Improving Classification Performance in the Bumptree Network by Optimising Topology with a Genetic Algorithm

Bryn V. Williams, Richard T. J. Bostock, David Bounds and Alan Harget
Department of Computer Science and Applied Mathematics, Aston University, Birmingham B4 7ET, UK
williabv@cs.aston.ac.uk

ABSTRACT

This paper presents a successful synthesis of evolutionary and connectionist methods, based on the genetic optimisation of a recently introduced neural network model, the bumptree network. We show that the bumptree network is inherently more suited to optimisation by a genetic algorithm (GA) than other neural network models such as the multi-layer perceptron (MLP). We describe a hierarchical genetic coding which addresses the problem of representing certain strong dependencies which exist between the bumptree's structural parameters, and show that our coding scheme has the desirable properties of *continuity*, *isomorphism*, *completeness*, *closure* and *low redundancy* with respect to the space of possible bumptree structures. We present empirical results which show that bumptree networks evolved by the GA significantly outperform the orthodox bumptree on several tasks, including the difficult real-world classification task of spoken vowel recognition.

1. INTRODUCTION

Recently, many researchers have sought to combine the GA and connectionist paradigms in hybrid learning systems, with varying degrees of success. Jones (1993) and Yao (1992) have produced recent bibliographies of this work. Although the exact nature of the hybridisation has varied, one factor which remains constant across the vast majority of the published work in this area is the neural network model chosen; almost all of these hybrid systems have been built around the multi-layer perceptron (MLP) neural network (Rumelhart, Hinton & Williams, 1986), the most widely used and understood learning model in the connectionist school.

In this paper, we describe a hybrid system in which a GA is used to optimise a rather different connectionist model, based on the bumptree structure (Omohundro, 1991), a binary tree of Gaussian functions which partitions a Euclidian space. Bostock (1993) has found that a connectionist classifier system based on the bumptree structure offers classification performance comparable to other neural network models, while being significantly faster in both training and access.

2. THE BUMPTREE NETWORK

The majority of connectionist classifiers are based on the MLP, in which each node i computes an output activation, y_i, as a function of its inputs:

$$y_i = f\left(\sum_j w_{ij} x_j\right) \qquad (1)$$

where the jth node input has activation x_j and connection weight w_{ij}, and f is a non-linear thresholding function, typically sigmoidal. Each such unit defines a decision surface in the form of a hyperplane in its input space. In an MLP trained for classification, these hyperplanes model actual class boundaries in the training data.

In the bumptree network, each node computes a more complex activation function, such as:

$$y_i = \prod_j \left(\frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}} \right) \qquad (2)$$

Here, y_i is a product of Gaussians, where μ_j and σ_j^2 are the mean and variance, respectively, of a Gaussian associated with the *j*th input. The decision surface of a node such as this is a hyper-ellipse which encloses a local region of the node's input space. The bumptree itself is a hierarchy of such nodes in the form of a binary tree, with the defining constraint that:

> ". . . each interior node's function must be everywhere larger than each of the functions associated with the leaves beneath it."
>
> (Omohundro, 1991) (i)

At the top level of the tree there are two nodes, which coarsely divide the input space into two regions. Each region is further divided into two smaller sub-regions, and so on. The purpose of the bumptree is to divide the input space into successively smaller regions until, at the leaf layer, the classification task has been reduced to a set of linearly-separable sub-tasks. Associated with each leaf node is a 'local expert', a linear network responsible for the classification of points which fall within that leaf node's active region. When a point x is input to the network, a descent of the tree, choosing the more active branch at each interior node, provides efficient access to the appropriate leaf node, l. The point is then classified according to the outputs y_i of the expert associated with node l:

$$y_i = \sum_j w_{ij}^l x_j \tag{3}$$

where the expert is a single layer network of weights w_{ij}^l connecting classifier input units j to output units i.

To train the bumptree network, the first step is to determine the topology of the tree, i.e. the number of layers and the centres (means) and radii (variances) of the node functions. Normally, the tree is built in a top-down manner by the repeated application of a local clustering algorithm to position the node functions. This process is described in (Bostock 1993) and is beyond the scope of this paper. In the present work, we have used a GA to evolve tree topology. Having constructed the tree, the weights associated with each expert linear network can be solved for exactly in a single pass to minimise training error, as in (Renals and Rohwer, 1989).

