Analysis of Genetic Algorithms Optimizing Topological Layout and Synaptic Weights

Taras Mychaskiw (4105797) <tm07qx@brocku.ca>
Evan Verworn (4582938) <ev09qz@brocku.ca>

***Abstract*—The purpose of this experiment was to compare the results of training artificial neural networks through standard backpropation and using a genetic algorithm. The genetic algorithm altered both the structure and weights of the network to attempt to encourage learning.**

I. Introduction

THIS restate abstract and state results a;sodghf;laksjdhfnpva;shdflnoakesjdhxbf kcjzsdujv jvkjzdsxhhjfvf

II. Background

A. Neural Networks

A high level description of a basic Neural Network is a single directional graph without cycles or reflexive edges. It is a mathematical model wherein given some number of inputs and some number of outputs, the outputs will react to the magnitude of the input values. A visualization is given in Figure 1.

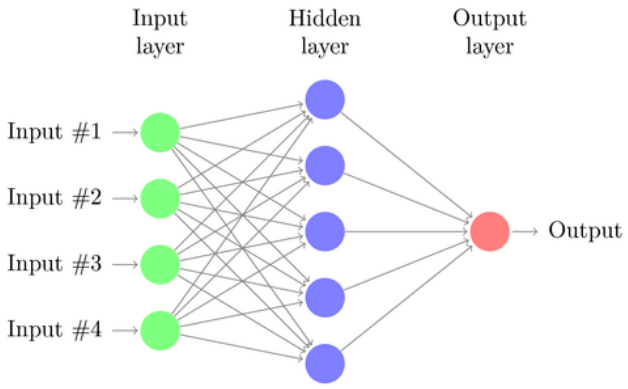

Fig. 1. An example single hidden layer neural network.

The output is *trained* to the desired output through manipulating the weights in the intermediate (hidden) and output layer edges.

A node calculates its output value by summing the output of all of its predecessors and multiplying that output by a weight assigned to that edge, this value is then squashed to a number traditionally between 0 and 1 by a Sigmoid function.

In the Figure 1 there is only one hidden layer, but in our experiments we are evolving a network that can have up to three hidden layers.

$$node\_output = \sum_{i=1}^{n} w_i x_i$$

Fig. 2. Where n is the number of edges coming *into* this node, w is the weight associated with that edge and x is the output value produced by the predecessor node.

Learning: A neural network learns by examples. Training examples are fed into the network, and the network changes the synapse weights so that when training is complete, the network will produce the required results when fed new problems it has not seen during training. The learning process is called *Back-Propagation*. [4]

Gradient Descent: Gradient descent is a common first order optimization algorithm. The is the reason we use a sigmoid activation function instead of the Heaviside function; the gradient of Heaviside is undefined at $x = 0$. Gradient descent has it's basis in vector calculus. The gradient of a multi-variable function points in the direction of fastest growth of that function. The negative gradient then points in the direction where the function decreases the fastest. The function, in the case for neural networks, is the error of the system, $f(\omega_1, .., \omega_n)$, a function of all the weights of the network. The basic gradient descent weight update rule is

$$\vec{\omega} \leftarrow \vec{\omega} - \nu\vec{\nabla}\xi(\omega) \tag{1}$$

where $\vec{\omega}$ is the synapse weights, ν is the learning rate, and $\xi(\omega)$ is the error function. The error function value can be determined during the *Back-Propagation* step, then one component of $\vec{\omega}$ is updated using Equation 2, where $g(\omega_i)$ is just the activation function of the neuron. [5]

$$\omega_i \leftarrow \omega_i - \nu\frac{\partial}{\partial\omega_i}g(\omega_i) \tag{2}$$

Back-Propagation: A network is first initialized by setting all of the synapse weights to small random numbers, usually between -1 and 1. A training example is fed into the network, and the output is calculated. Since each weight is a random number, the network output is completely different from the target output. The strategy is to calculate the error produced by each neuron. At the output layer, the error is simply $TargetOutput - ActualOutput$. According to the gradient descent equations, the error at the output layer is

$$err = \frac{\partial}{\partial\omega}g(x) \times (TargetOutput - ActualOutput) \tag{3}$$

where x is the value of the weighted sum.
Propagating the error back, the error at a hidden neuron j is

$$err_j = \frac{\partial}{\partial \omega} g(x) \times \left(\sum_i \omega_i \cdot err_i \right) \tag{4}$$

where i is a neuron in the next layer.
Finally, with all the error function values found, the weights of all the synapses in the network are updated using Equation 1. [3]

B. Genetic Algorithms

Genetic Algorithms or GAs are another mathematical model for finding an optimized solution in a large search space. This model is based off of Darwinian Evolution in that the best performing current solutions are bred together mixing genetic information from both parents into their children (known as a crossover operation). These children are then evaluated, just as their parents were, and subjugated to the same breeding rules.

