

2DELTA-GANN: A NEW APPROACH TO TRAINING NEURAL NETWORKS USING GENETIC ALGORITHMS *

Rajendra Krishnan

Fujitsu Australia
607 St. Kilda Rd.
Melbourne Vic 3000
krishnan@numbat.cs.rmit.oz.au

Victor B. Ciesielski

Department of Computer Science
RMIT, GPO Box 2476V
Melbourne Vic 3001
vc@goanna.cs.rmit.oz.au

Abstract

We describe a method of using Genetic Algorithms for training multi-layer perceptron networks in which the chromosomes encode “rules” for changing the network weights rather than the weights themselves. The genetic operators of crossover, selection and mutation are used to generate new rules which are then applied to the weight matrix. The approach is significantly better than other approaches to training networks using genetic algorithms and successfully solves a number of benchmark problems which are known to be difficult for backward error propagation.

1 Introduction

The determination of the weights of a multi-layer perceptron is fundamentally a multi-dimensional search problem. Backward error propagation and its variants are currently the most common methods for searching for an optimal point in weight space. However, genetic algorithms provide an alternative method for implementing search[4]. The genetic algorithm approach involves encoding potential solutions as bit strings of “chromosomes”, setting up an initial population of chromosomes and then using the genetic operators of selection, crossover and mutation to derive a new population. In theory these operators take the best potential solutions from the current generation and combine them to form a new population which should be better than the previous one. The computation of new generations continues until the best solution is ‘good enough’ by some criterion. The process requires a ‘fitness’ function which returns a numerical value for each chromosome.

In using genetic search for the determination of the optimal set of weights for a network, we have represented the network weights and biases in a chromosome. The position which encodes a single weight corresponds to a gene. Each chromosome represents the entire weight

set of a network and is an individual member of a population. The members of the population are subjected to the standard genetic operators of selection, crossover and mutation. After each new generation of chromosomes has been computed, the weights of the neural net are determined from each chromosome and the error on the training set is calculated. The error measure is used as the fitness of the chromosome.

In previous work on using genetic algorithms, Whitley and Hanson[13] and Montana and Davis[5], for example, the genes in the chromosomes encode the the weights of the neural net directly. In our approach a gene represents a ‘rule’ or method for changing the corresponding weight.

2 The Method

The various aspects of the method are described below with reference to the xor network in figure 1.

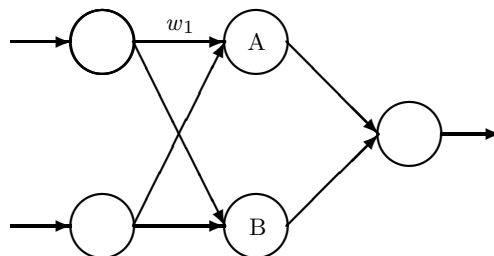


Figure 1: The XOR Network

2.1 The Gene Structure

In our approach, the gene is a composite structure:

- There are three rule bits which we call $x1, x2$ and $x3$.
- There are two floating point values which we call $delta1$ and $delta2$.

The $x1, x2$ and $x3$ values specify a heuristic rule to apply to $delta2$. $Delta1$ will then be modified by $delta2$, and this will in turn be applied to the weight or bias associated with the gene. The rule bits are interpreted as follows:

¹In A.C. Tsoi [Ed.] *Proceedings of the Fifth Australian Conference on Neural Networks*, University of Queensland, Brisbane, p38-41, 1994

```

if  $x_1$  then
  if  $x_2 \& x_3$  then
     $\delta_2 *= 2$ 
  else
     $\delta_2 /= 2$ 
   $\delta_1 += \delta_2$ 
endif
endif
weight (or bias) +=  $\delta_1$ 
end

```

For example if x_1, x_2 and x_3 are all 1 then δ_2 is doubled, the new δ_2 added to δ_1 and the new δ_1 to the weight W_1 .

The changing mechanism allows for a weight to be changed by a δ_1 value which in turn can be modified by the δ_2 value. The rate of change of the weight and δ_1 (their “gradient”) is changeable due to the δ_2 value. This is used to provide a heuristic “second order” change mechanism to the weight modification rule. This use of *deltavalues* is somewhat like the ‘delta coding’ described in [11].

Note that length of the chromosome, which becomes of the order of hundreds or thousands of bits using mechanisms such as dynamic parameter encoding (Schraudolph [8]), will be shorter using this representation. By manipulating shorter chromosomes, we hope to solve larger problems using genetic techniques.

2.2 Selection

The commonly used “roulette wheel” (Goldberg[4]) mechanism is used for parent selection. However the crossover and mutation operators need to be modified to accommodate the new gene structure.

2.3 Crossover

The crossover operator is based on parameterized uniform crossover (Syswerda[9] and Spears and DeJong[2]).

The uniform crossover mask is applied only to the *rule* bits of the chromosome, not the *deltas*. If there is an exchange of any of the rule bits during crossover, the δ_1 and δ_2 values are exchanged between the parent chromosomes.

