Project 4

Joshua Aney joshua.aney@student.montana.edu
Owen Cool owen.cool@student.montana.edu
Nicolas Perl nicolas.perl@student.montana.edu

Abstract

Population based algorithms are fundamental approaches in the field of evolutionary computation that can be used to solve complex optimization problems through an iterative, population centered approach. Our experiment explored the relative performance of feedforward network weight training through backpropagation, the genetic algorithm, differential evolution and particle swarm optimization when tested on 6 different datasets from the UCI machine learning repository, including both classification and regression problems. Using 10-fold cross-validation, we found that the regression models performed significantly worse relative to their respective null models than our classification models did.

1 Introduction

Population-based algorithms are optimization techniques that are used in the context of training models, in our case a feedforward neural network. They use multiple solutions simultaneously and are often inspired by biological processes or social behavior. Within this project, in addition to backpropagation, three popular algorithms in this field will be applied:

The **genetic algorithm with real-valued chromosomes** trains the neural net by creating an initial population of individuals (real valued in our case). Each individual in a population is evaluated by applying a fitness function, which assigns a score based on how close the individual is to the optimal solution. Individuals are then selected for reproduction based on their fitness using tournaments. These work by selecting k random individuals without replacement and taking the best out of those. In cross-over, selected individuals combine their chromosomes to form offspring. Cross-over swaps parts of chromosomes between pairs to mix genetic information. In case of real-valued parents, the following arithmetic recombination given to parents is performed.

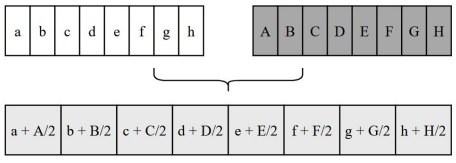

Figure 1: Cross-over of real-valued parents.

This process creates new solutions that might inherit the best features of both parents. To maintain diversity in the population and prevent premature convergence, mutations are introduced in some individuals. In case of real-valued parameters, a Gaussian distribution with a mean of 0 is uses to add some random noise.

$$x_i^{'} = x_i + \mathcal{N}(0, \sigma_i) \tag{1}$$

The offspring from crossover and mutation replace some of the current population.

This process is iterated until it meets a required fitness, reaches a maximum number of generations, or a stagnation point where no significant improvement occurs.

Differential evolution starts by creating a random initial population of vector-represented individuals within certain bounds. Every individual in the population is evaluated using a fitness function. Mutation proceeds from generating a donor vector from a set of trial vectors: Select 3 random, distinct vectors of an individual and calculate the weighted difference of 2 and add to the third by using

$$v_j = x_j^1 + \beta\left(x_j^2 - x_j^3\right) \quad \text{where} \quad \beta \in [0, 2] \tag{2}$$

Then, generate an offspring vector as a combination of the donor vector and a member of the population. Elements are selected using a "binomial" crossover, where the selection probability is α. The offspring vector is only selected if it is an improvement over the original vector. This process is also iterated until a certain termination criterion is met.

Particle swarm optimization PSO starts by randomly initializing a swarm of particles, each with their own position (potential solution) and velocity (how it moves through the search space). It continues to evaluate each particle's fitness and update its personal and the swarm's global best. Using these values and a few random factors, each particle's velocity and then its position is updated.

$$v(x_t) = \omega v(x_{t-1}) + c_1 r_1(\mathbf{pbest}(x_{t-1} - x_{t-1}) + c_2 r_2(\mathbf{gbest} - x_{t-1}) \tag{3}$$

Repeat this process until a stopping criterion is met.

2 Problem Statement and Hypothesis

Our experiment was conducted on 6 datasets from the UCI Machine Learning repository, namely "Wisconsin Breast Cancer," "Glass Identification", "Small Soybean," "Forest Fires," "Relative CPU Performance," and "Abalone." Our research focused on the question, " How do different training algorithms for densely-connected feedforward networks perform on classification problems with zero, one, and two hidden layers?"

