5-1-2018

On the feasibility of using genetic algorithms to optimize the structure of small multilayer perceptrons

Nicholas Dinep-Schneider
Mississippi State University

On the feasibility of using genetic algorithms to optimize the structure of small multilayer perceptrons

By

Nicholas Dinep-Schneider

An Honors Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Science
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

May 2018

Name: Nicholas Dinep-Schneider

Date of Degree: May 7, 2018

Institution: Mississippi State University

Major Field: Computer Science

Undergraduate Advisor: Christopher Archibald

Title of Study: On the feasibility of using genetic algorithms to optimize the structure of small multilayer perceptrons

Pages in Study: 22

Candidate for Degree of Bachelor of Science

Artificial neural networks, which mimic the human brain's ability to learn from experiences, are increasingly being used to analyze complex datasets. However, proper structural configuration requires intuition, trial-and-error, and frequent human attention. This study investigates an automated alternative that uses a Darwinian evolutionary strategy to optimize the structure of a small-scale Modified National Institute of Standards and Technology (MNIST) image classification network. Using accuracy as a measure of fitness, it examines the effect of varying the amount each network learns prior to differential reproduction. The accuracy of optimized networks was significantly higher than random initial networks, being increased by up to 0.133% (1.435 standard deviations above initial mean accuracy). The use of evolutionary strategies in design holds promise for producing networks that are appreciably more accurate than randomly-generated networks without a large ongoing input of human attention.

ACKNOWLEDGEMENTS

I would not have been able to complete this honors thesis without the help of several exceptional people.

I would like to thank Dr. Christopher Archibald, my thesis advisor, for guiding me through this process and helping me turn my ideas into reality. I have been able to learn and accomplish more than I had hoped, and I am truly grateful.

I would also like to thank Dr. Cindy Bethel and Mrs. Becky Gardner, for agreeing to serve on my committee. While neither specialize in my area of research, both of them have gone out of their way to help me see this project through to the end.

Additional thanks to the staff of the Computer Science department and of the Shackouls Honors College, especially Dr. Joseph Crumpton and Dr. Seth Oppenheimer, for making it possible for me to take both of my thesis classes concurrently.

Finally, I would like to thank my father, Dr. John Schneider, for his advice on sample size and statistical analysis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION AND PROBLEM

1.1 Artificial Neural Networks

Artificial neural networks ("neural networks," "networks," or "ANNs") are computer algorithms that attempt to mimic, to a certain degree, the human brain's ability to learn from experiences. They are increasingly being used to analyze and understand complex datasets like medical symptoms, trends on social media, and human speech. Although they can never be guaranteed to be 100% accurate or optimal, well-designed neural networks can quickly, efficiently, and accurately outperform humans on large or complex tasks, and can even do surprisingly well in areas that have traditionally been dominated by humans such as identifying objects in images or playing Go [13].

The key phrase, however, is well-designed. A simple unidirectional, or "feedforward," neural network has four essential components:

1. A set of inputs.
2. A processing center ("hidden layer") with many artificial neurons ("neurons").
3. A set of output neurons.
4. Weighted unidirectional connections from inputs to neurons, and from neurons to outputs ("weights" or "connections").

If this design is insufficient for a given problem, more complex feedforward neural networks ("deep networks," "multilayer perceptrons," or "MLPs") can be constructed by chaining several hidden layers together, so that the output of one becomes the input of the next. In either case, training the network is then achieved by iteratively changing its weights to bring its predictions closer to 100% accuracy.

For a finished network to perform as desired at the end of its training period, each component must be properly configured before training even begins. While certain types of networks are known to work well in general for certain types of problems, many of the elements of configuration--batch size, epochs, dropout, layer size, etc.--can require intuition coupled with trial and error to maximize performance.

1.2 Paper Topic

The purpose of this study was to examine an automated alternative to the intuitive, human-designed approach. More specifically, the proposed process uses an evolutionary strategy to optimize the number of layers and neurons in a small-scale multilayer perceptron that identifies handwritten digits from images. The hypothesis for this study was that this process would have the ability to create networks that were more accurate than randomly-generated networks, while not requiring the amount of attention and input that a human-designed network would. If successful, it would suggest the feasibility of a "set and forget" approach to multilayer perceptron design, where the creator could simply set the problem up, then return a number of hours later to find a structurally optimized and fully-trained network.

1.3 Novelty of Concept

The approach taken here differs from all previous approaches. Most experiments concerning the evolutionary optimization of MLPs, such as that of Vlahogianni and his colleagues [12], have focused on nonstructural elements like the network's weights (as a replacement for the most commonly used training algorithm, backpropagation), the training algorithm details, or the input format. Those that do explore structural optimization, e.g. Rocha and his colleagues [8], do so by changing the connectivity of a single-layer MLP. This appears to be the first attempt to efficiently optimize the number of layers in a MLP and the number of neurons for each layer, and it does so without limiting the number of layers, neurons, and/or connections.