We selected uniform crossover for its ability to combine building blocks of genetic material spread wide apart on longer chromosomes - as demonstrated by Syswerda’s experiment with the sparse one max problem in [9]. Also the effect of uniform crossover may render inversion as a reordering operator unnecessary. In the context of a neural network weight optimisation problem this becomes important. If a net has to be trained through a part of the solution space by changes to weights which are far apart on the chromosome, the linkage between the weights is less likely to be established by one and two point crossover, for example. Also mentioned in the literature is the possibility that uniform crossover can be more explorative than traditional crossover. This is important in exploring the multimodal spaces which arise in the neural net optimisation problem.

2.4 Mutation

When mutation is applied, it is applied to the rule bits and new randomly generated values of δ_1 and δ_2 are used for that gene.

When using steady state genetic algorithms, the adaptive mutation operator of Whitley[13] was used. This operator applies mutation to the offspring of crossover at a level proportional to a difference measure between the parents. However, a simplification of the operator we used is that the difference between two parents was measured roughly in terms of the difference in fitness values between the two chromosomes. No special method was used to compute a difference measure between two chromosomes with the special gene structure.

2.5 Network Topology

In addition we have made a standard modification to the structure of the neural net to avoid the *competing conventions* problem[7, p4]. The problem may be described briefly as follows: In the XOR network above, if we exchange the input and output weights and biases of units A and B, the network will continue to compute the same output. In a genetic algorithm this is a problem (considered to be a “devastating problem” by Belew, McInerney and Schraudolph[1]), as the two resulting networks will have different chromosomes but have the same fitness. Thus the population of chromosomes can end up having apparent diversity but much less real diversity, leading to the premature convergence of the genetic algorithm. To avoid this problem, we used only one hidden unit per layer, and connected each layer to all the previous layers. Figure 2 shows the network for the 423 minimal adder problem.

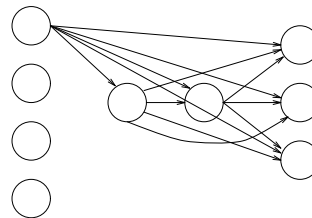


Figure 2: Network Topology for 423 Minimal Adder. (Only connections from one input unit are shown)

This gives a network structure similar to the networks built by the cascade correlation architecture of Fahlman[3]. In practice, networks with this “cascaded” style were optimised much more readily than the usual networks where there are several units per hidden layer with no interconnections within a layer.

No.	Problem	2DELTA-GANN				Whitley-GANN		BP
		Regime	Trials	Pop	Success	Trials	Pop	Trials
1	10-5-10 Encoder	Genesis	15,100	1000	100%	NA		258
2	10-5-10 Encoder	Genitor	5,800	200	75%	NA		258
3	10-5-10 Comp	Genesis	168,300	1000	42%	NA		668
4	10-10-1 Dtoa	Genesis	22,210	1000	95%	NA		248
5	10-10-10 Random	Genesis	207,430	1000	70%	NA		98
6	6-6-1 Parity	Genesis	912,000	1000	25%	NA		
6a	6-6-1 Parity	Genesis	112,000	1000				127,000
7	6-6-1 Parity	Genitor	137,200	300	28%	NA		
7a	6-6-1 Parity	Genitor	37,200	300				127,000
8	4-2-3 Minimal adder	Genesis	130,430	1000	56%	100,000	3,000	FAILS
9	4-4-3 Adder	Genesis	63,500	1000	56%	1,250,000	2,000	38,600
10	4-4-3 Adder	Genitor	33,000	100	50%	500,000	5,000	38,600
10a	4-4-3Adder	Genitor	9,000	100				

Table I: Comparison of 2DELTA-GANN with other approaches

3 Implementation

The experiments were run using two flavours of genetic algorithms:

1. Generational replacement (Genesis)
2. Steady state (Genitor)

In generational replacement storage is allocated for two populations. The new population is computed from members of the old population only. For these experiments we used the GAUCSD 1.4 program with modifications to handle the new gene structure. In the steady state approach storage is allocated for one population only. New children overwrite existing members of the population. For these experiments the GENITOR algorithm of Whitley [12] was used, modified again to suit the new gene structure. The neural network software used was essentially a modified version of the “bp” program which is available with the PDP book[6].

4 Results

The results of the running the algorithm on a selection of problems are shown in table I. Problems 1-5 were based on the set used by Veitch and Holmes[10] for benchmarking of neural network training algorithms and show the performance of 2DELTA-GANN compared to standard backpropagation. In both cases the results are averaged over 25 trials. In all problems where the output is 0/1, 2DELTA-GANN stops when all output values for 1 targets are 0.6 or higher and output values for 0 targets are 0.4 or lower, as in [10]. For the two genetic approaches in the table a ‘trial’ corresponds roughly to a fitness function evaluation. Thus 1 trial involves 1 forward pass through the training set. There is no direct equivalent to a “trial” in BP. However there is a forward and backward pass through the network for each training instance per epoch. Hence 1 epoch corresponds roughly to 2 trials. This method has been used in determining the “trials” column for BP.