A major advantage of the bumptree over other connectionist models is speed, both in terms of training time, and in access of the trained network. With the bumptree, it is only necessary to compute the activation of two functions at each level of the tree in order to classify a point. An MLP with n hidden units requires n hyperplanes to be evaluated each time a point is classified. In contrast, a complete bumptree with n terminal functions has $\log_2 n$ layers, and so only requires $2\log_2 n$ function evaluations to classify a point. Thus the bumptree offers a significant access-speed advantage over the MLP, and, furthermore, this speed advantage increases exponentially with increasing network size. During training, the bumptree's output weights are minimised in a single pass through the training set, whereas the MLP typically employs an iterative gradient descent method requiring thousands of passes. The speed advantages alone make the bumptree a much more attractive prospect for optimisation with a GA than the MLP. As the classifier forms the evaluation function for the GA, it operates in the innermost loop of the algorithm. Any increase in speed of the evaluation function is multiplied by every chromosome evaluated in every generation. In GA experiments, the kind of speedup offered by the bumptree over the MLP can make the difference between hours of computing time, and days.

3. CODING BUMPTREE TOPOLOGY

GAs operate on a coding of the parameters to be optimised. It is increasingly recognised that a GA's effectiveness depends critically on the choice of an appropriate coding scheme and suitable genetic operators for the given problem. Unfortunately, finding the best coding and genetic operators is not generally a trivial task, and is, in fact, a major stumbling-block which plagues efforts to optimise neural networks with GAs (e.g. Hancock 1992, Radcliffe 1991). In this section we describe our coding scheme for the bumptree.

The bumptree classifier is coded as a chromosome of fixed length, built up of 30 blocks of real-valued genes, as shown in figure 1.

Each block defines a single node in the tree, with each gene in a block coding a separate parameter for that node, such as a function radius. The largest bumptree that this coding can represent therefore has 30 nodes, i.e. is a complete tree with 4 layers. This limit is rather arbitrary, and serves to keep the chromosome to a manageable length without severely limiting the expressive power of the trees produced. The first two blocks on the chromosome define the two root-level nodes, which are always present in the decoded tree. Each node has a single gene which determines whether it is a terminal (leaf) node, or has two descendants. Each node is coded at a fixed position on the chromosome, so if the node defined by block n is non-terminal, its two descendants are defined by blocks $2n+1$ and $2n+2$ (see figure 1).

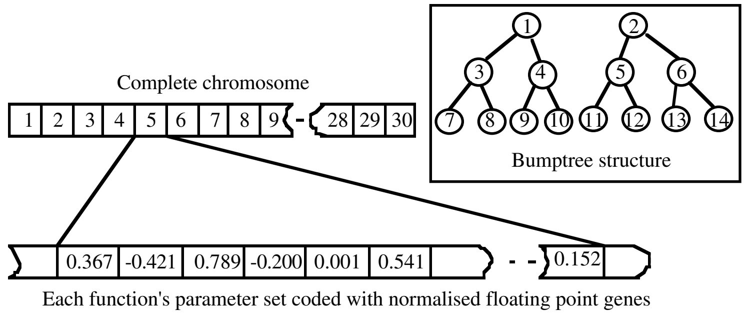

Figure 1. Genetic representation of bumptree parameters

The problem with coding structural parameters for the bumptree is that there exist strong dependencies between the parameters. By definition (i), a node function must always be contained within its parent's volume. Thus, the limits on a function's radii and centre co-ordinates are not known *a priori,* but depend on the centre and radii of its parent. A GA operating with a naïve coding scheme, where each function's centre and radii are coded in a global co-ordinate system, will be unlikely to generate legal bumptree structures; even given an initial population of legal bumptrees, recombination or mutation of chromosomes would inevitably produce children which violate (i).

Our genetic coding is tailored to address this problem. In our coding, every gene is a single real value, constrained to the range ±1.0. The process of translating a chromosome into a bumptree is represented diagrammatically in figure 2, and is as follows. First, it is necessary to define the volume in which the bumptree is to be constructed, i.e. some region of the input space (fig. 2(a)). We define this volume as a hyper-ellipse, centred on the mean of the training data and large enough in each dimension to enclose all the data points in the training set. We then define a co-ordinate system based on this volume, in which the origin is at the volume's centre, and, in each dimension of the space, a unit offset from the origin is equal to the volume's radius in that dimension (fig. 2(b)). The genes which code the centre co-ordinates of each of the two root-level functions are interpreted in terms of this co-ordinate system. In each dimension, the value of the gene which codes a root-level function's radius is mapped onto the region between the function's centre and the perimeter of the volume within which the bumptree is contained. Having constructed the two root-level functions, each non-root-level function in the tree is constructed in a similar manner, a function's co-ordinate system being defined by the volume enclosed by its parent (figs. 2(c), 2(d)).