1.4 Overview

Chapter 2 will cover major limitations of neural networks, provide a brief summary of their history and development, and give a high-level explanation of their functionality. Chapter 3 will describe the methods used for this study, including hardware and software, code design and philosophy, and statistical analysis procedures. Finally, Chapter 4 will summarize, analyze, and discuss the results of the study.

CHAPTER II

BACKGROUND

2.1 Neural Network Limitations

It is first important to understand that, however magical ANNs may appear, they are limited in several ways. As mentioned above, they can never reach a human level of accuracy on many tasks that humans perform with ease. Since they operate by finding a relatively simple function to approximate a complex problem, they are fundamentally incapable of giving the correct/best output for every input; the more varied a range of possible inputs, the less accurate an ANN can be.

In addition, current ANNs cannot reason--that is, unlike humans, they cannot learn a fact, remember it, prioritize and analyze it, and combine it with other facts to find solutions. This means that, for any kind of input on which they have not explicitly been trained, they cannot do better on average than an algorithm that guesses randomly.

2.2 History

Neural networks are a branch of machine learning that originated from an effort to reproduce human intelligence from the bottom up by simulating neurons. There have been three major waves of interest and research in the field, each corresponding to a new concept or technology.

The first wave began in the 1940s, fueled both by new theories about how learning worked [6] [3] and by the creation of some of the first learning models [9], although the latter were only able to simulate single "perceptrons," or neurons. Of the three waves, this was the one most driven by the hope that one could use artificial neural networks to understand the brain and/or use knowledge of the brain to improve neural networks. Unfortunately, neural networks were not advanced enough for the former, while too little was known about cognition for the latter, and this remains the case today. As a result of these limitations and the demonstration that networks using linear algorithms are unable to learn nonlinear problems [7], interest in neural networks began to subside.

The second wave began in the 1980s. Investigations were made into the idea that systems with many simple, interconnected, multitasking parts could produce complex behaviors [11], and more powerful computing technology was developed that could simulate these new and exciting systems. In addition, the rectified linear unit [1] and backpropagation [10] [5 as cited in 2] algorithms for training were developed, both still used today and discussed in more detail below. These four advances meant that neural networks could now truly be networks of more than a few neurons, and much progress was made. But, by the mid-1990s, the number of startup companies overpromising and underdelivering led to another loss of interest in neural networks.

The third wave, which is currently still rising, began in 2006 with the advent of networks and algorithms that are much less computationally expensive than those used previously [4]. These algorithms, combined with continued advances in technology that enable even older, slower algorithms to perform acceptably, have led to a proliferation of

research and discoveries in the field of machine learning. Deep neural networks in particular have flourished, due to their ability to outperform not only other kinds of machine learning, but many human-designed solutions as well. [1]

2.3 Neural Network Design

Neural networks can be visualized as a set of interconnected neurons that perform several different functions as each piece of data is processed. Input neurons merely rebroadcast received data, while processing neurons, output neurons, and connections modify and/or selectively transmit data. While some networks feed the output of the network back to input neurons, effectively simulating a simple memory and creating a network capable of learning trends in data (a "recurrent network"), problems such as the one examined in this study are better suited to a feedforward network. Since there is no connection between successive images of digits, any data from a previous image would only confuse the issue and lessen the accuracy of the network as a whole.

Supervised learning, the training method used for this study, requires each piece of training data to contain both a set of numeric inputs and a corresponding set of target outputs. During the training process, the network alternates between two modes:

1. The first mode, forward propagation, where the network uses the set of inputs to produce a set of what it "thinks" the outputs should be. [1]
    a. Each input neuron receives one numeric value, corresponding to one variable of the original problem--the value of one pixel, for example.

b. These values are transmitted along connections to the processing neurons and transformed en route by each connection's trainable scalar weight. The number of processing neurons each input connects to can vary from one (in a sparse hidden layer) to all (in a dense hidden layer).

c. Each processing neuron calculates the sum of its inputs, which is then transformed using an activation function that mimics the firing behavior of a biological neuron to some degree. This could be a simple binary or linear function, but most modern networks use more powerful nonlinear functions such as *output = max {0, input}*. This particular function is called the rectified linear unit, or "ReLU," and is the function used for this study.

d. If the network has more than one hidden layer, steps b and c are repeated for each additional layer, with the inputs now coming from the previous layer of processing neurons instead of from the input neurons.

e. Finally, the output neurons each sum their inputs and then apply an activation function, which may be different from that of previous layers. Each neuron may apply the function individually, or it may be applied across all the outputs. For example, a classification problem might set the output corresponding to the most likely class to 1, and set all others to 0 ("softmax").

