Exploring Performance Differences Between Genetic Algorithms and Back-Propagation on Feed-Forward Neural Networks

Braeden Hunt BRAEDEN.HUNT@STUDENT.MONTANA.EDU
Department of Computer Science
Montana State University
Bozeman, MT 59715, USA

Tyler Koon TYLER.KOON@STUDENT.MONTANA.EDU
Department of Computer Science
Montana State University
Bozeman, MT 59715, USA

Editor: Braeden Hunt and Tyler Koon

Abstract

The following study considers six experiments that quantitatively evaluate the performance differences between feed-forward neural networks trained using different learning algorithms in both classification and regression settings. The four algorithms under consideration are Back Propagation, Genetic, Differential Evolution, and Particle Swarm Optimization. The performance of these models is quantified using 0-1 Loss, F_1 Score, and Mean Squared Error metrics which have been computed using a stratified ten-fold cross-validation technique. Each of the experiments considers one of six independent and widely varied datasets, and used each algorithm to train networks of zero, one, and two hidden layers. The performance measurements from each experiment have been compared, offering interesting insight into the relative performance and behavior of these learning algorithms.

Keywords: Feed-Forward Neural Network, Back-Propagation, Genetic Algorithm, Differential Evolution, Particle Swarm Optimization, Regression, Classification

1. Introduction

In recent years, neural networks have become among the most popular learning techniques in the machine learning and data science communities. Perhaps the most critical step when implementing neural networks is that of selecting an appropriate learning algorithm. The chosen algorithm can have profound impacts on the speed of training and accuracy of results for any given problem. Our study attempts to contribute insight by exploring and documenting the relative performance of four different training algorithms across a broad range of research settings and neural network configurations, offering background context to aid future researchers in selecting an appropriate learning algorithm.

Specifically, our study considers the relative performance difference between four learning algorithms: Backpropagation (Backprop), Genetic Algorithm (GA), Differential Evolution (DE), and Particle Swarm Optimization (PSO). These learning algorithms were used to train neural networks with zero, one, and two hidden layers in both the classification

and regression settings across widely varied data contexts. To answer our research question, we quantified the performance of a model using three evaluation metrics: 0-1 Loss and F_1 Score for classification data, and Mean Squared Error for regression data. These metrics were computed for each model using stratified ten-fold cross-validation, offering average performance measures for each model. Initially, we hypothesized that the Particle Swarm Optimization and Backpropogation algorithms would perform similarly due to using descent-like optimization techniques. That said, we believed Particle Swarm Optimization would converge quicker as it considers *many* active agents distributed across the search space. Additionally, we predicted the Genetic and Differential Evolution algorithms would perform similarly as both implement similar searching methods that consider chromosome performance in small batches. We understood this to improve exploration but reduce exploitation of the underlying data, leading us to believe that both would perform worse than Backpropogation and Particle Swarm Optimization. Lastly, we predicted the three biologically-inspired algorithms would be more resilient to changes in layer count than that of Backpropogation due to the vanishing gradient problem. The results of our study suggested that these predictions were correct with the exception that PSO did not perform as well as Backpropagation.

The following paper is composed of four sections that describe our experimental design, implementation, and findings in detail. We begin in Section 2 with a comprehensive overview of our experimental design. This is accompanied by our findings in Section 3, which are discussed at length in Section 4. Finally, we conclude with a review of the study and proposal for future research in section 5.

2. Experimental Design

In this section, we detail our experimental approach to address the research question. We begin by describing the chosen datasets for the study (Section 2.1), followed by an overview of our software design (Section 2.2) and a review of our evaluation technique (Section 2.3). We then describe the underlying learning algorithms considered in the study (Section 2.4) and conclude with a comprehensive description of our experiments (Section 2.5).