We hypothesized that datasets with more dimensions would experience significant performance decreases relative to backpropagation when using the genetic algorithm, differential evolution, or particle swarm evolution, as the number of weights in these datasets' corresponding neural networks increases with the number of dimensions in the dataset. Thus, the size of the search space increases exponentially with the number of dimensions

in the dataset, causing population-based algorithms to struggle to fully explore the search space, and hurting performance. Furthermore, we hypothesized noisy datasets would perform better when using the genetic algorithm, differential evolution, or particle swarm evolution, as we believed the noise in the dataset would lead to a greater variability in model performance within the search space, and population-based algorithms would be able to navigate this more effectively. Similarly to the previous experiment using neural networks, we hypothesized that datasets that are less complex and well separated will perform better with less complex models (i.e less layers), regardless of the training algorithm used, as we saw significant performance decreases on datasets of this type using backpropagation.

We also hypothesized that the genetic algorithm and differential evolution would perform better on noisy datasets relative to particle swarm optimization, for similar reasons that we expected the population-based algorithms in general to perform better on such datasets. Specifically, we believed that the learning process of the genetic algorithm and differential evolution emphasize exploration more heavily than exploitation (due to their use of mutation in the learning process), and thus would navigate the more volatile search space more effectively.

We further hypothesized that the CPU performance dataset would have a performance increase when using the genetic algorithm, differential evolution, and particle swarm optimization because of the noise and skew of the dataset. The CPU performance dataset has a large amount of noise compared to the amount of features and is skewed toward 0. The forest dataset also has a significant skew toward 0 and a large amount of noise, but due to its very large number of dimensions, we believed that any performance increases it saw using population-based algorithms relative to backpropagation would be counteracted by the performance decrease it saw from its high dimensionality.

Lastly, we hypothesized that all the training algorithms would perform better on classification datasets than regression datasets, as what we are learning is "simpler". Rather than trying to learn a function defined throughout the feature space as in the case of regression, we are instead learning a limited number of partitions in the feature space to classify our data points. While the results for these different problem types are difficult to compare, we believed that by comparing each dataset's results to the performance of the null model for that dataset, we would find that we were much more likely to outperform the null model on classification problems than regression. An important caveat to this is that a greater number of classes would make these partitions harder to learn, and thus the glass dataset would likely underperform relative to the other classification datasets.

3 Experimental Approach and Program Design

We approached the problem stated above in a very sequential manner. Our first task was to create a hypothesis based on our prior knowledge for each of the datasets. We then created a design document that would help us formulate our approach on how we were going to attack the tasks at hand. This document contained instructions and design figures for how we were going to structure our code and test our hypotheses. This was then followed by understanding and coding our implementation of the genetic algorithm, differential evolution, and particle swarm optimization. We used the base neural network class from the previous project as the structure for the neural network for these algorithms.

These algorithms were set up to train neural networks with an arbitrary number of hidden layers and an arbitrary number of nodes in each layer. The code for each of the algorithms is set up by having an algorithm class and then methods to help complete the actual algorithm. The class took in a set of hyperparameters, as well as the training data and labels in the constructor, and then there was a train() method that would be responsible for returning the weight vector. This weight vector would then be passed into the network then the network would be able to evaluate the test data that is provided.

For the genetic algorithm, there is a population that is initialized in the constructor and then there is crossover and mutation that occurs on members of the population. The rate at which crossover and mutation occurs is meant to be tuned. The individuals are then updated based off of the update function specified above in the introduction. This process then occurs over a set number of iterations until the performance converges. This process is then similar for the differential evolution algorithm. The main difference is the update function. This update function is specified above and will occur every iteration that the algorithm is run. The particle swarm optimization algorithm is set up quite differently though. There is a population of particles that is initialized at the constructor, and each particle has a personal and global best. These are then weighted by the personal best weight and global best weight hyperparameters which need to be tuned. There is then a velocity vector that is created for each particle. This velocity vector is how the particle's positions get updated. Finally, there is also an inertia hyperparameter that is also meant to be tuned. This is then used to weight the actual velocity and move the particle faster or slower. The final update function is specified in the introduction. For all of the tuning, a basic grid search will be performed over all of the hyperparameters to find the best combination of hyperparameters.

Outside of any of the network or algorithm classes, there are dataset classes that are meant to do any data preprocessing and normalization. These classes then have methods that will return the data to then be passed into fold functions which will then be passed into the algorithm classes. These fold functions are meant to split up the data into stratified folds which are then used in the cross validation functions. There are also hyperparameter tuning functions which are meant to find the optimal set of hyperparameters for a certain dataset.