2. The second mode, back-propagation, where the network is trained based on the accuracy of the results of the first mode. [1]
    a. The back-propagation algorithm is used to compute the gradient for each weight and parameter in the network--that is, to compute a measure of the effect of each trainable value on the final result. The algorithm recursively applies the calculus chain rule across each junction in the network, beginning at the outputs and moving back along to the inputs, to obtain a map of the effect of each value.
    b. A separate algorithm, for example stochastic gradient descent, is then used to modify each value based on the effect it had on the final outputs, and how close those outputs were to the expected outputs included in the training data.

This back-and-forth between the two modes means that, every time the network makes a guess, it receives immediate feedback and fine-tuning. Although this makes supervised learning more computationally expensive than the other common training methods, unsupervised and reinforcement learning, it provides more immediate, guided results than either of the others, and remains a viable option.

Finally, after training concludes, the network is tested on a separate set of data drawn from the same distribution, staying in the first mode for each piece of data. It is important to test on data not previously used for training in order to accurately gauge the network's final accuracy during actual use with novel inputs.

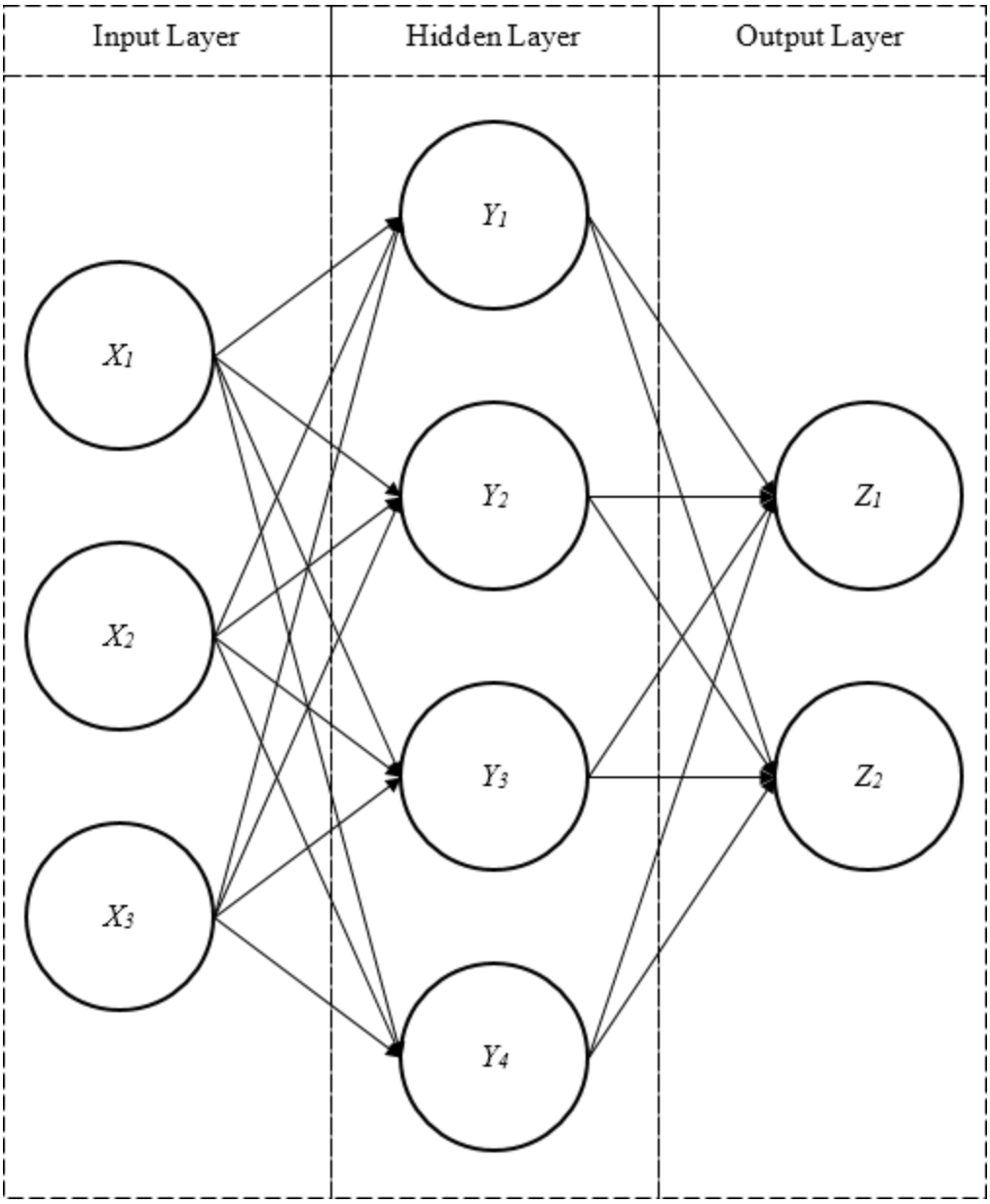

Figure 1 Multilayer Perceptron

This is a visual example of a simple dense feedforward multilayer perceptron with three input nodes $\{X_1, X_2, X_3\}$, one hidden layer with four neurons $\{Y_1, Y_2, Y_3, Y_4\}$, and two output nodes $\{Z_1, Z_2\}$.

CHAPTER III

METHODS

3.1 Hardware and Software

This project was coded and run on an Asus ROG G751JY-VS71(WX) laptop with an Intel Core i7-4720HQ CPU with 16 GB RAM and a NVIDIA GeForce GTX 980M GPU with 4 GB RAM, running Windows 10 Home version 1709. The code was written in Python 3.5 using the JetBrains PyCharm Community Edition 2017.3.3 IDE, and the neural networks were created and trained using the Keras 2.1.5 PyCharm library with the Tensorflow GPU 1.6.0 PyCharm library as a backend to accelerate training.

3.2 Code Design

The networks used for this study were small deep feedforward multilayer perceptrons, as described above. The data used for training was the Modified National Institute of Standards and Technology ("MNIST") database, consisting of 70,000 28x28-pixel greyscale images of single handwritten digits. 60,000 images were used for training, and the other 10,000 were reserved for testing. Each network had 784 inputs (28 * 28 individual pixel values per image), a variable number of hidden layers and neurons (each neuron using the ReLU activation function), and 10 output neurons (all using the softmax activation function, with each neuron corresponding to a different digit).