2.1 Data Sets

Our study used six independent data sets sourced from the UCI Machine Learning repository to explore our hypothesis. Of these data sets, three contained classification data and three contained regression data. The three classification data sets were the Wisconsin Breast Cancer Database (January 8, 1991), Glass Identification Database, and Small Soybean Database. The three chosen regression data sets for this study were Abalone, Computer Hardware, and Forest Fires. Each of these data sets offered a unique combination of features, sample sizes, and response values, improving the breadth of our study by affording a more diverse set of experiments. The diversity of these six data sets offered an appropriate platform to evaluate our hypothesis with respect to many different contexts in both a classification and regression setting. Accompanying this variability in the composition of these data were a number of issues that needed to be addressed through data wrangling and preprocessing. This included the need to handle categorical features of varying complexity,

account for missing data, and drop irrelevant features. The details of these challenges and our solutions are discussed at length in Section 2.5.

2.2 Program Design

To implement our experiments, we developed software tools to facilitate the preprocessing, training, and evaluation of the underlying neural networks. These tools were broken into three categories: Algorithms, Utilities, and Evaluation. The Algorithms component contained our implementation of the feed-forward neural network and each of the three genetic algorithms. A generic 'Layer' class was used to interface between the neural network and the individual learning algorithms, using tools from the Utilities section to handle the conversion of layer data (weights) into chromosomes. This section also implemented various crossover, mutation, and selection algorithms through generic interfaces that afford the ability to quickly modify and expand the behavior of the underlying learning algorithms. This design decision proved useful for exploration during our experiments and offers the opportunity to easily expand the scope of this study to consider many variations of the chosen learning algorithms in future research.

The utilities component contains tools that support our experiment implementations. This includes a class for preprocessing the datasets which involves reading data from CSV files, dropping irrelevant and missing data, and applying modifications to features as required. Accompanying this preprocessing utility is a tuning utility that produces semi-optimal hyper-parameters for our learning algorithms. This utility relies on a standardized 'hyper-parameters' dictionary to interface with each of the learning algorithms, offering a convenient way to define the various hyper-parameters for each algorithm while maintaining a consistent class signature. We gave it a range, step size, and the default value for each hyper-parameter and it tuned them one at a time. Lastly, a utilities class exposes static methods for serializing network weights into chromosomes as well as deserializing chromosomes back into network weights.

Finally, our facilities for evaluating trained networks are contained in the Evaluation component of our program. This includes a k-fold cross-validation utility that is responsible for decomposing data into training and testing sets, breaking testing sets into k stratified folds, normalizing continuous data between 0 and 1, and computing model performance against each fold. Additionally, this utility implements our 'EvaluationMeasure' class which exposes methods for computing the precision, recall, 0-1 Loss, F_β score, and Mean Square Error for results from each fold. These two classes form the basis for our experimental design which is presented in Section 2.5.

2.3 Evaluation Methodology

In order to answer our research question and compare the relative performance between each of the chosen learning algorithms, we developed a standard means of quantifying model performance. For classification-based experiments, we quantified model performance through a composite of the 0-1 Loss and F_β Scores. These two metrics offered a general overview of the capabilities of the trained model, and by extension the training algorithm, by considering accuracy, precision, and recall. A $\beta = 1$ was chosen for the F_β score as it

equally weights the precision and recall, a trait which is desirable given our goal of comparing general model performance.

For regression data, we considered the performance of a model to be quantified by the Mean Squared Error between the predicted and expected output. This metric was chosen due to its weighted consideration of prediction error, offering polynomial increases in error for predictions that deviate further from the ground truth. This penalizes learning algorithms that perform marginally worse, making them easier to compare given our relatively limited number of experiments and datasets.

In general, these three metrics offer a strong representation of network performance while remaining computationally simple and easily comparable. That said, future research in the area might consider additional performance measures which provide a more targeted insight on specific model features. One such alteration might consider different values for β so as to focus on impacts to precision *or* recall as opposed to their aggregation.

2.4 Algorithms

Our study considered the relative performance differences of four optimization-based learning algorithms: Backpropagation, Genetic, Differential Evolution, and Particle Swarm Optimization. As we explored in our previous study, the Backpropagation algorithm propagates error through the network layers and optimizes the layer weights using stochastic gradient descent. Because the neural network structure and evaluation process in this previous study are identical to the process used in our current study, we used these existing backpropagation performance results as a baseline when comparing the remaining three evolutionary algorithms presented below. For these algorithms, we defined the fitness of a given chromosome as one minus its 0-1 Loss when serialized to a network and tested against the underlying data fold. This metric offers a simple, standardized value that represents holistic performance making it ideal for judging chromosome fitness.

