

# Evolving Complex Neural Networks That Age

John R. Podlena and Tim Hendtlass

Centre for Intelligent Systems,  
School of Biophysical Sciences and Electrical Engineering,  
Swinburne University of Technology,  
P.O. Box 218 Hawthorn 3122.  
Email : jrp@bsee.swin.edu.au & tim@bsee.swin.edu.au

## ABSTRACT

The combination of the broad problem searching capabilities of a genetic algorithm with the local maxima location capabilities of a hill climbing algorithm can be a powerful technique for solving classification problems. Producing a number of specialist artificial neural networks, each an expert on one category, can be beneficial when solving problems in which the categories are distinct. This paper describes combining genetic algorithms, hill climbing and sets of specialist artificial neural networks to solve a difficult character recognition problem. It also describes a method by which the effects of a large "elite" sub-population can be counter-balanced by using an aging coefficient in the fitness calculation.

## INTRODUCTION

The genetic algorithm (GA) is a paradigm which is based on both gene recombination and the Darwinian "survival of the fittest" theory. A group of individuals in some environment have a higher probability to reproduce if their fitness (their ability to thrive in this environment) is high. Offspring are created via a crossover operation (as in gene recombination) and mutation. The algorithm, described by Holland [1], aims to increase the average fitness of the individuals over a number of generations.

The notion of combining the broad search capabilities of the genetic algorithm and the hill climbing ability of the artificial neural network has occurred to many researchers [2]. This has led to much work in the areas of weight and topology optimisation individually, but less on the two combined. In the area of weight optimisation, a population of fixed architecture networks is "bred" (using crossover and/or mutation operations) to create offspring whose weights will hopefully be closer to a solution than their parents [3]. An individual's breeding chances is directly related to their "fitness" (e.g. average error over a test set). This method has shown promise when using either

crossover or mutation as the main GA operator. One of the problems with such net training is that the GA has trouble completing the solution to the problem and thus the GA is often only applied initially, the best network after a number of generations being left to home in on a solution using its normal learning rule [4].

Research has also been done on the topological design of neural nets employing various adaptations of the GA, most with successful results [5,6]. Some use back propagation networks, applying a genetic algorithm to the number of hidden nodes which subsequently produces the most structurally fit individual for a given task. Often this uses a specific initial architecture, and "grows" a network which has a unique (and not necessarily maximally connected) structure. One method for the addition and removal of these nodes is to map the topology into the chromosome (or genotype) such that it represents both nodes and connections, subsequent crossover causing fluctuations in the overall number of nodes [4]. However, this method does not pass on weight information between successive individuals. Each new individual must be trained from an initial weight state before it can be placed in the new generation, increasing computational time.

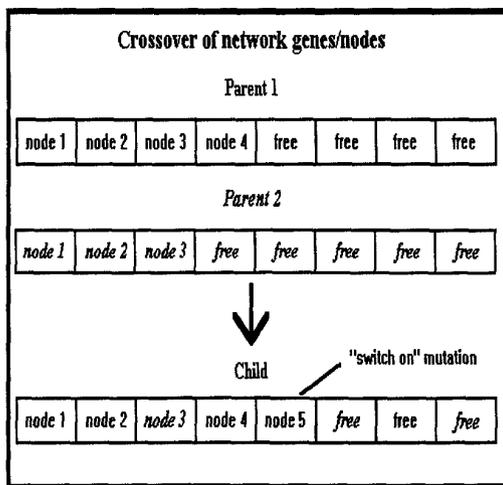
The combination of weight training by both GA and normal techniques has also been researched [2]. The weights are initially evolved using a GA for a fixed period. The best resulting network completes training by conventional methods. Some studies involving the simultaneous evolution of both the topology of the net and the weights have been reported. These involve the use of weight crossover and have shown restrictions on network complexity [7].

## THE ADAPTED GENETIC ALGORITHM

The work described in this paper uses an adaptation of the standard genetic algorithm to develop the most appropriate topology for a given architecture

and to find an appropriate learning coefficient for all nodes. Gradient descent and genetic mutation are the only mechanisms used to converge the network weights to a solution. The variation to the standard genetic algorithm described below was designed to overcome the problems reported by other researchers.