Elements of Darwinian natural selection, including differential reproduction and mutation, were incorporated into the model. For each trial ("population"), an initial group ("generation") of 18 networks ("individuals") was generated with 1-6 hidden layers and 50-200 neurons per layer. Each individual in the generation was trained for a fixed number of training runs over the entire training image set ("epochs"), then tested and ranked by final accuracy on the testing image set ("fitness"). The next generation was then created using the following algorithm:

1. Add the structural information ("genomes")--number of layers, number of neurons per layer--of the two fittest individuals in generation G_n to generation G_{n+1}, unaltered.
2. Repeat the following 16 times, to total 18 individuals in G_{n+1}:
    a. Simulate reproduction:
        i. Randomly select one individual from the fittest third of G_n.
        ii. Select a contiguous block of layers ("chromosomes") from the beginning of the individual's genome, including the first layer and up to but not including the output layer.
        iii. Randomly select another individual from the top third of individuals.
        iv. Select a contiguous block of layers from the end of the individual's genome, including the output layer and up to but not including the first layer.
        v. Splice the two parts together, creating the genome for a new individual.

b. Simulate mutation:

i. With a 1/10 chance for each non-output layer, add or remove a random number of neurons between 1 and 20 ("point mutation"). If the best fitness of G_n is less than that of G_{n-1}, the chance rises to 1/8. If in addition the best fitness of G_{n-1} is less than that of G_{n-2}, the chance rises to 1/6.

ii. With a 1/12 chance for each individual add or remove a new layer with 50-200 neurons, leaving the first and output layers unchanged ("chromosomal mutation"). If the best fitness of G_n is less than that of G_{n-1}, the chance rises to 1/8. If in addition the best fitness of G_{n-1} is less than that of G_{n-2}, the chance rises to 1/4.

c. Add the new individual to the next generation.

The best fitness, the fittest individual's genome, and the average fitness were archived for each generation. After 20 generations, the saved genomes for the initial, final, and overall-best-fitness generations ("population test individuals" or "PTIs") were each trained for 20 epochs, to assess how well the initial and final products of the algorithm would perform in a more real-life situation.

The numbers of generations per population, individuals per generation, initial hidden layers per individual, and initial neurons per layer were chosen in an attempt to maximize the number of trials that could be run--larger initial values tended to result in a prohibitive increase in the amount of time needed to run each trial.

The percentage of individuals considered reproductively fit, as well as the mutation rates in 2.b.i and 2.b.ii, were chosen based on initial results during the algorithm design process. While obviously not optimal, the former appeared to pass on fit genomes while rejecting undesirable ones. Similarly, the latter appeared to introduce a necessary amount of genetic variation without creating too much random fluctuation, while at the same time giving the next generation a "kick" of increased mutation if performance was decreasing.