All of these functions are meant to work together to pass the data through an algorithm class which will then pass a weight vector into the network class. Once the network has the weight vector, it will be able to predict values or classes for the specified dataset.

4 Experiment Results

The genetic algorithm, and differential evolution performed very similarly to the backpropagation algorithm for all layer sizes. The particle swarm optimization algorithm performed significantly better than the other training algorithms on one and two hidden layers. It performed about the same on zero hidden layers.

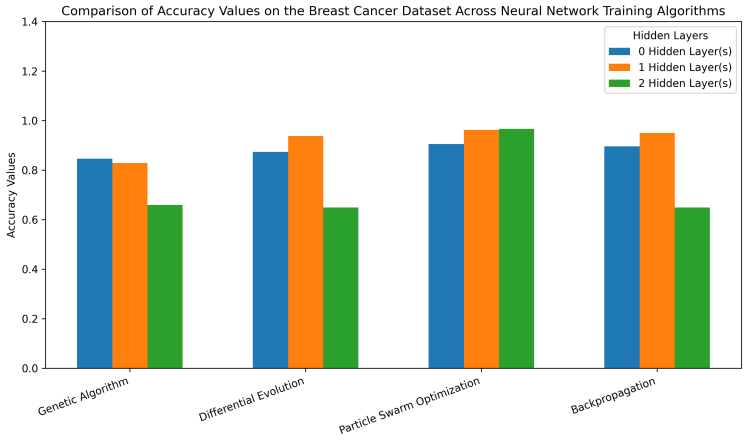

Figure 2: This bar graph represents the accuracy values of the individual training algorithms when testing the breast cancer dataset with 0, 1 and 2 hidden layers.

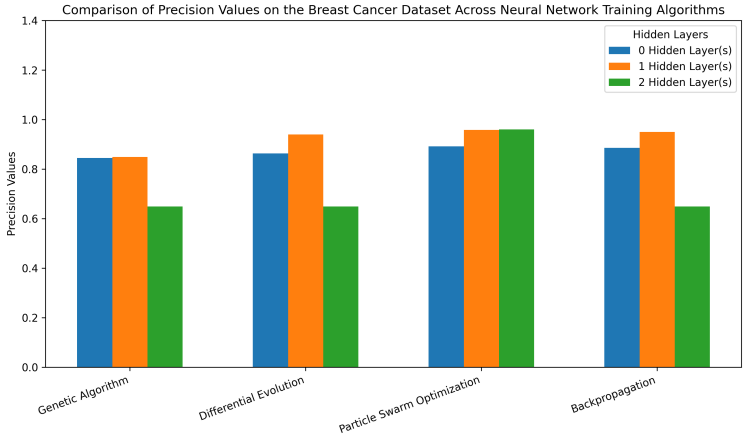

Figure 3: This bar chart shows the precision values of the individual training algorithms when testing the breast cancer dataset with 0, 1 and 2 hidden layers.

All of the algorithms performed very similarly on the glass dataset. The only outlier was with zero hidden layers where particle swarm optimization out performed all the other models besides backpropagation.

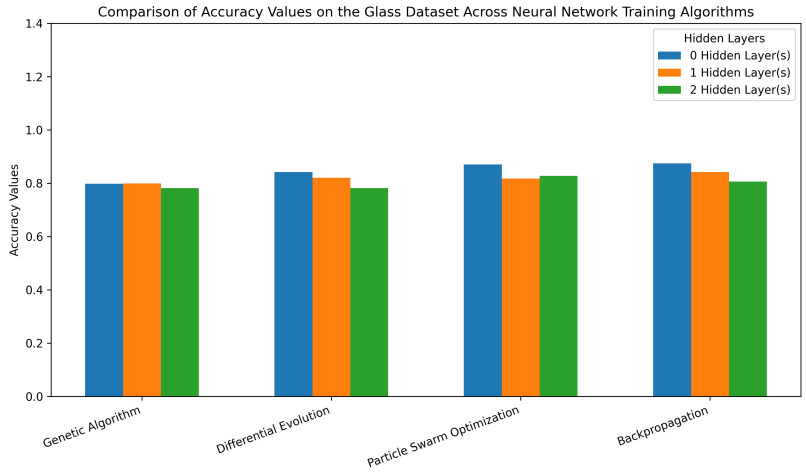

Figure 4: This bar graph represents the accuracy values of the individual training algorithms when testing the glass dataset with 0, 1 and 2 hidden layers.