The Genetic algorithm uses an iterative process to propagate augmented information between populations of candidate solutions, retaining the fittest individuals. The augmentation process occurs during each iteration and is the result of crossover and mutation events. Our implementation used the tournament selection method to determine which parents in a generation would perform this propagation of information based on fitness. We favored this method for its local competition which offered lower selection pressure and prevented any single parent from dominating the selection process. Additionally, we implemented the uniform crossover method for its improved genetic diversity and consequent lower selection pressure. This used a uniform distribution to swap corresponding gene information between two selected chromosomes. For mutation, we implemented a similar process where a uniform distribution was used to replace gene information with a random weight value constrained by the initial weight bounds. To propagate this augmented chromosome information to the next generation, we used the steady-state replacement method to replace all selected parent chromosomes with their augmented children. Once again, we chose this method for its lower selection pressure and computational simplicity.

The Differential Evolutionary algorithm functions very similarly to the Genetic algorithm: populations of candidate solutions are augmented through crossover and mutation processes, and the fittest solutions are propagated to future generations. The primary differ-

ences between the Differential Evolutionary and Genetic algorithms are that of representation and order of events. The Differential Evolution algorithm encodes genetic information as vectors rather than bit-strings, and the mutation process occurs before the crossover process. Our implementation used the random selection method to randomly select unique candidate chromosomes from the population. These chromosomes were then used to create a trial vector using differential mutation:

$$u_j = x_1 + \beta(x_2 - x_3)$$

Where β is a tunable scaling factor and x_1, x_2, x_3 correspond to the three selected candidate chromosomes. We decided to use the binomial crossover method to augment the trial vector, which offered lower exploitation and better exploration than contesting methods. This involved uniformly replacing the target chromosome's genes with corresponding genetic information from mutated chromosomes at random. Finally, we used the direct comparison of fitness for the target and trial vectors, inserting the champion back into the population.

The PSO implementation has a much simpler implementation. We start with a population of particle objects, each of which contains the hyper-parameter values for *inertia*, c_1, and c_2, its current position, its best position, and its best fitness. In each generation, every particle's velocity and position are updated by passing the global best position seen before the start of the generation according to this equation:

$$v_i(t+1) = \omega v_i(t) + c_1 r_1(pb_i(t) - x_i(t)) + c_2 r_2(gb(t) - x_i(t))$$

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

Where $v_i(t)$ and $x_i(t)$ is the velocity and position of particle i at time t. ω is the inertia, c_1 and c_2 are scale factors, r_1 and r_2 are random numbers in $U(0,1)$. $pb_i(t)$ and $gb(t)$ are the personal best position of particle i and the global best position of all particles at/by time t

2.5 Experiments

Our study considered six independent experiments, each of which implemented the learning algorithms discussed in Section 2.4. These algorithms were used to train three neural networks with zero, one, and two hidden layers for each of the six underlying datasets. The shape of each network as well as the performance results for the Backpropagation algorithm were taken from our previous study which implemented a similar evaluation process and considered the same folds. These experiments followed the same structure, consisting of three stages: Preprocessing, Tuning, and Evaluation.

The preprocessing stage handled the ingestion of the underlying dataset for each experiment. This included dropping irrelevant features, performing feature conversions, and dropping samples with missing data. The selected datasets for this study were well-structured and required little processing. For datasets that contained samples with missing feature data, we decided to simply drop the affected samples. There were so few samples missing feature data that the effects of this decision were considered negligible. Additionally, we dropped features used for identification that were deemed irrelevant to the relationships between samples.