The genotype/chromosome structure used consists of a template of initially empty "slots", each slot able to hold a hidden node, complete with weights, learning coefficient and a Boolean "activation" flag. When the system is initialised, the first N nodes are activated for each individual in a given population, and the learning coefficient, number of hidden nodes and weights set to random numbers. This creates a chromosome with some slots filled/activated and others left empty. At the time of breeding these slots are crossed over, the probability for a slot from one parent being placed in the child chromosome being 0.5. This initially causes little change in structure, however when combined with a switch on/off node mutation (with equal mutation probability for on/off) the crossover method enables greater diversity in structure. A simplified example is shown below, displaying crossover at each node point (a uniform crossover [14]), whether used or "free", and a mutation in the child. The uniform crossover operator was used, despite its inherent destructiveness, due to its robustness (see [15]).



There is a low probability that the on/off state of any node may be toggled during the birth process. When a node is activated there is an equal probability that either the weights previously stored in the switched off node or an initialized set of weights will be used. This operation has biological parallels, some genes being "switched off" in cell development but still being retained in the DNA for possible next generation use. It also has practical value in its ability to retain what may be a

successful component of a previously successful individual (this gene retention is called Lamarckian learning [8]). It has been seen [7] that one problem with crossover is that parents with differing structures do not necessarily produce offspring with valid structures. By keeping the switch on/off mutation rate low, a population can at any time include a number of individuals of a particular structure, improving the probability of compatible parents breeding valid individuals.

The problem of competing conventions or the permutation problem (networks with different symmetry but similar nodal functionality competing in the same population), are often cited in literature as a major reason for distress in evolving neural networks. Literature studied in this regard appear to take the problem for granted; the few who implement preventative measures to test the theory have found that it is not a significant problem. In one study ([16]), such preventative measures were found to be actually detrimental to the algorithm's effectiveness. In this vein, no measures to remove the competing conventions phenomena appears in our algorithm.

In the adapted algorithm described in this paper, mutation is the only GA operator affecting the weights values of nodes (as a static mutation probability). Individuals do not lose nodal weight information from one generation to the next through weight crossover methods (correlation between the two parent's hidden unit individual weights values is unlikely [7]). The probability of "stagnation" of the evolution process may be diminished by attempting to ensure a diverse population of individuals. To help achieve this, the probability for mutation can be made inversely proportional to the difference between the current parents.

The algorithm implemented is as follows:

**For all individuals:**

- initialise  $i$  nodes to initial learning coefficient
- initialise all weights to random values

**Until error of best individual net below some minimal value Loop:**

- train nets in population for a fixed number of iterations
- calculate fitness based on net error and age
- calculate probability for parenting new generation
- set  $n$  worst individuals to zero mating probability ( $n=N_{dead}$ )
- copy  $m$  of the best individuals into new population ( $m=N_{copied}$ )
- increment the age of replicated individuals
- breed individuals to fill next generation pool,

**EndLoop**

## BREEDING STRATEGY

The above algorithm displays two features not yet described. The first is the "culling" of the worst N individuals in the population at breeding time by setting their probability of breeding to zero. This effectively increases the next generation's chances of producing more viable individuals by giving the 'fit' breeding parents larger "slices" of the breeding probability "pie". The second feature implemented is a further deviation from the classic genetic algorithm. Using the standard breeding method, once a parent is selected for breeding a crossover probability (often 0.5) determines whether the individual will be crossed over with another parent, or whether it will be copied intact into the next generation. One problem with such a method is that relatively low fitness individuals can be copied straight into the next generation, while highly fit individuals can be crossed over and thus have the possibility of losing their important genetic material. The benefits for always copying N of the most fit parents directly into the new generation (known as elitism, this is almost always restricted to one or two individuals) are two fold. Information from the best parents can not be lost via unfortunate crossover combinations, and the best individuals have a chance to make progress based on their local heuristic over a number of generations. One of the dangers of such an extended elitist strategy is the possibility of highly trained parents that are stuck in local minima forever being copied directly into the next generation. To encourage diversity, each parent in this algorithm has an associated "age penalty" coefficient which effectively reduces the parents fitness to breed with time. This age penalty is implemented as a fitness modifier, where the resulting fitness is equal to:

$$\text{fitness}' = \text{fitness} - (\text{fitness} - (\sum \text{pop\_fitness}_i) / \text{pop\_size}) * \text{Age} / 100,$$

where **fitness'** is the modified fitness, **fitness** is the individual's original fitness, and  $\sum \text{pop\_fitness}_i / \text{pop\_size}$  is the average (unmodified) fitness of the population.