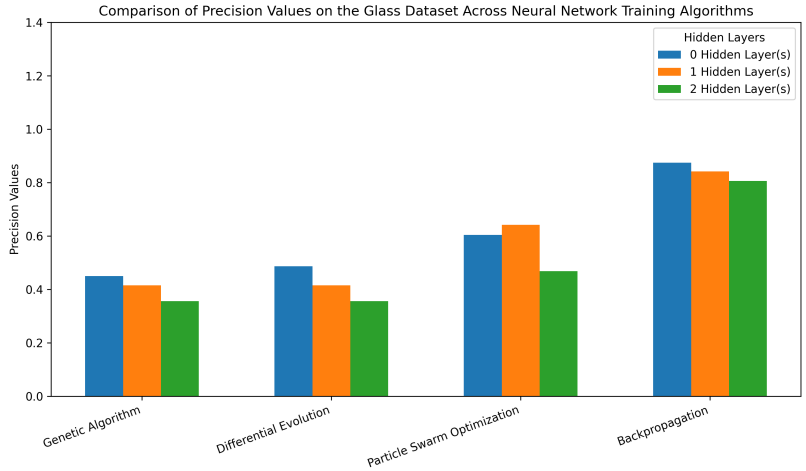

Figure 5: This bar chart shows the precision values of the individual training algorithms when testing the glass dataset with 0, 1 and 2 hidden layers.

The soy bean dataset performed slightly worse on all of the population based algorithms. The performance increased with backpropagation was very minimal though.

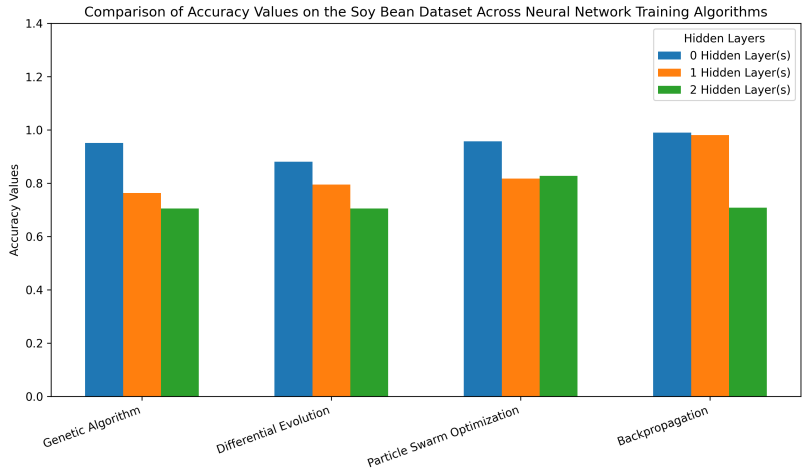

Figure 6: This bar graph represents the accuracy values of the individual training algorithms when testing the soy bean dataset with 0, 1 and 2 hidden layers.

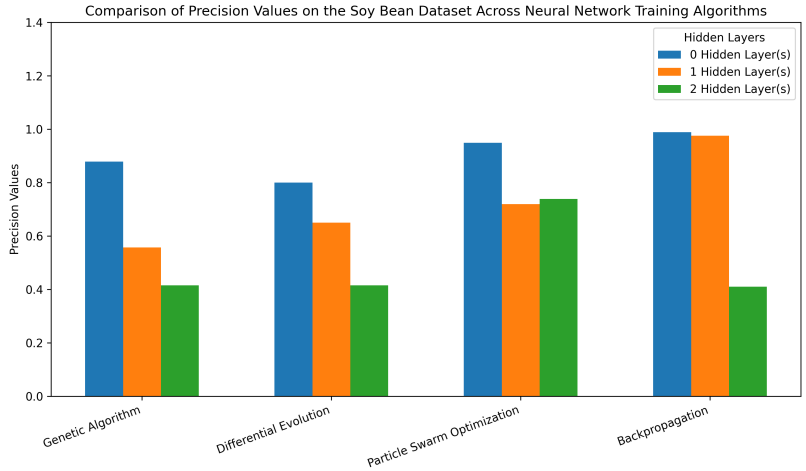

Figure 7: This bar chart shows the precision values of the individual training algorithms when testing the soy bean dataset with 0, 1 and 2 hidden layers.