The dependent variable for this study was the amount of training each individual in a given population received before assessment of fitness: 1 epoch, 2 epochs, 3 epochs, or 4 epochs. Data was gathered by simulating 12 initially random populations for each case. This variable was chosen because of its relationship to the amount of time taken by the algorithm--a parallel real-world study might examine the characteristics of a crop at 1, 2, 3, and 4 weeks of growth to determine which age provided the best early predictor of a plant's health at 20 weeks.

3.3 Statistical Analysis

The average accuracies of the first and last, and first and best, PTIs were compared for each case to determine whether there had been a significant improvement in accuracy from the first generation (i.e., whether the algorithm could indeed produce networks more accurate than random generation could). Due to the small (< 30) sample size, the paired t-test was used to assess significance at a 95% level of statistical confidence, and Cohen's *d* was used to determine how impactful the effect was.

CHAPTER IV

RESULTS AND DISCUSSION

4.1 Results and Analysis

Taking the mean of all 12 instances of each generation for every case, the best accuracy (Table 1, Figure 1) and average accuracy (Table 2, Figure 2) were increased by the genetic algorithm for all four cases, as shown below.

The important values, however, are those in Table 3, which shows the mean of all 12 instances of each PTI for every case. Although all four mean final-generation ("last") PTI accuracies were greater than their corresponding initial-generation ("first") PTIs, the 3- and 4-epoch cases had a much larger difference between first and last PTIs than the 1- and 2-epoch cases did (Figure 3), suggesting a larger chance of improvement even before analysis. The overall-best-accuracy ("best") PTIs did not present any obviously consistent behavior, presumably due to their inconsistent distribution across generations (Figure 4).

The following hypotheses were used to test the eight firstPTI-bestPTI and firstPTI-lastPTI differences for statistical significance:

Null hypothesis H_0: $\mu_{bestPTI} \leq \mu_{firstPTI}$ and alternative hypothesis H_1: $\mu_{bestPTI} > \mu_{firstPTI}$, or null hypothesis H_0: $\mu_{lastPTI} \leq \mu_{firstPTI}$ and alternative hypothesis H_1: $\mu_{lastPTI} > \mu_{firstPTI}$.

Generation	1-Epoch	2-Epoch	3-Epoch	4-Epoch
1	0.9487	0.9648	0.9712	0.9747
2	0.9501	0.9657	0.9722	0.9760
3	0.9509	0.9665	0.9729	0.9755
4	0.9517	0.9669	0.9730	0.9767
5	0.9527	0.9671	0.9738	0.9764
6	0.9532	0.9676	0.9734	0.9761
7	0.9529	0.9681	0.9735	0.9765
8	0.9527	0.9672	0.9742	0.9763
9	0.9524	0.9680	0.9741	0.9765
10	0.9531	0.9683	0.9735	0.9767
11	0.9532	0.9679	0.9734	0.9764
12	0.9537	0.9683	0.9732	0.9766
13	0.9535	0.9683	0.9738	0.9768
14	0.9542	0.9676	0.9740	0.9764
15	0.9539	0.9686	0.9734	0.9763
16	0.9540	0.9684	0.9736	0.9766
17	0.9541	0.9677	0.9738	0.9766
18	0.9541	0.9680	0.9735	0.9765
19	0.9537	0.9683	0.9739	0.9770
20	0.9542	0.9677	0.9739	0.9769