$$\text{age} = \text{age} + \text{age\_increment}$$

where the **age** value is incremented for all replicated individuals each generation. The **age\_increment** effectively controls the number of iterations an elite individual survives.

The strategy described above for parent retention differs from the more general forms of the genetic algorithm. In one method, called the Generational Replacement Genetic Algorithm (GRGA), the new

generation replaces all the previous individuals (except those parents not crossed over). Another common method, Steady State Genetic Algorithm (SSGA), replaces only one or two members of the population. Results from the variation of the number of N individuals retained from the previous generation are outlined below.

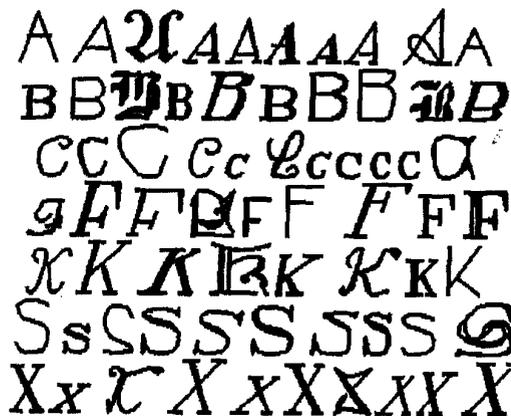
## FOUR BIT PARITY PROBLEM

The performance of the algorithm on the four bit parity problem (a classically difficult problem for the back propagation network) has previously been reported [9]. Trials involving changing the amount of least fit individuals removed from the breeding pool (Ndead) showed that setting this to any value greater than zero improved the performance, the exact value used appeared not to be important. Similarly setting the elitism parameter "Ncopied" to a non zero value also assisted, a value of approximately 1/2 the population size appearing optimum for this problem and approximately halved the time required to find the solution.

The four bit parity problem was also used to test a multiple hidden layer version of the above algorithm, which was not restricted in the number of hidden node layers an individual net could have (only bounded), and included a node connection on/off mutation. The population was initialised to random numbers of nodes, hidden layers and nodal connections. The resulting networks converged on a solution in less iterations than the single layer version, however this was counter-balanced by the complexity of the winning network's structure.

## RESULTS FROM A DIFFICULT LETTER RECOGNITION PROBLEM

The single hidden layer version of the algorithm was applied to Frey and Slate's [10] letter recognition problem data. The problem set consists of 20,000 unique letter images generated by randomly distorting pixel images of the 26 uppercase letters from 20 different commercial fonts. The fonts used included Script, Italic, Serif and Gothic. Below is an example of part of the character set and the distortions involved.



Above: An example of the character set

Sixteen numerical attributes are provided to summarise each example (e.g. dimensions of the encapsulating box, number of 'on' pixels etc.). Frey and Slate used "Holland style adaptive classifiers" (a type of machine rule induction) to solve the classification problem, training their system using the first 16,000 examples. Testing was carried out using the remaining 4,000 examples.

Our algorithm was applied to multiple sub-populations or "families" of networks, each responsible for creating individuals that recognises one specified letter (for example, the 'A' recognition task is handled by one population subset, while 'B' is handled by another). This "Bottom up" approach was chosen to reduce the complexity of the problem via the familiar "divide and conquer" paradigm [11]. Each family used a training data set (from the first 16,000 examples) which consisted of around 615 examples of the letter assigned to that family and 615 drawn from the pool of all other letters. Each family had a population size set to 40 networks. Each network

was trained using a full sequential run through this data set after which the adapted GA described above was applied to the entire family. It was found that after the GA had produced an appropriate network (had converged sufficiently), it was advantageous to "switch off" the GA and let the best network in a family home in on the solution by the normal back propagation training algorithm.

