Based Algorithms are expected to outperform Backpropagation because PBAs are less prone to getting caught in local minima. Of the PBAs, the GA will perform better than DE and PSO because of its unique mutation operator. Mutation in the GA, through a creep term, introduces an entirely random component to the direction an "offspring" moves from its "parent," which is more likely to introduce diversity later in a run than the other two Population Based Algorithms. This is because the stochastic portion of PSO affects only a step's magnitude in the search space and, for DE, the direction a member "moves" in is entirely determined by randomly selected members currently in the population.

While mutation in the GA has the possibility of introducing diversity, it also makes the GA more similar to a random walk. This will give the GA the slowest convergence rate. Differential Evolution will converge the quickest of the three PBAs. This is because DE only accepts a new member of the population if it is more fit than its direct parent. This gives quicker convergence than the other two PBAs. However, Backpropagation will converge quicker than any of the PBAs because, for a local region, it moves in the direction that most increases fitness.

2. Algorithms

Population Based Algorithms typically reference individual members of populations as vectors associated with a fitness value. Each element in a vector corresponds to some value in a model that is being trained by a PBA.

Recall that the output of a Neural Network is entirely dependent on the values of the weights within in the network. So a neural network can be converted to vector form by listing its weights. Since weights are real-valued, if w represents the number of weights that a neural network contains, then the vector is a point in the Euclidean Space $\mathbb{R}^w$. The fitness associated with the vector is the accuracy or MSE (respectively for classification and regression) of a network on a training data set.

This sets up the search space of weights and the objective function based on the network's fitness.

2.1 Genetic Algorithm

The Genetic Algorithm (GA) is an evolution-inspired process often used training a model. The algorithm represents possible solutions as chromosomes—vectors of gene values that parameterize a model. Given a population of randomly initialized chromosomes, the GA applies cycles of selection, recombination, mutation, and replacement to individuals within the population. Chromosomes with better fitness are expected to survive through these generations, preserving desirable genes as the population searches the fitness landscape (Whitley, 1994).

In each generation, individuals are selected for recombination. Recombination generates an intermediate population by applying crossover between parents, an operator which randomly swaps genes between chromosomes at some rate P_c. Next, the mutation operator is applied to each individual in the intermediate population, randomly modifying genes

at some rate P_m. The intermediate population then completely or partially replaces the original population, completing the generation.

Selecting parents of the intermediate population often follows fitness proportionate selection, rank-based selection, or tournament selection. Fitness proportionate selection selects parents with probability equal to the fitness of the individual divided by the sum of the fitnesses of all individuals in the population. Rank-based selection selects parents with probability

$$P(\vec{x}_i) = \frac{2}{|P|} * \frac{|P| - rank(\vec{x}_i)}{|P| + 1}$$

where $|P|$ is the size of the population and $rank(\vec{x}_i)$ is the ranking of the individual $\vec{x}_i$ based on fitness. Ranked-based selection tries to minimize the high selection pressure exhibited by fitness proportionate selection. Another method that balances selection pressure is tournament selection, which selects k individuals from the population at random and chooses the one with the best fitness.

One-point, two-point, and uniform crossover are common implementations of crossover. With probability P_c, one-point crossover selects a random point in a chromosome, swapping all values of two parent individuals after the crossover point. Two-point crossover swaps values between two selected crossover points. Uniform crossover differs by swapping values at any given point with probability P_c. Each crossover operator contributes to the search by creating children that may inherit more ideal gene combinations from both parents (Whitley, 1994).

Mutation operators visit every gene, mutating with probability P_m. Specific mutation operators rely on the gene encoding. If genes are binary, mutation flips the bits. If genes are categorical, a new category is chosen from the domain. If genes are real-valued, mutation may generate a new random value or implement creep. With creep, the new gene value x_i' is given by

$$x_i' = x_i + N(0, \sigma_i)$$

where σ_i is a tunable standard deviation. Mutation helps the search by reinforcing diversity and potentially jumping chromosomes to a completely new spot in the search space (Whitley, 1994).

Replacing the original population with individuals from the intermediate population may follow generational or steady-state replacement. Generational replacement replaces the entire population with the intermediate population. This form of replacement may cause dramatic changes between generations and exhibits a high selection pressure. Steady-state replacement applies a gradual change between generations by replacing only a proportion of the original population with individuals from the intermediate population.