Table 1 Mean Best Accuracy Per Generation

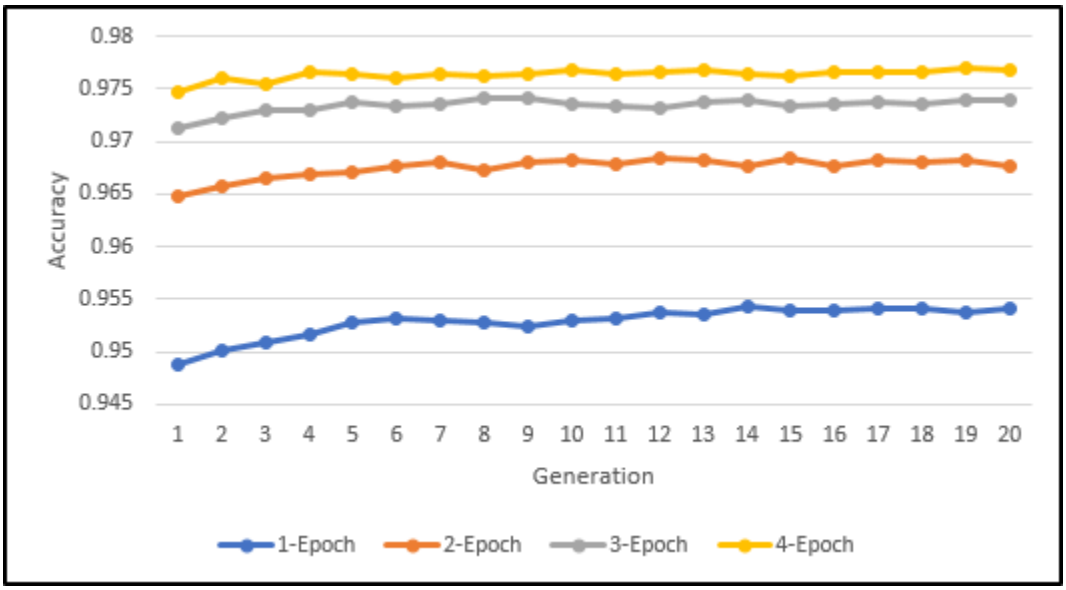

Figure 1 Mean Best Accuracy Per Generation

Generation	1-Epoch	2-Epoch	3-Epoch	4-Epoch
1	0.9374	0.9555	0.9629	0.9676
2	0.9403	0.9595	0.9676	0.9715
3	0.9432	0.9610	0.9684	0.9725
4	0.9441	0.9620	0.9688	0.9730
5	0.9449	0.9624	0.9695	0.9736
6	0.9455	0.9633	0.9697	0.9732
7	0.9458	0.9628	0.9697	0.9735
8	0.9460	0.9629	0.9703	0.9736
9	0.9456	0.9633	0.9705	0.9735
10	0.9455	0.9632	0.9701	0.9735
11	0.9454	0.9636	0.9701	0.9737
12	0.9462	0.9639	0.9698	0.9737
13	0.9469	0.9637	0.9702	0.9736
14	0.9467	0.9633	0.9702	0.9738
15	0.9467	0.9637	0.9700	0.9735
16	0.9467	0.9634	0.9704	0.9738
17	0.9467	0.9635	0.9702	0.9735
18	0.9461	0.9636	0.9702	0.9738
19	0.9461	0.9635	0.9703	0.9738
20	0.9465	0.9634	0.9706	0.9738

Table 2 Mean Average Accuracy Per Generation

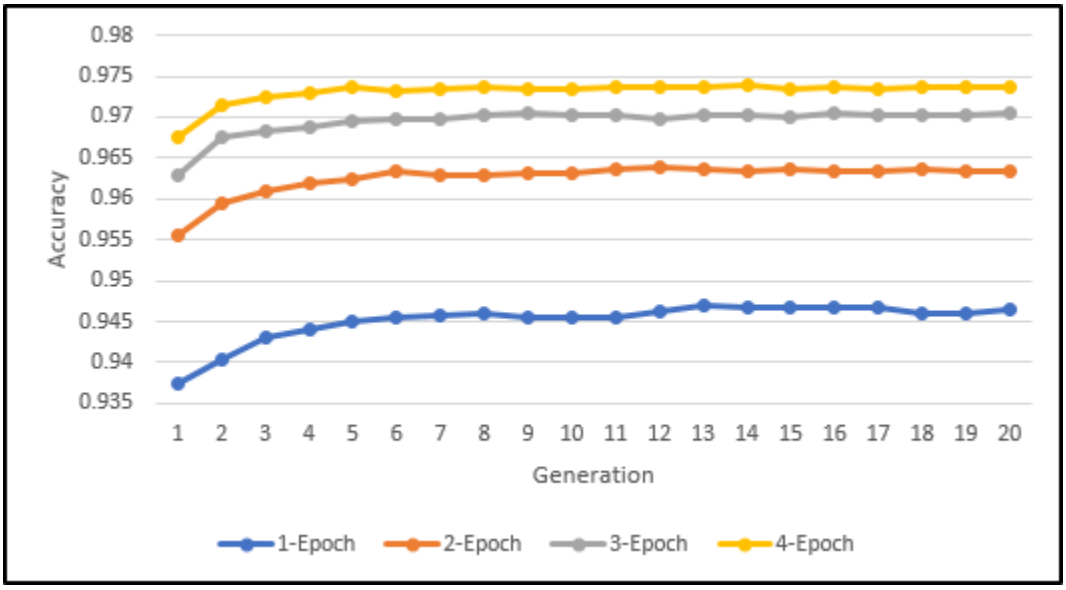

Figure 2 Mean Average Accuracy Per Generation

Trial Group	PTI	Mean Final Trained Accuracy	Mean Standard Error
1-Epoch	First	0.9814	0.000383
	Best	0.9821	0.000338
	Last	0.9817	0.000471
2-Epoch	First	0.9815	0.000527
	Best	0.9818	0.000280
	Last	0.9821	0.000255
3-Epoch	First	0.9815	0.000381
	Best	0.9819	0.000209
	Last	0.9827	0.000228
4-Epoch	First	0.9813	0.000314
	Best	0.9822	0.000200
	Last	0.9826	0.000213