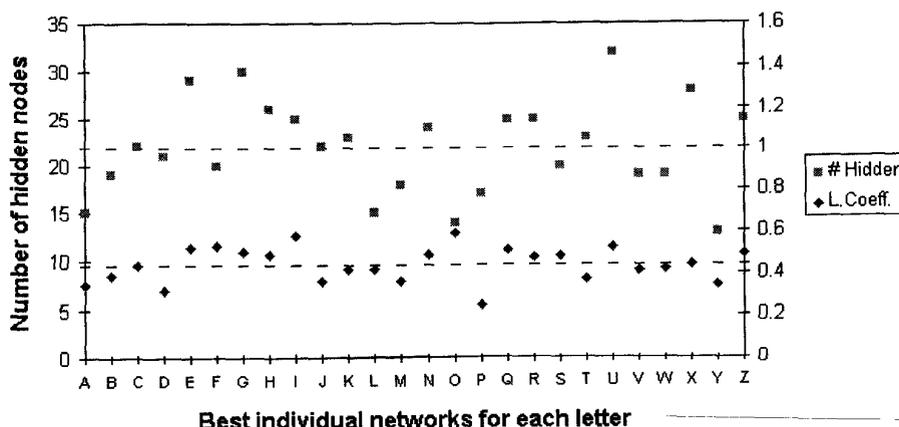
The ultimate individuals produced by the different families had a range of different numbers of hidden nodes and average learning coefficients. Figure 1 indicates the diversity of the solutions obtained by the different families for each of their respective letters. Note the deviation of learning coefficient and number of hidden nodes from their means (0.44 and 22 respectively); most letters have a learning coefficient deviation from the mean that correlates with their hidden

The above graph indicates the diversity of the solutions obtained by the different families for each of their respective letters. Note the deviation of learning coefficient and number of hidden nodes from their means (0.44 and 22 respectively); most letters have a learning coefficient deviation from the mean that correlates with their hidden node number deviation. It is interesting, but may not be significant, that those letters which do not show this correlation tended to have higher post training error rates than those which did.

After an average of 350 GA epochs the best performing network from each family was selected and subsequently trained for around a further 8,000 cycles using just the back propagation learning rule. The winning net from each of the families was shown each of the 4,000 test examples. The category assigned was that of the net with the overall highest output.

The "aging" strategy was found to be significantly

Figure 1: Learning Coeff. and # of Hidden Nodes for Best Individual Nets



beneficial in the prevention of stagnation (or getting stuck in local minima). Trial runs of the algorithm with the aging strategy produced populations with a 10% to 20% higher average fitness than those without.

The overall result obtained was 87.1% (3484 correct, 516 incorrect). With the addition of hand tuned thresholding this could be lifted 88.2% correct responses (3528 correct, 472 incorrect). These results compare favourably with Frey and Slate's best result of 82.7% which also required hand tuning, and are superior to our attempt to solve the problem on a single large network using commercial software which failed to yield results over 75%.

Another attempt to solve the problem was made by Fogarty [12], who used the "first nearest neighbour" algorithm for classification. This entailed retaining 16,000 examples in memory and using them to classify each member of the testing set. To find which class a testing example belonged to, the Euclidean distance between it and each of the 16,000 examples in memory was calculated using the 16 primitives of both the unknown and of the example being considered. The unknown was assigned the class of the closest example found. This method produced good results (average of 95.7%), however for each test input it needs 16,000 examples in memory (16 integers and one letter for each,) and needs to make 16,000 distance calculations to arrive at each classification. Using only 1,600 examples in memory reduced the average result to 82.57% correct over the 4,000 testing examples. Since our algorithm produces 26 networks with small memory requirements giving almost instant classification it obviously has significant advantages.