2.2 Differential Evolution

Differential Evolution (DE) is another evolution inspired algorithm with similarities to the GA as both use crossover and mutation on chromosomes. It cycles through selection, mutation, crossover, and replacement on individuals in the population.

The DE begins with a population P of vectors that are randomly generated. A target vector $\vec{x}_t$ is randomly selected from the population. For mutation, a set of vectors $\{\vec{x}_1, \vec{x}_2, ..., \vec{x}_n\}$ with odd cardinality is randomly selected from the data set. These vectors

are combined using the formula below to create a trial vector $\vec{x}_p$ where β is a tunable parameter (Price et al., 2006).

$$\vec{x}_p = \vec{x}_1 + \beta(\vec{x}_2 - \vec{x}_3) + ... + \beta(\vec{x}_{n-1} - \vec{x}_n)$$

Uniform crossover is then applied to the target and trial vector according to a tuning parameter. The fitness for the target and trial vectors are then calculated and if the trial vector has a higher fitness, it replaces the target vector. A generation is complete when this process has iterated through the size of the population (Price et al., 2006).

2.3 Particle Swarm Optimization

PSO differs from the GA and DE in that there is no parent-offspring relationship. Instead, PSO consists of particles that interact and move around the search space to find the optimal solution.

A swarm consists of N individuals each with a position $\vec{x} \in \mathbb{R}^w$ and a fitness value. PSO then uses an iterative update rule to change each individual's position. The velocity for an individual is named $\vec{v} \in \mathbb{R}^w$, which is composed of 3 summed components.

The first component is inertia. For a given iteration's velocity $\vec{v}_t$, inertia is given as $\omega * \vec{v}_{t-1}$ where $\omega \in \mathbb{R}$ can be referred to as the inertia weight. Note that a "large inertia weight facilitates a global search while a small inertia weight facilitates a local search" (Shi and Eberhart, 1999).

Second, there is a cognitive component. Each particle records its own best performing position $\vec{x}_c \in \mathbb{R}^w$ across all iterations. With $\vec{x}$ still representing the particle's current position, this component is given as $c_c * r_c * (\vec{x}_c - \vec{x})$ where r_c is randomly selected from $(0, 1) \subset \mathbb{R}$ and c_c is a tuned parameter (Eberhart and Kennedy, 1995).

The final component is social. For this component, a topology is declared on which particles can communicate between each other. Where if two particles a, b can communicate, then a knows both the current position and fitness of b. Similarly, b knows the current position and fitness of a.

Two common topologies used in PSO are the global and local topologies. The global topology allows each particle to communicate with every other particle. The local topology declares that each particle can communicate with only two other particles. Explicitly, place the particles in the order of initialization. Then, particle p_k communicates with both p_{k-1} and p_{k+1} (with $p_{-1} \equiv p_N$ and $p_{N+1} \equiv p_0$). This communication persists across all iterations (Eberhart and Kennedy, 1995).

Now, take an arbitrary particle p and let T_p denote the set of particles that can communicate with p. Let $\vec{x}_s \in \mathbb{R}^w$ denote the position of the most fit particle $q \in T_p$. Then, the value of the social component is given as $c_s * r_s * (\vec{x}_s - \vec{x})$ where r_s is randomly selected from $(0, 1) \subset \mathbb{R}$ and c_s is a tuned parameter.

Thus the velocity for each particle is given as the sum

$$\vec{v}_t = \omega * \vec{v}_{t-1} + c_c * r_c * (\vec{x}_c - \vec{x}) + c_s * r_s * (\vec{x}_s - \vec{x})$$

The position of each particle is now updated according to $\vec{x}_t = \vec{x}_{t-1} + \vec{v}_t$. This process is iteratively repeated for each particle until the values of the vectors converge or a maximum number of iterations is reached (Eberhart and Kennedy, 1995).

3. Experiment

3.1 Evaluation Metrics

Classification data sets are evaluated with accuracy and mean squared error (MSE). The MSE metric implemented measures the squared error between the predicted and actual class distributions. In concept, this is similar to a Brier score but slightly different in computation. Accuracy indicates how well the algorithm classifies individual examples.