The abalone dataset performed significantly worse when using the genetic algorithm and differential evolution. Particle swarm optimization performed similarly to backpropagation. The particle swarm optimization significantly out performed backpropagation when using one and two hidden layers.

Particle swarm and differential evolution performed significantly worse than the results from backpropagation on the relative CPU performance dataset. The genetic algorithm did produce comparable results to backpropagation only being slightly worse.

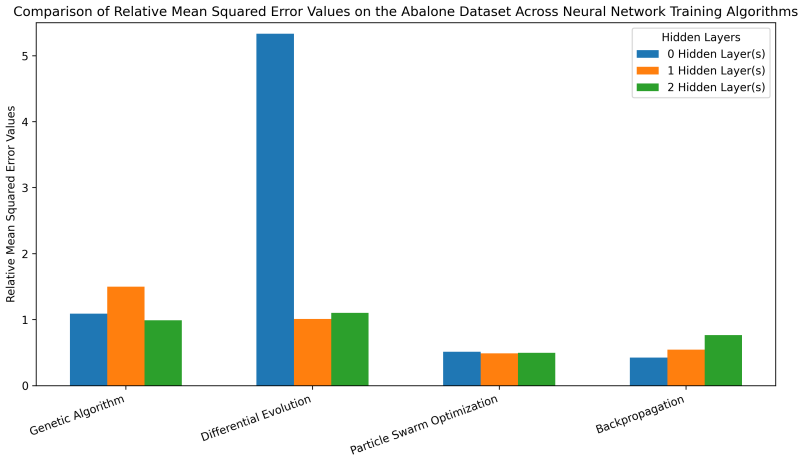

Figure 8: This bar chart shows the relative mean squared error values of the individual algorithms when testing the Abalone dataset with 0, 1 and 2 hidden layers.

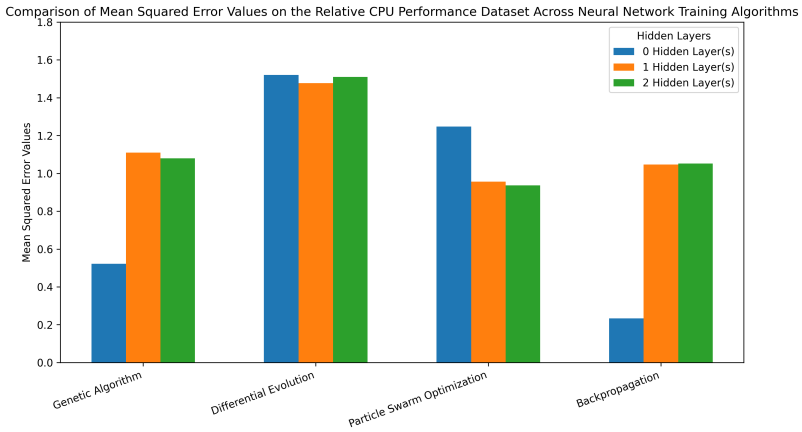

Figure 9: This bar chart shows the relative mean squared error values of the individual algorithms when testing the Machine dataset with 0, 1 and 2 hidden layers.

All of the training algorithms performed poorly on the forest fire dataset. The performance between layer sizes was similar across all training algorithms.

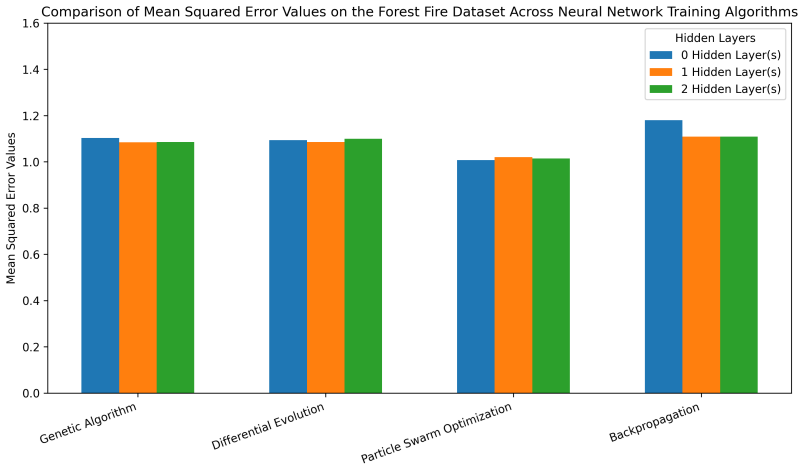

Figure 10: This bar chart shows the relative mean squared error values of the individual algorithms when testing the Forest dataset with 0, 1 and 2 hidden layers.