## CONCLUSION

The implemented algorithm successfully breeds and trains a suitable network configuration for each letter of a difficult letter recognition problem. With low mutation rates, the system changes its learning coefficient and hidden node numbers to explore different areas of the problem space. In such problems where success is highly dependent on the initial conditions and architecture of an individual network, the algorithm shows promise in its ability to change such parameters as it learns. Some of the unique features of the above algorithm are: a) its "slot" mechanism for a dynamic nodal topology, with crossover at each node point, b) Lamarckian probability mutation (the probability of keeping crossed over individual node's weights intact), c) removing the worst "Ndead" individuals from a

breeding pool and e) an aging strategy to counter-balance the negative effect of the elitist (Ncopied) operation.

The expansion of this system to a multiple sub-population algorithm, with test problems being broken down into subgroups to be learned by population subsets, was found to have superiority over the use of machine rule induction or large single neural networks for a difficult character recognition problem. The large memory and computational requirements of the nearest neighbour solution for classification on test examples make it less of a practical solution than our method.

Future work in this area may include the storage of multi-generational information for each sub-population, such as previously invalid topology configurations or initial weights, to be used by future generations to avoid going over old paths. The multiple layer / connection mutation version of the algorithm could be applied to Frey and Slate's data to compare results. The algorithm could be expanded further to include a "distributed" genetic algorithm, which would break each sub-population down further into subsets (e.g. two 'A' families with populations of  $\approx 20$  each), increasing diversity and therefore the chances of finding a near optimal solution. The interaction between the two population subsets within a family would be restricted to the occasional "migration" of a fit individual from one subset to another (as in Brill, Brown and Martin's work [13]).

## REFERENCES

1. John H. Holland, Adaptation in Natural and Artificial Systems, second edition, MIT Press, 1992.
2. International Workshop on Combinations of Genetic Algorithms and Neural Networks, COGANN-92, IEEE Computer Society Press, California, 1992.
3. D.E. Goldberg, Genetic Algorithms in Search, Optimisation, and Machine Learning, Addison-Wesley Publishing Company, Reading, Mass. 1989.
4. W. Schiffmann, M Joost, R.Werner, Synthesis and Performance Analysis of Multilayer Neural Network Architectures, University of Koblenz, Technical Report 16. 1992.
5. Guy Smith, Back Propagation With Dynamic Topology and Simple Activation Functions, International Journal of Neural Networks, Vol.2, No2/3/4, June - December 1991.
6. C.H. Chu, C.R. Chow, A Genetic Algorithm Approach to Supervised Learning for Multilayered

Networks, World Conference on Neural Networks, vol. 4, pages 744-747, 1993.

7. Peter J. Angeline, Gregory M. Sanders, Jordan B. Pollack, An Evolutionary Algorithm that Constructs Recurrent Neural Networks, IEEE Transactions on Neural Networks, vol. 5, no.1, January 1994.

8. Ackley D. H., Litman M. L., A Case for Lamarckian Evolution, In C. G. Langton (Ed.), *Artificial Life III*, Reading, MA: Addison-Wesley.

9. John R. Podlana, Tim Hendtlass, A Modified Genetic Algorithm Applied to Neural Network Design, Proceedings of the Sixth Australian Conference on Neural Networks, February 1995.

10. Peter W. Frey, David J. Slate, Letter Recognition Using Holland-Style Adaptive Classifiers, Machine Learning, 6, 161-182, 1991.

11. Christine L. Valenzuela, Antonia J. Jones, Evolutionary Divide and Conquer : A Novel Genetic Approach to the TSP, Evolutionary Computation, Volume 1, Number 4.

12. Terence C. Fogarty, First Nearest Neighbour Classification on Frey and Slate's Letter Recognition Problem, Machine Learning, 9, 387-388, 1992.

13. Frank Z. Brill, Donald E. Brown, Worthy N. Martin, Fast Genetic Selection of Features for Neural Network Classifiers, IEEE Transactions on Neural Networks, Vol. 3, No. 2, March 1992.

14. Lawrence Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.

15. William M. Spears, Kenneth A. De Jong, On the Virtues of Paramitized Uniform Crossover, Proceedings of the Fourth International Joint Conference on Genetic Algorithms, 230-236, Morgan Kaufman, C.A., 1992.

16. Peter J. Hancock, Genetic Algorithms and Permutation Problems: A Comparison of Recombination Operators for Neural Network Structure Specifications, in *IEEE COGAN-92*, IEEE Computer Society Press, California, 1992.