Below we show that this coding scheme has properties which support meaningful mutation and recombination, which we refer to as *continuity* and *isomorphism* respectively. We also consider the fundamental properties of *completeness* and *closure*, which Gruau (1992) defines as desirable in a genetic coding, and finally show that our coding has the important quality of *low redundancy.*

Continuity. In a real-valued coding, 'creeping' mutation (e.g. Davis 1991), combined with selective pressure, leads to genetic hill-climbing, fine-tuning solutions and improving the genetic stock. For this hill-climbing to occur, there must be continuity in the fitness landscape, which is generally equivalent to saying that a small change in the genotype should cause a small change in the phenotype. In our coding, there is generally a smooth and continuous mapping between the space of chromosomes and the space of bumptrees. As the coding is hierarchical, a small change in the dimensions of one function in the tree will be transmitted down through its descendants, stretching or compressing them slightly and preserving structural relationships. Also, since all genes are real values in the same range, ±1.0, no gene is any more sensitive to perturbation by a given amount than any other. This is an improvement on most binary codings, in which some bits are more significant than others, or real-valued codings which are not normalised. We therefore say that our coding has good continuity, and effectively supports the beneficial aspects of mutation.

Isomorphism. Crossover must be able to assort and recombine meaningful building blocks. This means that any structure or modularity known to exist in the problem should be preserved in the genetic code, otherwise useful information is discarded and the GA's efficiency is reduced. For the bumptree, the minimum functionally relevant component is an individual node. On a larger scale, whole subtrees also represent meaningful functional units. As node parameters map to constant positions on the chromosome in our coding, it is simple to restrict crossover points to node boundaries. It is also simple to implement a subtree crossover operator, similar to that used by Koza (1992), to exchange equivalent subtrees between parents. Finally, it is worth noting that crossover in a real-valued coding only recombines values which exist in the population. In a binary coding, crossover typically splits binary words, effectively causing random implicit mutation of parameters coded at the crossover boundaries. We say that our coding has the desirable property of isomorphism, as it supports operators which perform meaningful recombination of relevant bumptree building blocks.

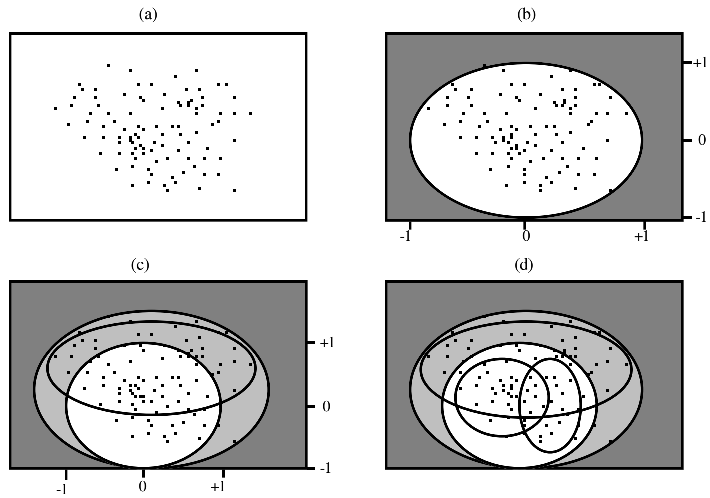

Figure 2. Translation of normalised chromosome into bumptree structure

Completeness. Given the constraint that some maximum number of nodes is to be allowed in the tree, our coding scheme is clearly able to represent all possible bumptrees within a given volume of discrete space, so we say that the representation is effectively complete.

Closure. An ideal coding should be closed with respect to the space of possible phenotypes. For the bumptree, this means that every possible genetic string must map to a legal tree structure. In our representation scheme, functions' centres and radii are coded as proportions of their parents' volumes, and this ensures that the integrity of the tree structure is inherent to the coding. It is impossible to represent an illegal bumptree structure, so the representation is closed.