To evaluate regression data sets, mean error (ME) and MSE are used. Both are computed using z-scores (the number of standard deviations from the mean) so that comparisons can be made between data sets. MSE squares the distance between a real and predicted value, the squares are then averaged over the entire testing set. ME is computed similarly, but will not square the difference. MSE emphasizes the effect of outliers while ME captures whether the learner tends to over or underestimate the values in the test set.

3.2 Preprocessing Choices

First, all the examples in the data set are randomly scrambled and then assigned to sets for ten-fold cross validation. All categorical variables are converted to integers and the preprocessor also generates a similarity matrix for each categorical variable that is used for determining distances between categorical variables. All numerical variables are normalized between 0 and 1. The data sets did not contain any missing variables.

3.3 Algorithm Choices

3.3.1 Genetic Algorithm

Any GA implementation requires specific crossover, mutation, selection, and replacement methods. For this implementation, uniform crossover, creep mutation, rank-based selection, and steady-state replacement are used. Uniform crossover was selected for its more disruptive tendencies, a desirable trait for low population implementations (Whitley, 1994). Creep mutation was selected for its stability. Rank-based selection is expected to provide practical convergence times while also supporting diversity in the population. A bonus is that this method mitigates the large decrease in selective pressure as the members of the population approach similar fitnesses (Whitley, 1994). Additionally, rank-based selection has no tunable parameters. Steady-state replacement is implemented by generating an intermediate population one fourth of the size of the original population. Then, intermediate population members replaced random members in the original population, but only if they had better fitness. A population of size 64 was used, providing sufficient performance (Whitley, 1994).

3.3.2 Differential Evolution

The first decision to be made for Differential Evolution is the population size. This is set to 100 in an attempt to maintain diversity in the population. The number of difference vectors used in mutation must also be determined. A single difference vector was selected because it has been experimentally shown to produce good results (Gämperle et al., 2002).

3.3.3 Particle Swarm Optimization

Three decisions were made for PSO. First, the global topology was declared on how particles in the swarm communicate. The global topology is the original topology used in PSO and was chosen over the local topology because it has been reported that the local topology requires more iterations to reach a specified error level (Eberhart and Kennedy, 1995). Second, the ω term in the velocity equation decreases linearly over the course of a run. Experimentally, a linearly decreasing ω from 0.9 to 0.4 provides a nice exploration-exploitation balance for PSO (Bansal et al., 2011). Finally, the size of the population is 100 individuals. This has been experimentally shown to perform well (Shi and Eberhart, 1999).

3.4 Tuning

3.4.1 Genetic Algorithm

Tuning the GA required finding optimal standard deviations used for mutation creep, P_c, and P_m. Setting the standard deviations at 100 times the initial boud for network weights was found to be optimal. A grid search for $(P_c, P_m) \in \{0.1, 0.05, 0.01, 0.005\} \times \{0.05, 0.02, 0.01, 0.005\}$ tuned crossover and mutation rates. The optimal values are displayed in Table 1.

	abalone	car	segmentation	forest fires	machine	wine quality
P_c	0.1	0.1	0.1	0.1	0.1	0.1
P_m	0.005	0.005	0.05	0.05	0.02	0.02

Table 1: Optimum Parameters for the Genetic Algorithm

3.4.2 Differential Evolution

Tuning for DE requires tuning the crossover rate, denoted P_c, and mutation denoted, β. A grid search for $(P_c, \beta) \in \{0.25, 0.5, 0.75\} \times \{0.5, 1, 1.5\}$ each dataset. The optimal values are displayed in Table 2.

	abalone	car	segmentation	forest fires	machine	wine quality
P_c	0.5	0.25	0.25	0.25	0.5	0.5
β	1.5	1.5	1.5	1.5	0.5	1

Table 2: Optimum Parameters for Differential Evolution

3.4.3 Particle Swarm Optimization

Tuning for PSO requires setting the values of c_c of c_s. Experimentally, PSO performs best with $c_c = c_s = 2$ (Shi and Eberhart, 1999). As such, a grid search was performed on $(c_c, c_s) \in \{1, 2, 3\} \times \{1, 2, 3\}$ for each data set. Then the optimum parameters were selected based on fitness of the produced networks (accuracy for classification and MSE for regression). Specific values are shown in Table 3.