Table 3 Mean Final Trained Accuracies

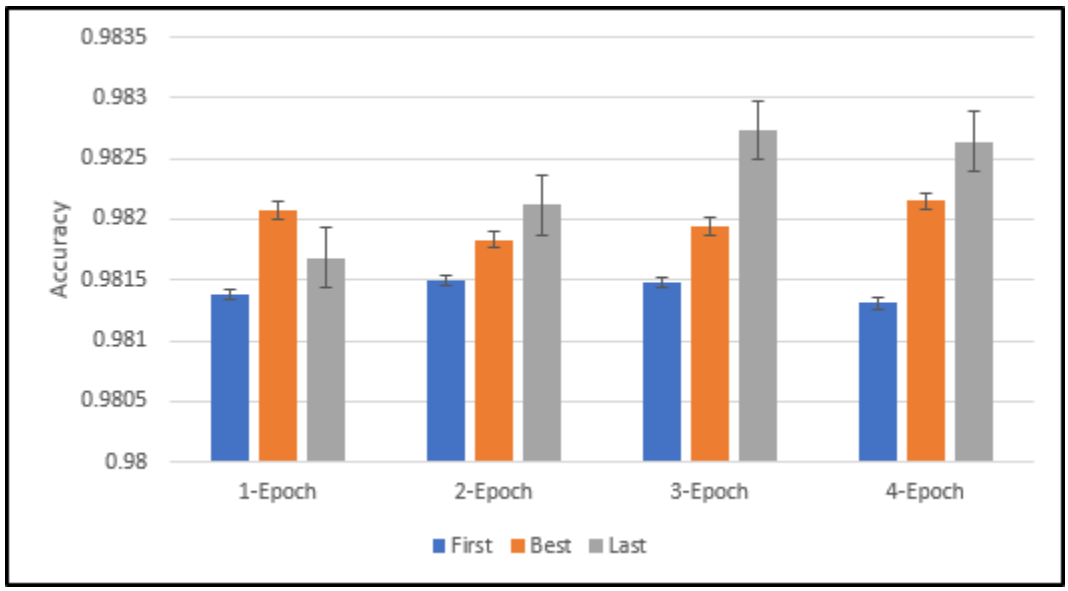

Figure 3 Mean Final Trained Accuracies

As shown in Table 4 below, there were five PTIs with a significant increase in accuracy: the 1-epoch best PTI, and 3- and 4-epoch best and last PTIs. All five increases had a Cohen's *d* above 0.2 and were therefore nontrivial; the 3-epoch best increase was small, the 1-epoch best was medium, and the three other increases were large.

Trial Group	Comparison	T-Statistic	Indicated Hypothesis at $t_{0.05} = 1.796$	Cohen's *d*
1-Epoch	First-Best	t = 1.962	H_1	d = 0.553
	First-Last	t = 0.499	H_0	--
2-Epoch	First-Best	t = 0.584	H_0	--
	First-Last	t = 0.932	H_0	--
3-Epoch	First-Best	t = 1.902	H_1	d = 0.438
	First-Last	t = 2.505	H_1	d = 1.156
4-Epoch	First-Best	t = 1.965	H_1	d = 0.922
	First-Last	t = 3.843	H_1	d = 1.435