Like in biological evolution corruption of the genetic code can happen, this is a possibly destructive mutation that encourages diversity between parents. This mutation can provide new genetic code to the child that it might benefit from that it couldn't have received from the parents.

The difficulty of this search method is programically defining the layout of the genetic code (a Chromosome) and creating different mutation and crossover functions that hopefully it can benefit from.

III. Methodology

This goal of this experiment is to use a GA to optimize a neural network and compare the result to that of a vanilla neural network trained by back propagation. The following are the graph operations that were chosen as the crossover and mutation functions.

A. Mutations

Mutations are possibly destructive operations that encourage diversity and explore the search space. All of these mutations where weighted the same and had the same chance of being used in all of the experiments.

1) Add Node: If the current graph allows for more nodes add one in the first possible hidden layer and connect it to any proceeding nodes behind it. For each new output edge from this node, randomize the weight associated with it. So while the node might be 'fully connected' to all proceeding nodes, as the weight approaches 0 that edge effectively becomes disconnected.

This function can only add nodes to the hidden layers of the neural networks. The input and output layers have a fixed amount of nodes that must exist, but can be disconnected.

2) Remove Node: This randomly selects a non-output node with connections and removes all outgoing connections. This function can disconnect input layer nodes. This could be considered beneficial to the network as 'feature selection' and could remove data that could potentially add noise to the input layer, and then get propagated down the line.

3) Add Edge: This method selects a random edge and changes its value.

If the edge does not exist it is created with a uniformly distributed 'weight of connectivity' ranging from [0...1). If the edge does exist, a new value within the same range is given and the weight is overridden.

4) Remove Edge: Randomly selects a connected edge to remove. There is no restriction in what layer this can happen.

B. Crossovers

Crossovers build off of existing solutions and exploit the genetic code we have found thus far to be useful.

1) Union: A union of all nodes and edges of the two graphs, if two edges exist on the two graphs then weights of both edges are averaged and this value becomes the new weight in the child.

This is a crossover that can easily create bloat in the child that doesn't help it in any way. This child now has the superior genetic code of both parents with all of the unhelpful (malignant) mutations from both. [1]

2) Intersection: The intersection function is a 'clean-up' function, but can be very destructive if there is too much diversity between two parents. Just like in biological evolution if there is too much difference between the parents (different species) the child may not be as functional as each of the parents.

When this type of crossover is selected, all edges and nodes that both parents share get passed onto the child. All other nodes/edges that are not shared by **both** parents is discarded. [1]

This function helps to reduce bloat that is created from the different mutations.

3) Roulette Union: An alteration to the standard union crossover. Instead of merging both parents into one child, a coin is flipped which decides which parent to take an edge from. If heads, then the edge from Parent A is copied exactly, else the edge from Parent B is copied.

If both parents have a connection from node $A \rightarrow B$ then there is a 100% chance that the edge $A \rightarrow B$ exists. If one parent has a connection $A \rightarrow B$ and the other doesn't, then there is a 50/50 chance of the child receiving the link or not, compared to inheriting all edges like the previous mentioned union.

This is done for all aspects of both parents, if both parents have the same aspect it will for sure show up in the child. This is more how biological evolution works wherein the Chromosome is built from a random selection of a little of Parent A and a little of Parent B.

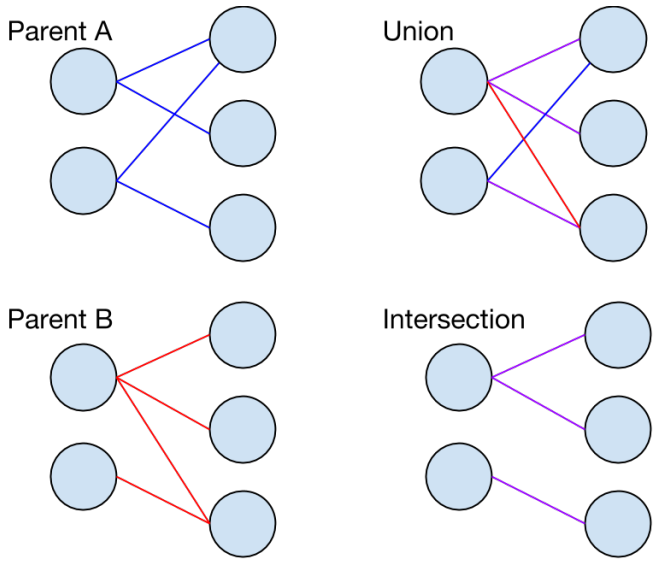

Fig. 3. Example of union and intersection crossovers.