Low redundancy. Problems arise if two identical bumptrees can have quite different genotypes, for instance if both have the same genetic information but coded in a different order on their chromosomes. Recombination then tends to duplicate some selective traits and omit others, leading to a situation analogous to two normal people producing a child with two left feet. This effect reduces the GA's ability to discover and combine good genetic building blocks. This kind of redundancy in genetic codings for neural networks has been called the 'permutation problem' (e.g. Radcliffe, 1991), and has proved a formidable obstacle in attempts to optimise the MLP genetically. Consider using a GA to optimise a fully-connected MLP with a single hidden layer of n nodes, coded genetically as an ordered list. As the functionality of the MLP is independent of any order in the hidden units, there are $n!$ different chromosomes which represent effectively the same network. This drastically increases the size of the search space the GA has to work with, and introduces discontinuities of the 'two left feet' kind. With the MLP, this effect explodes in severity with increasing network size.

Now consider the case of the bumptree. Each non-terminal node has two descendants, which may be transposed without affecting the bumptree's functionality. The total number of ways of ordering the bumptree without affecting its functionality is the product of the number of permutations at each layer. For a complete tree with n terminal nodes, this product evaluates to 2^{n-1}, so there are 2^{n-1} different chromosomes for each possible bumptree. Clearly this is not an ideal one-to-one mapping, but it is a significant improvement on the MLP situation. Aside from the trivial result that $2^{n-1} \leq n!$ for all positive n, the amount of coding redundancy for the bumptree only scales geometrically with increasing network size, as opposed to the factorial growth with the MLP. Using an order-based coding, a bumptree with 8 units in the leaf layer can be represented by 128 different chromosomes. For an equivalent MLP, with 8 hidden units, the figure is 40320. For 10 units the figures rise to 512 for the bumptree, compared to more than 3.6 million for the MLP.

4. EXPERIMENTS

We have tested the genetically-optimised bumptree network against an orthodox bumptree network on three problems. The first is the IRIS problem, a rather simple benchmark classification task well-known in the connectionist field. The second, PARITY-6, is the six-bit parity mapping, i.e. a six-dimensional version of XOR. As classes are highly inseparable, this data has proved extremely problematic for the orthodox bumptree. Finally, we have applied the GA-bumptree to the difficult real-world classification task of spoken vowel recognition. For this task, speech input is pre-processed to reduce each utterance to a point in a 2 dimensional input space, which is classified into one of 10 output classes corresponding to different vowel sounds. The data is split into a training set and a test set, each containing 320 example patterns. In all these experiments the fitness function for the GA is the inverse of the mean square error (MSE) across the bumptree's output units, across all examples in the training set.

The GA uses local selection, reproduction and replacement strategies, as in (Davidor, 1991) and (Collins & Jefferson, 1991), as a preliminary study found that this helps to prevent premature convergence and improves the search (Williams, 1993). The choice of other experimental parameters was also based on these preliminary experiments. A population size of 400 is used, mapped onto a 20x20 toroidal lattice, and 10% of the population is replaced in each generation. Candidates for selection and replacement are the fittest and least-fit, respectively, encountered during random walks of 5 cells from a randomly selected initial point on the lattice. The subtree crossover operator selects a random node in the tree and swaps subtrees beginning at that node between two parents, and mutation is applied to every new chromosome generated, by slightly perturbing the values of 10% of the genes, sampled with uniform probability along the length of the chromosome. A mutated gene is modified by a random quantity uniformly distributed in the range ± 0.2. This may seem like a large amount of mutation compared to mutation rates typically reported in the literature, but recall that our crossover operator does not introduce any additional mutation, as is often the case with binary GAs. The mutation operator also checks each mutated gene's new value and 'clips' this value if necessary to prevent it from exceeding the normalised range of ± 1.0. Experiments were run for a maximum of 20000 trials, i.e. 500 generations, and all results are averaged over 10 runs.

A common practice in the neural network literature involves monitoring performance on a test set during training, and terminating training when generalisation ability peaks, before over-fitting occurs. The same method can be applied to the GA-bumptree classifier, by terminating evolution before the GA begins to produce bumptrees which over-fit the training set. While we accept that this traditional method does not give a true measure of a system's ability to generalise, it does provide a simple framework for comparison between connectionist classifiers, and is the method which has been used here for both the orthodox and GA-bumptrees. Table 1 compares the GA-bumptree classifier's performance, in terms of the number of examples correctly classified after training is terminated, with that of the orthodox bumptree.