5 Discussion

As shown in figures 2 to 7, our regression models performed significantly worse relative to their respective null models than our classification models did. While this is consistent with our hypothesis, the relative differences in performance are much more significant than we initially expected, and thus require an explanation outside the one presented in our hypothesis. Almost every model performed roughly as well as the null model, achieving a relative mean squared error near one. However, the zero hidden layer neural network trained on the Abalone dataset using differential evolution significantly underperformed relative to the null model, achieving a relative mean squared error around 5. Further, the neural networks trained on the Abalone dataset using particle swarm optimization all outperformed the null model by a large margin, achieving relative mean squared errors around 0.5, as did the one hidden layer network trained on the CPU performance dataset using the genetic algorithm. Thus, we can see that these models do occasionally perform as intended, but can also underperform to the null model, or even underperform past the null model. Further, the regression models trained using population-based algorithms often underperformed relative to equivalent models trained using backpropagation, while classification models trained with population based-algorithms performed roughly as well as their backpropagation equivalents.

While there are certainly dataset and model-specific reasons for this, the inconsistency of these results makes those reasons hard to decipher. A more illuminating question is to examine our experimental approach and its impacts on our models' performances. The population-based training algorithms we used have many variations in varying degrees of complexity, and the implementations we used were the simplest variations of these algorithms. Thus, there are likely many tools that could be used to address these performance issues. For instance, we could have used different methods for selection and crossing over for the genetic algorithm, and treated the Gaussian distribution bandwidth as a hyperpa-

rameter to be tuned. This may have increased our performance on some of these datasets as different members of the population are selected for reproduction, different offspring are created, and different amounts of noise are added to offspring during mutation. A key strength of these algorithms is their customizability, and experimenting with customization to the fullest extent was not in the scope of our experiment, so we experienced significant performance losses. Later experiments should focus on research questions surrounding the performance of different versions of these population-based algorithms.

The classification models performed very well relative to the regression models, and outperformed the null model in all cases, as shown in figures 8 to 10. Further, they performed roughly as well as the models trained using backpropagation. Similar conclusions can be drawn here as in the regression models, though. If we were to customize our training algorithms to our specific datasets more carefully, we likely would see significant improvements in performance. It is worth noting, though, that the ease of implementation of these population-based algorithms makes their comparable performance to backpropagation very impressive. Further, these results affirm our conclusions from previous experiments that neural networks using more layers tend to perform worse on less complicated, less noisy datasets. The change in performance as more layers are added across different training algorithms affirms that the issue with more layers is not in the training algorithm, but rather the structure of the neural network.

6 Conclusion

While our results for classification datasets were consistent with our hypothesis, our results for the regression dataset were not. However, these results do roughly align with theory and practice regarding the training algorithms studied. We found that population-based training algorithms not properly customized to the specific problem they are addressing fail to consistently outperform the null model on regression problems and do not exhibit remarkable performance on classification problems. Further research should focus on the various changes that can be made to these population-based algorithms that can be made to enhance their performance. Further, some existing parameters in our algorithms could be treated as hyperparameters to be tuned, rather than being set to a fixed value. While we believe that these methods would be likely to improve performance, it is important to note their increased computational complexity, and note the current algorithms' comparable performance to the backpropagation algorithm on classification datasets. While this further research into improved methods is certainly necessary, its necessity does not invalidate the efficacy demonstrated for these training algorithms in our research.

7 Appendix

Group work distribution

Paper — The paper was split up into sections for each person in the group. Nick wrote the abstract and introduction. Josh wrote the results and program design. Owen wrote the rest
Code — The code was written by Owen, Josh, and Nick.

Design Document — The design document was completed as a group with all members involved to ensure everyone understood the timeline and distribution of labor for the project.

Video Essay — The video essay was done by a combination of Owen and Josh