IV. Experiments

The purpose of this project is to compare results of vanilla backpropation to the results of a Genetic Algorithm applied to optimizing a neural network's weights and topography.

Traditionally a comparison like this only focuses on the GA optimizing the weights **or** the topography of the neural net. We have attempted to do both simultaneously. To compare fairly choose two problems that would be challenging.

The first problem is in need of feature reduction with it's many inputs and few possible outputs.

The second problem is a noisy data problem with many key variables it must compare for the final result.

A. Connect 4

This dataset contains all legal 8-ply positions in the game of connect-4 in which neither player has won yet, and in which the next move is not forced.

The input is the full state of the board (who is in each position) and the expected output is either 'win', 'loss' or 'draw' for the 'first player'.

This experiment's description could be simplified to creating a neural network as the heuristic function of a connect-4 board.

B. Quality of Wines

Two datasets were created, using red and white wine samples. The inputs include objective tests (e.g. PH values) and the output is based on sensory data (median of at least 3 evaluations made by wine experts). Each expert graded the wine quality between 0 (very bad) and 10 (very excellent).

These datasets were then merged together. The input layer contains "fixed acidity", "volatile acidity", "citric acid", "residual sugar", "chlorides", "free sulphur dioxide", "total sulphur dioxide", "density", "pH", "sulphates" and "alcohol" for each of the wines.

The expected output is a single value between 0 and 10. [2]

TABLE I
Wine Quality Parameters

Run	1	2	3	4
Generations	20	70	100	100
Population	200	200	200	100
Crossover	0.8	0.8	0.8	0.8
Mutation	0.2	0.2	0.2	0.2
Tournament Size	50	50	50	50
Data Type	C4	W	W	W
Data Size	18000	6000	6000	1000

V. Analysis

A. Connect 4

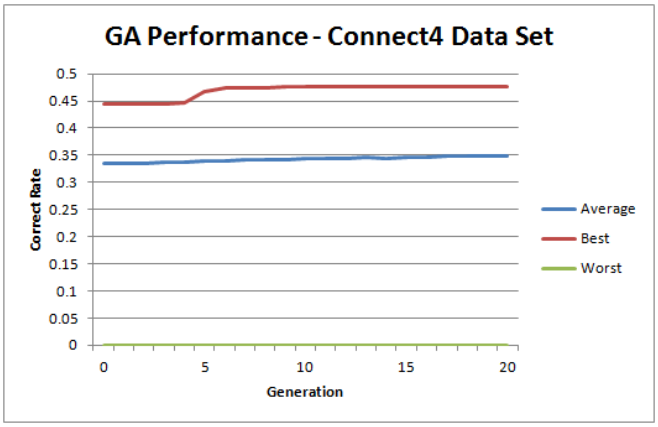

Fig. 4. An example single hidden layer neural network.

1) Union/Intersect:

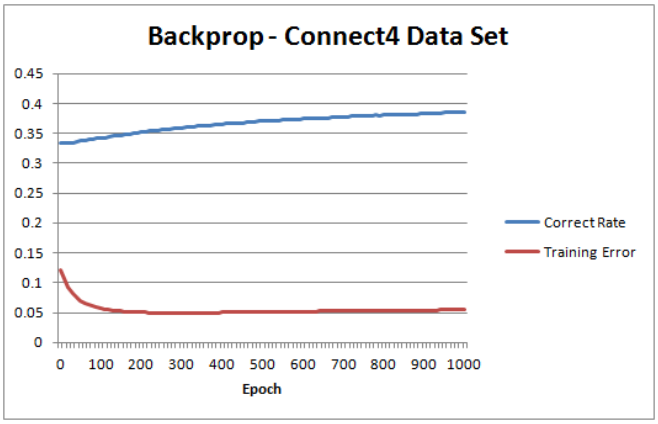

Fig. 5. An example single hidden layer neural network.