After processing the underlying dataset in each experiment, we then tuned the three biologically-inspired algorithms implemented in this study (recall that the Backpropagation algorithm was tuned in our previous study). This tuning process was facilitated by the tuning utility described in Section 2.2 and involved performing a simple linear search through a range of hyper-parameter values. This search process used a step size to guide the scanning process, and the range of values considered for each parameter was chosen based on intuition and experimentation using 10-Fold cross-validation. Additionally, we selected the number of generations and individuals in each population to produce running times similar to that of Backpropagation for each dataset. This allowed us to compare the results on a more equal footing as training/tuning time is a significant factor to consider when deciding which learning algorithm to use. Due to the time-constrained nature of this project and a large number of tunable hyper-parameters in the underlying learning algorithms, we tuned parameter values independently from one another. Additionally, we used the same parameters across all network configurations for each algorithm in a given dataset. This decision likely impacted the results of our study, and future research should consider additional tuning methodologies. The implications of these decisions are discussed at length in Section 4.2.

The final stage in our experiment design was responsible for evaluating the performance of each model. These results were produced through a stratified 10-fold cross-validation process, and were used to compute the evaluation metrics described in Section 2.3: 0-1 Loss and F_1 Score for classification data, and Mean Square Error for regression data. These metrics were recorded for each model, and the results were compared between and within experiments.

3. Results

This section provides additional context for the underlying experiments and reports the results obtained in the study.

3.1 Breast Cancer Identification Results

The Wisconsin Breast Cancer Database (January 8, 1991) describes samples of breast tumors. Each sample has nine continuous features (encoded from 1-10) and a tenth classification feature representing whether or not the tumor is malignant or benign. For the F_β score, we considered malignant to be the "positive" class and benign to be the "negative" class. In total, 699 samples are described by the data set: 241 malignant, and 458 benign. Of these samples, 16 are missing a single attribute and zero are missing two or more attributes. Given the relatively small proportion, we decided to drop all instances missing any feature data. The results from the experiment are as follows:

0 Hidden Layers		
Algorithm	Testing Loss	Testing F_1 Score
Backprop	0.911	0.881
Genetic	0.696	0.164
Differential Evolution	0.653	0.599
Particle Swarm Optimization	0.851	0.735

Table 1: Network shape: [9, 2]

1 Hidden Layer		
Algorithm	Testing Loss	Testing F_1 Score
Backprop	0.984	0.978
Genetic	0.739	0.342
Differential Evolution	0.469	0.311
Particle Swarm Optimization	0.932	0.896

Table 2: Network shape: [9, 2, 2]

2 Hidden Layers		
Algorithm	Testing Loss	Testing F_1 Score
Backprop	0.985	0.980
Genetic	0.691	0.536
Differential Evolution	0.532	0.208
Particle Swarm Optimization	0.870	0.830

Table 3: Network shape: [9, 2, 2, 2]

3.2 Soybean Identification Results

The Small Soybean Database describes samples of soybeans through 35 nominalized categorical features. Each sample is classified into one of four output classes: D1 (10 instances), D2 (10 instances), D3 (10 instances), and D4 (17 instances). For our F_β computations, we considered the D1 class as "positive", and all other classification levels as "negative". In total the dataset describes 47 samples, none of which are missing feature data. The results from the experiment are as follows:

0 Hidden Layers		
Algorithm	Testing Loss	Testing F_1 Score
Backprop	1.000	1.000
Genetic	0.390	0.507
Differential Evolution	0.515	0.700
Particle Swarm Optimization	0.780	0.900

Table 4: Network shape: [35, 4]

1 Hidden Layer		
Algorithm	Testing Loss	Testing F_1 Score
Backprop	0.980	1.000
Genetic	0.375	0.233
Differential Evolution	0.265	0.367
Particle Swarm Optimization	0.420	0.217

Table 5: Network shape: [35, 35, 4]

2 Hidden Layers		
Algorithm	Testing Loss	Testing F_1 Score
Backprop	1.000	1.000
Genetic	0.390	0.250
Differential Evolution	0.240	0.173
Particle Swarm Optimization	0.510	0.267

Table 6: Network shape: [35, 35, 35, 4]