	abalone	car	segmentation	forest fires	machine	wine quality
c_c	3	2	2	2	1	1
c_s	2	3	1	2	3	2

Table 3: Optimum Parameters for Particle Swarm Optimization

4. Results

For the final run, optimal parameters were selected for each data set. Results are shown in Fig. 1, 2, and 3. In general, Backpropagation outperformed the PBA on classification data sets while the opposite is true for regression data sets. This partially realizes the hypothesis that Population Based Algorithms would perform better than Backpropagation. The PBAs may outperform Backpropagation on regression sets because it is often required that output layer weights are large, which may be easier for a PBA (with many members in a population) to find.

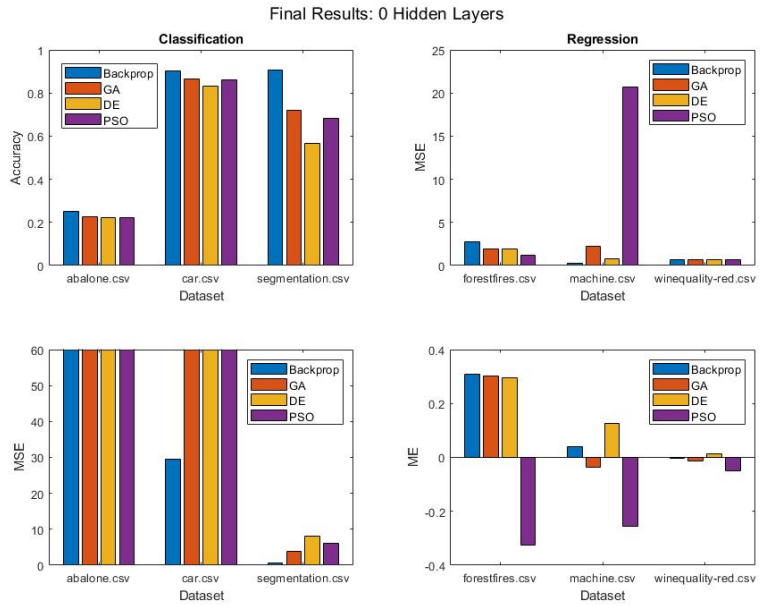

Figure 1: Final Results of a 0 Hidden Layer MLP. Color indicates training algorithm type.

The greatest difference in performances between the two algorithms is shown on the forest fires data set in Fig. 2 where the MLP has 1 hidden layer. For a network trained with Backpropagation, the MSE was 26 while the MSE for networks trained by PBAs is less than 3. The most likely explanation is that PBAs were able to pass by the local optima that Backpropagation found for the weight configuration.

The performance of every training method with 2 hidden layers is particularly interesting. Across all training methods, 2 hidden layers resulted in the worst performance, likely because of the increased search space dimensionality. Backpropagation and each of

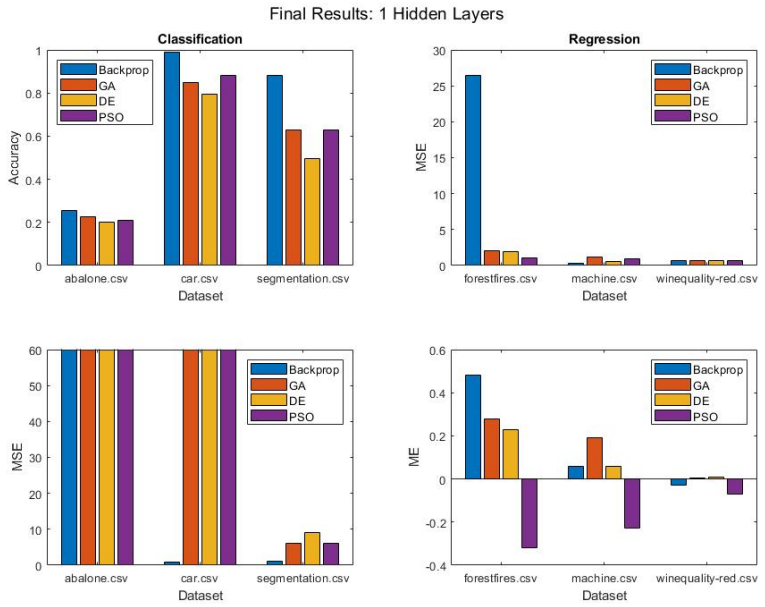

Figure 2: Final Results of a 1 Hidden Layer MLP. Color indicates training algorithm type.