Table 4 Paired T-Test and Cohen's d Significance Results

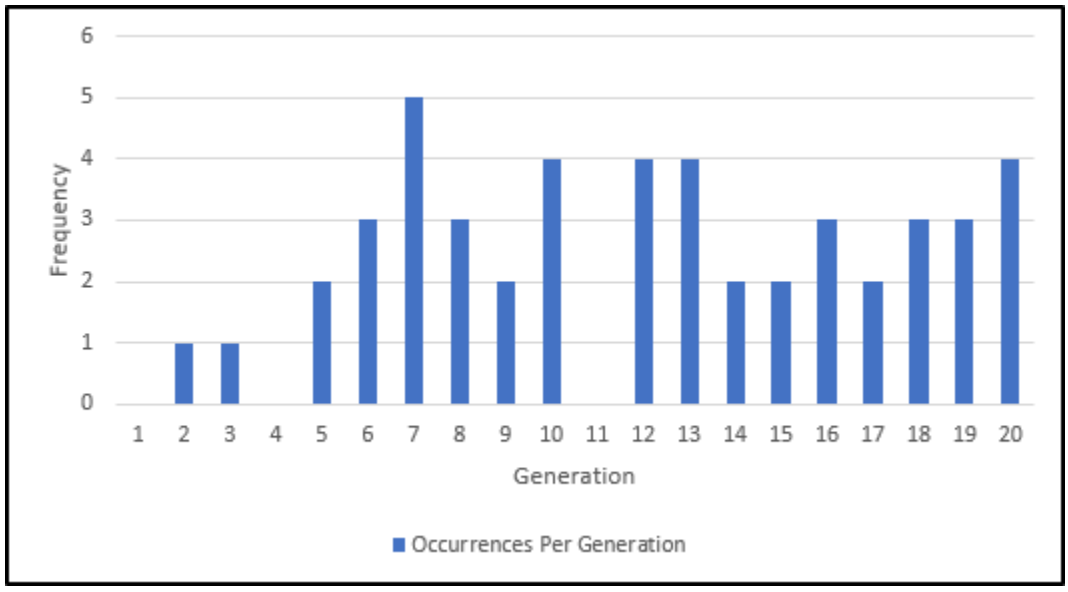

Figure 4 Best PTI Frequency Per Generation

4.2 Discussion and Conclusions

The results of this study confirm the hypothesis that, by assessing, reproducing, and mutating generations of networks--in other words, using the simplest of evolutionary principles-- the structures of small deep feedforward MLPs can be improved.

Even with the simple approach used, the algorithm was able to increase the average accuracy of the last PTIs by up to 0.00133, or 0.133%. Although this may not seem like a useful improvement in absolute terms, a network of this type with a processing throughput of 500,000 images a minute would make 665 fewer errors every hour.

In addition, these results present several interesting and nontrivial possibilities for continued exploration, such as:

- Due to hardware speed limitations, it was impractical to test cases beyond 4 epochs; more epochs might or might not produce even higher accuracies.
- One approach considered during the design of this project was to have networks pass on their weights in addition to their structures during reproduction, mimicking Lamarckian evolution; the effects of this on accuracy are unknown, but could be informative.
- Many of the values used, as mentioned above, were partially intuited and are presumably non-optimal. Adjusting them could greatly improve the performance of the algorithm.

This study, admittedly, had several limitations. The scarcity of published research on related topics made it difficult to confirm that no previous work was being duplicated. The hardware available for the project, although consistent with the intended small scale

of the project, was still a limitation in that it substantially limited the number of individuals, epochs, layers, and neurons which could reasonably be tested. In addition, the relatively small number of tests performed for each case meant that significance of results was more difficult to prove.

However, despite these and other difficulties, this study resulted in a significant advance in small-scale neural network design philosophy. It is important to recognize that this study's main accomplishment is a proof of principle. It answers the question of whether structural genetic optimization *can* be done, not how it can be done best or whether it should be done at all. This approach does indeed create networks that are more accurate than randomly-generated networks without requiring as much human input as current approaches, and it opens the door to potential streamlining of neural network creation.

REFERENCES

[1] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, 1980, pp. 193-202.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016. http://www.deeplearningbook.org.

[3] D. Hebb, *The Organization of Behavior*, Wiley, 1949.

[4] G.E. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, 2006, pp. 1527-1554.

[5] Y. LeCun, *Modèles connexionistes de l'apprentissage*, Ph.D. thesis, Université de Paris VI, 1987.

[6] W.S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 115, no. 5, 1943, pp. 115-133.

[7] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, 1969.

[8] M. Rocha, P. Cortez, and J. Neves, "Evolution of neural networks for classification and regression," *Neurocomputing*, vol. 70, no. 16-18, 2007, pp. 2809-2816.

[9] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, 1958, pp. 386-408.

[10] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning representations by back-propagating errors," Nature, vol. 323, 1986, pp. 533-536.

[11] D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, 1986.

[12] E.I. Vlahogianni, M.G. Karlaftis, and J.C. Golias, “Optimized and meta-optimized neural networks for short-term traffic flow prediction: A genetic approach,” *Transportation Research Part C: Emerging Technologies*, vol. 13, no. 3, 2005, pp. 211-234.

[13] M. Watson, “Computer Learns To Play Go At Superhuman Levels 'Without Human Knowledge',” blog, 18 Oct. 2017; https://www.npr.org/sections/thetwo-way/2017/10/18/558519095/computer-learns-to-play-go-at-superhuman-levels-without-human-knowledge