3.3 Glass Identification Results

The Glass Identification Database describes samples of glass fragments through nine continuous features (note, the 'ID' feature was dropped during our preprocessing stage due to irrelevancy) describing the physical properties of the fragment. Each sample is classified into one of seven classes (encoded as integers 1-7) representing the type of glass. For out F_β computation, we considered class 2 to be "positive", and the remaining classes to be "negative". In total, the dataset contains 214 samples, none of which are missing feature data. The results from the experiment are as follows:

0 Hidden Layers		
Algorithm	Testing Loss	Testing F_1 Score
Backprop	0.633	0.610
Genetic	0.214	0.161
Differential Evolution	0.114	0.026
Particle Swarm Optimization	0.403	0.433

Table 7: Network shape: [9, 7]

1 Hidden Layer		
Algorithm	Testing Loss	Testing F_1 Score
Backprop	0.797	0.788
Genetic	0.315	0.216
Differential Evolution	0.139	0.052
Particle Swarm Optimization	0.350	0.328

Table 8: Network shape: [9, 9, 7]

2 Hidden Layers		
Algorithm	Testing Loss	Testing F_1 Score
Backprop	0.782	0.803
Genetic	0.287	0.156
Differential Evolution	0.140	0.000
Particle Swarm Optimization	0.320	0.365

Table 9: Network shape: [9, 9, 9, 7]

3.4 Forest Fires

The Forest Fires data set describes 517 instances, none of which contain missing feature values. There are a total of 13 attributes, two of which are cyclical. The rest are continuous values that represent the conditions of the recorded fire. We attempt to calculate the logarithmic transformed burned area of the forest from these values: $\hat{y} = log(y + 1)$. This transformation helps address the skew towards an area of zero.

0 Hidden Layers	
Algorithm	Testing MSE
Backprop	1.957
Genetic	3.631
Differential Evolution	3.220
Particle Swarm Optimization	2.214

Table 10: Network shape: [12, 1]

1 Hidden Layer	
Algorithm	Testing MSE
Backprop	1.823
Genetic	3.496
Differential Evolution	3.635
Particle Swarm Optimization	2.058

Table 11: Network shape: [12, 6, 1]

2 Hidden Layers	
Algorithm	Testing MSE
Backprop	1.848
Genetic	3.504
Differential Evolution	3.603
Particle Swarm Optimization	2.054

Table 12: Network shape: [12, 3, 3, 1]

3.5 Computer Hardware

The Computer Hardware data set describes 209 instances, none of which are missing feature values. There are a total of ten attributes, two of which were dropped due to being identifiers. The rest are continuous values that describe computer hardware. We attempt to calculate the Estimated Relative Performance (ERP) from these values.

0 Hidden Layers	
Algorithm	Testing MSE
Backprop	838.223
Genetic	23756.190
Differential Evolution	23750.254
Particle Swarm Optimization	7780.216

Table 13: Network shape: [7, 1]

1 Hidden Layer	
Algorithm	Testing MSE
Backprop	6168.209
Genetic	23754.004
Differential Evolution	23756.797
Particle Swarm Optimization	15771.943

Table 14: Network shape: [7, 7, 1]

2 Hidden Layers	
Algorithm	Testing MSE
Backprop	659.142
Genetic	23750.764
Differential Evolution	23763.036
Particle Swarm Optimization	18492.528

Table 15: Network shape: [7, 7, 7, 1]

3.6 Abalone

The Abalone data set describes 4177 instances, none of which are missing feature values. There are a total of nine attributes. One is a nominal value ('sex') that we implemented one-hot-encoding for. The rest are continuous values about the size and weight of the abalones. We attempt to calculate the abalone rings (which correspond to the sample's age) from these values. The results from the experiment are as follows:

0 Hidden Layers	
Algorithm	Testing MSE
Backprop	8.100
Genetic	100.980
Differential Evolution	104.337
Particle Swarm Optimization	15.789

Table 16: Network shape: [8, 1]

1 Hidden Layer	
Algorithm	Testing MSE
Backprop	4.286
Genetic	100.980
Differential Evolution	104.786
Particle Swarm Optimization	14.225

Table 17: Network shape: [8, 8, 1]

2 Hidden Layers	
Algorithm	Testing MSE
Backprop	4.288
Genetic	96.136
Differential Evolution	105.683
Particle Swarm Optimization	12.968

Table 18: Network shape: [8, 8, 8, 1]

4. Exploration of Results

In this final section, we consider the results of our study and draw conclusions regarding the relative performance between the four chosen learning algorithms. Additionally, we address potential limitations of our implementation and consider future research on the subject. This section concludes with a brief summary of the study and our findings.