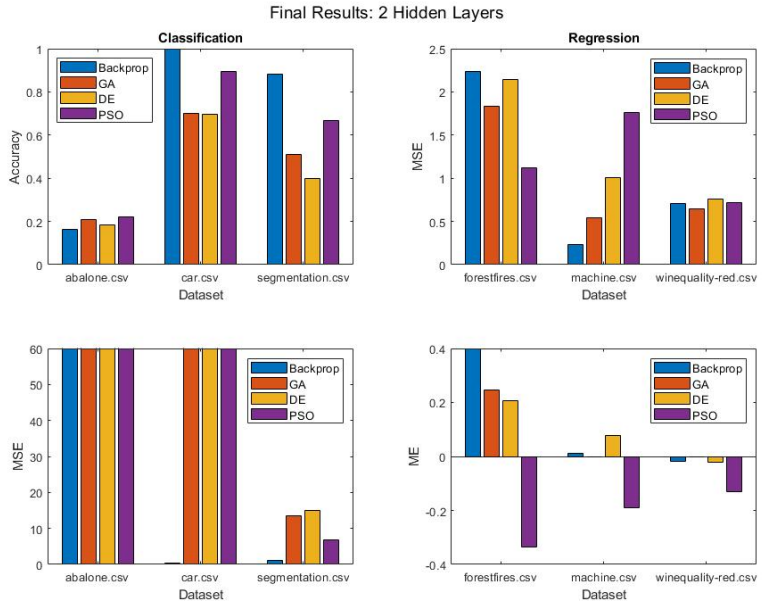

Figure 3: Final Results of a 2 Hidden Layer MLP. Color indicates training algorithm type.

the PBAs are not immune to this limitation, and there appears to be no clear best training method when network depth is increased.

There also seemed to be no preference between Population Based Algorithms. The best performing algorithm varied by both data set and layer.

In general, the results show that algorithm performance is highly situational. It should be noted that the relative performances of each algorithm could likely be changed by further tuning parameters. Thus, the PBAs may have underperformed because of the multitude of decisions and parameters.

In terms of convergence rate. The hypothesis that Backpropagation would be fastest of all training algorithms was found to be true. DE was the next fastest, and, interestingly, the GA converged rather quickly, losing most of its diversity within 100-200 generations. Surprisingly, Particle Swarm Optimization took the longest time to converge. This is attributed to the linearly decreasing inertia weight. Starting at 0.9 heavily focuses the members on exploring the search space and gradually decreasing the inertia weight leaves exploitation for the end of a run.

5. Conclusion

Feed forward neural networks are complex models that can universally approximate functions. Training these networks with Backpropagation or Population Based Algorithms is effective. In general, no training method consistently outperforms the others. Every training method still requires critical tuning and suffers as model complexity increases.

One conclusion that can be drawn is that Population Based Algorithms train neural networks slower than Backpropagation.

Ultimately, selection between training algorithms depends on the model as well as the information to be learned. Certain data may be more conducive to hill climbers such as Backpropagation. Other data may benefit from the theoretically more "global" search methods implemented in Population Based Algorithms. Essentially, there is still no free lunch.

References

Jagdish Chand Bansal, PK Singh, Mukesh Saraswat, Abhishek Verma, Shimpi Singh Jadon, and Ajith Abraham. Inertia weight strategies in particle swarm optimization. In *2011 Third world congress on nature and biologically inspired computing*, pages 633–640. IEEE, 2011.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43. Ieee, 1995.

Roger Gämperle, Sibylle D Müller, and Petros Koumoutsakos. A parameter study for differential evolution. *Advances in intelligent systems, fuzzy systems, evolutionary computation*, 10(10):293–298, 2002.

Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

Yuhui Shi and Russell C Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1945–1950. IEEE, 1999.

Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, Jun 1994. ISSN 1573-1375. doi: 10.1007/BF00175354. URL `https://doi.org/10.1007/BF00175354`.