2) Vanilla Backprop:

B. Wine Quality

1) Union/Intersect:

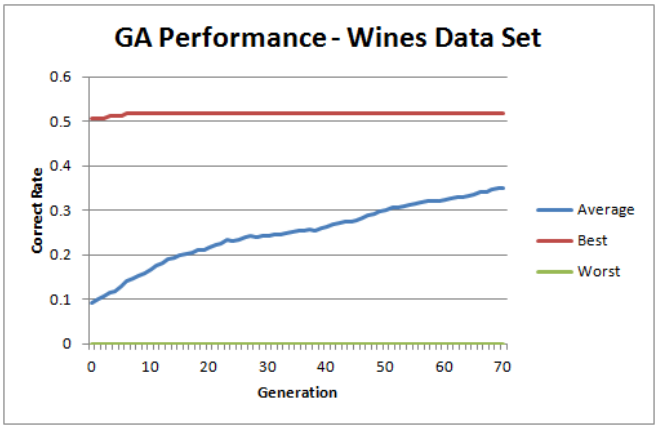

Fig. 6. An example single hidden layer neural network.

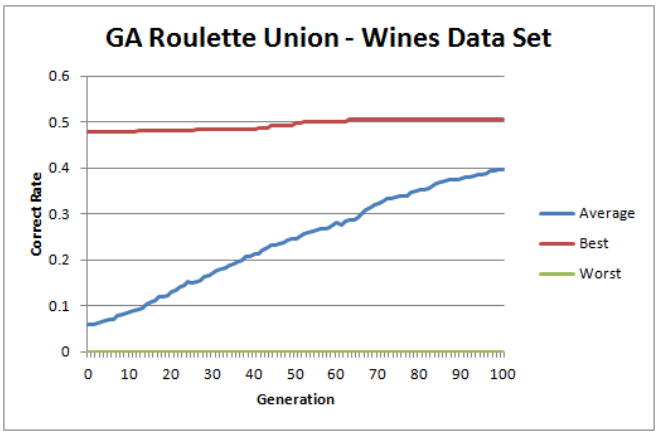

Fig. 7. An example single hidden layer neural network.

2) Uniform:

3) Change:

4) Vanilla Backprop:

VI. Conclusion

Things could have done better.
We believe this is due to crossovers used.
Blah-de-Blah.

References

[1] Lachlan Plant, Graph Theory Consultant
[2] P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553. ISSN: 0167-9236.
[3] "The Backpropagation Algorithm" Retrieved February 2013 from http://www4.rgu.ac.uk/files/chapter3 - bp.pdf
[4] Ombuki-Berman, B. "COSC 4P80 Lecture Slides" Brock University, Winter 2012/13
[5] Russell, S. J., and P. Norvig. "Artificial Intelligence, A Modern Approach." Pearson College Div, 2010.

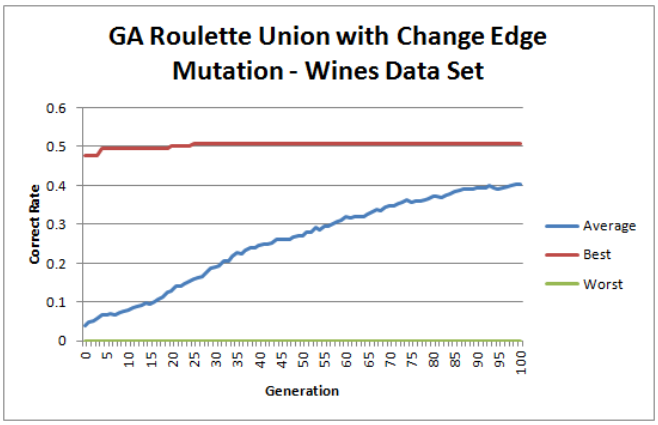

Fig. 8. An example single hidden layer neural network.

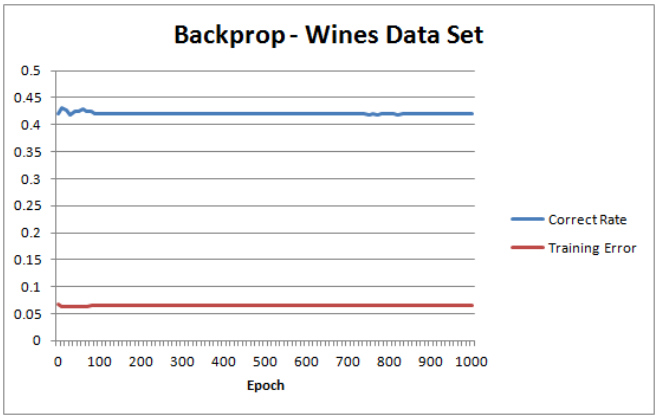

Fig. 9. An example single hidden layer neural network.