	IRIS						PARITY-6				VOWEL RECOGNITION					
	Training %		Test %		# nodes		Training %		# nodes		Training %		Test %		# nodes	
	mean	s.d.	mean	s.d.	mean	s.d.	mean	s.d.	mean	s.d.	mean	s.d.	mean	s.d.	mean	s.d.
BT	95.5	3.2	92.1	3.4	15.2	7.9	56.9	6.0	9.4	6.7	72.0	7.1	65.6	6.1	52.6	40.2
GA	99.7	0.5	96.3	0.8	9.0	4.9	77.2	8.4	9.0	3.6	78.4	1.5	75.0	2.4	18.4	2.9

Table 1. Comparison of results for orthodox bumptree (BT) and GA-bumptree (GA).

These results indicate that the GA-bumptree offers significantly increased performance on these problems. Furthermore, with the exception of the PARITY-6 problem, trees produced by the GA are significantly smaller than those produced by the orthodox bumptree algorithm.

5. CONCLUSIONS

We have described a neural network system based on the bumptree structure, and have shown that the bumptree network is naturally more suited to hybridisation with a GA than the MLP. This is because the bumptree's speed offers a significant advantage from the GA point of view, and because our genetic coding contains significantly less redundancy than is typical in hybrid GA/NN systems, making the GA-bumptree classifier virtually immune to the serious 'permutation problems' which afflict other GA/NN hybrids. As well as this *low redundancy*, we have shown that our genetic coding scheme, based on a hierarchy of normalised floating-point genes, has the desirable properties of *continuity*, *isomorphism*, *closure* and *completeness* with respect to the problem space.

The GA-bumptree is a promising hybrid between connectionist and evolutionary methods. For the classification tasks studied, the GA is consistently able to discover bumptree topologies which are smaller than, and which give a significant improvement in classification performance over, trees built using the orthodox top-down bumptree construction algorithm. Work is in progress to develop the model further.

REFERENCES

Bostock, R. T. J., 1993, Doctoral dissertation, in preparation, Dept. of Computer Science and Applied Mathematics, Aston University, UK.

Collins, R. J. and Jefferson, D. R., 1991, AntFarm: Towards Simulated Evolution, *Artificial Life,* II, Addison-Wesley, New York.

Davidor, Y., 1991, A Naturally Occuring Niche & Species Phenomenon: The Model and First Results, *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann.

Davis, L., 1991, *Handbook of Genetic Algorithms,* ch. 5, Van Nostrand Reinhold, New York.

Gruau F., 1992, Cellular Encoding of Genetic Neural Networks, TR 92.21, Laboratoire de l'Informatique pour le Parallélisme, Ecole Normale Supérieure de Lyon.

Hancock, P. J. B., 1992, Coding Strategies for Genetic Algorithms and Neural Nets, Doctoral dissertation, Centre for Cognitive and Computational Neuroscience, University of Stirling, Scotland.

Jones, A. J., 1993, Genetic Algorithms and Their Application to the Design of Neural Networks, *Neural Computing and Applications,* 1, pp 32-45, Springer-Verlag, London.

Koza, J. R., 1992, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, Massachusetts.

Omohundro, S. M., 1991, Bumptrees for Efficient Function, Constraint, and Classification Learning, *Advances in Neural Information Processing Systems,* **3**, Morgan Kaufmann.

Radcliffe, N., 1991, Genetic Set Recombination and its Application to Neural Network Topology Optimisation, Report No. EPCC-TR-91-21, University of Edinburgh.

Renals, S. and Rohwer, R., 1989, Phoneme Classification Experiments Using Radial Basis Functions. *International Joint Conference on Neural Networks*, I, pp 461-467, Washington D.C.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J., 1986, Learning Internal Representations by Error Propagation, *Parallel Distributed Processing,* **1**, pp 318-362 MIT Press, Cambridge MA.

Williams, B. V., 1993, in preparation.

Yao, X., 1992, A Review of Evolutionary Neural Networks, Commonwealth Scientific and Industrial Research Organisation Technical Report, Victoria, Australia.