- 1 This is the standard 10-5-10 encoder problem. 2DELTA-GANN using the Genesis regime solves this problem in 15100 trials. The population size used was 1000 and all attempts converged to a solution. Backprop requires 258 trials for this problem.
- 2 This is the same problem as (1), but using the Genitor control regime.
- 3 This is the 10-5-10 complement encoder[10].
- 4 This is the digital to analog problem of [10]. A 10 digit binary number containing a single 1 bit is to be mapped to a floating point output as follows: 1000000000 \rightarrow 0.1, 0100000000 \rightarrow 0.2, 0010000000 \rightarrow 0.3.... The error criterion is 0.03 for each output. If the target output is 0.4 for example, the net output is considered correct only if it is in the range 0.37 to 0.43. This problem tests “fine learning”.
- 5 This problem requires the network to learn a mapping between 10 random inputs and 10 random outputs.
- 6-7a This network requires solving the 6 bit parity problem. 6a and 7a give the results for this problem if only the runs which converged are considered. More than half of the BP runs failed on this problem.
- 8 This problem requires the addition of two 2 bit numbers into a 2 bit sum and a carry. The minimal network for this network has 2 hidden nodes and direct connections between input and output. BP becomes “stuck” in local minima on this network. Additional hidden nodes are needed[6]. The results for Whitley-GANN are taken from [13]. However, the results for the two GANNs are not directly comparable since different convergence criteria were used. The Whitley-GANN was terminated before the error had been reduced to the desired level.

9-10a The two bit adder problem with 4 units in the hidden layer. The results for Whitley-GANN are taken from [13] and [12].

On the smaller problems BP is clearly superior to both flavours of 2DELTA-GANN. However, these training problems are fairly simple, and do not pose a problem for standard learning methods. To properly compare genetic algorithm methods with methods like back-propagation, the problems must be multimodal and difficult to solve. On simpler unimodal problems we cannot expect a genetic algorithm to outperform a gradient descent method (Goldberg[4, Ch. 1]). However when we come to more difficult problems like the six bit parity problem, 2DELTA-GANN becomes competitive with back-propagation and starts to surpass it. 2DELTA-GANN is clearly superior to the Whitley-GANN in which the neural network weights are encoded directly in the chromosome.

Our results on the steady state GA (Genitor) versus generational replacement (Genesis) are in agreement with [13]. The steady state approach is clearly superior.

5 Conclusions

We have presented a method, 2DELTA-GANN, of training neural networks with genetic search. The method appears to be superior to other GANN algorithms in the literature, both in terms of number of trials to convergence and the length of the chromosome needed. On small problems BP is better than 2DELTA-GANN. However, 2DELTA-GANN solves some problems known to be very difficult for BP. Further work is needed to test 2DELTA-GANN on difficult problems.

An advantage of 2DELTA-GANN is that it needs no gradient information. This means that one can experiment with different types of activation functions for the neural network. Another advantage can be realised when the GA is run on massively parallel machines. The parallel nature of the GA means that running on such machines we can hope to achieve speedups of upto a factor of the population size, as the individual population members can be applied to separate copies of the neural network and evaluated in parallel. On such a machine we can expect the GA to outperform BP comfortably.

References

- [1] R.K. Belew, J. McInerney, and N.N. Schraudolph. Evolving networks: Using the genetic algorithms with connectionist learning. CSE Technical Report CS90-174, University of California, 1990.
- [2] K.A. DeJong and W.M. Spears. On the virtues of parameterized uniform crossover. In Richard K. Belew and Lashon B Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, San Mateo, July 1991. Morgan Kaufman.
- [3] S.E. Fahlman and C. LeBierre. The cascade correlation learning architecture. Technical Report CMU-CS-90-100, Carnegie Mellon University, 1990.
- [4] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine learning*. Addison Wesley, Reading, Ma, 1988.
- [5] D. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th International on Artificial Intelligence*, pages 762–767, 1989.
- [6] D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing: Explorations in the microstructure of Cognition*. MIT Press, Cambridge, Massachusetts, 1989.
- [7] J. David Schaffer, Darrel Whitley, and Larry J. Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *COGANN-92, International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37, Baltimore, June 1992. IEEE Computer Society Press.
- [8] N.N. Schraudolph and R.K. Belew. Dynamic parameter encoding for genetic algorithms. CSE Technical Report CS90-175, University of California, 1990.
- [9] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9, San Mateo, 1989. Morgan Kaufman.
- [10] A.C. Veitch and G. Holmes. Benchmarking and fast learning in neural networks - results for back-propagation. In *Proceedings of the 2nd Australian Conference on Neural Networks*, pages 133–139, Sydney, 1991.
- [11] D. Whitley, K. Mathias, and P. Fitzhorn. Delta coding: An iterative search strategy for genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 77–84, San Mateo, 1991. Morgan Kaufman.
- [12] Darrel Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121, San Mateo, 1989. Morgan Kaufman.
- [13] Darrel Whitley and Thomas Hanson. Optimizing neural networks using faster more accurate genetic search. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 391–396, San Mateo, 1989. Morgan Kaufman.