4.1 Discussion

We initially hypothesized that PSO and Backpropagation would perform similarly to each other, with both outperforming the Genetic and Differential Evolution algorithms. We also predicted that the Genetic and Differential Evolution algorithms would perform similarly to each other. Finally, we predicted that there would be little variation in the performance of the three evolutionary models across the three network depths for each problem. On the first point, we were incorrect. Backprop performed markedly better than PSO. However, we were correct on the second point, where Backprop and PSO performed significantly better across all networks and problems when compared to GA and DE. We believe this to be due to the ability of these two algorithms to use momentum and utilize a much more significant pull towards locally optimal solutions than random gene crossover and mutation. We were correct in our prediction that GA and DE performed similarly. Finally, we were correct in our prediction that the evolutionary algorithms did not vary much in performance due to the number of hidden layers in the networks.

4.2 Limitations

Although our results provide interesting insight into our research question, it should be noted that there are a number of limitations in our study that should be considered when interpreting these results. Perhaps most notably, our experiments only considered six datasets which represent a small subset of research domains. This limits the generalizability of our experiments to similar contexts, impacting the effectiveness of our results in assisting future research projects in the algorithm selection process. Additionally, the small sample size for some of the chosen datasets resulted in small cross-validation folds. This may have skewed our results to favor those algorithms that are more robust to smaller sample sizes,

and future research should consider methods for handling this imbalance in data such as imputation.

Another limitation of our study is that we used a primitive tuning process to optimize hyper-parameters for our chosen learning algorithms. This process considered parameters independently, and only searched a small space of candidate values that were manually defined. This limitation is largely the result of time constraints imposed on the study and has likely resulted in lower-than-expected performance results. That said, we consider the impact of this limitation to be negligible in the context of our study as we were primarily interested in the *relative* performance of the chosen models.

Additionally, our experiments used the same set of tuned hyper-parameter values when testing each network shape for a given algorithm. This likely skewed our results to favor those algorithms that were more robust to network shape. Tuning for each network shape would allow each algorithm to perform under its most optimal conditions for the given context, offering a more accurate representation of the algorithm's behavior. Once again, these limitations were the result of time constraints imposed on the project, and future research should take the time to implement more comprehensive tuning processes to address these problems.

Finally, our experiments did not show our algorithms at their absolute best, as we imposed training time limitation to match those times used for backpropagation in our previous study. If we increased this time across all algorithms, we may have found that the evolutionary algorithms eventually perform better at the cost of much more training time.

5. Summary

The process of choosing a learning algorithm when implementing neural networks is critical to developing efficient and useful models. The large variety of available algorithms often leads to lengthy and time-consuming selection processes that delay the goals of the underlying projects. Our research attempts to address this problem of algorithm selection by offering additional background resources on the relative performance and behavior of popular learning algorithms. In particular, this study considered the backpropagation, genetic, differential evolution, and particle swarm optimization algorithms in the context of six independent datasets that represent both the classification and regression setting. To answer our research question, we developed a set of experiments that computed the mean 0-1 Loss, F_1 Score, and Mean Square Error for neural networks with zero, one, and two hidden layers trained using each algorithm. These experiments implemented a tuning facility that found semi-optimal hyper-parameters which were then used in stratified 10-fold cross-validation to produce generalizable results. Our experiments found that backpropagation clearly converged to strong solutions faster than the evolutionary algorithms for networks of 0-2 hidden layers for our given problems.

References

We used the provided course materials.