Artificial Neural Network Architectures via Genetic Algorithms using Nested Graph Grammars

Wolfgang W. Bein * Bradley J. Hendricks †

Abstract

We give a new encoding scheme for Artificial Neural Network architectures which is inspired by quad-tree structures from geometric design. We define a novel nested grammar scheme to represent encodings. We then use a genetic algorithm via graph grammar chromosome to evolve network topologies. As in direct encoding the scheme allows for an intuitive interpretation of mutation and crossover operations while offering the advantages of scalability through graph grammars.

1 Introduction

Early work on applying genetic algorithms to Artificial Neural Networks (ANNs) by Montana, Davis and others [6] recognized genetic algorithms as an alternative tool to backpropagation for deriving a satisfactory weighting method [5]. Subsequently work [3, 4] went further and proposed the use of genetic algorithms to evolve network topologies. (See Mitchell [5] and Golden [2] for further background.)

In direct encoding ([4]) the artificial neural network architecture is encoded as a binary matrix and a crossover operator typically divides the parent matrices into four quadrants, by randomly selecting locations to slice the array vertically and horizontally. The children result from taking the upper left and lower right quadrants from one parent and merging them with the lower left and upper right quadrants from the other. It is known that such an approach can be successful in special cases, but lacks scalability. Kitano [3] shows that scalability can be better achieved by the use of graph grammars.

Our graph grammar approach is inspired by quad-tree structures, which are commonly used in geometric design. There, a quadratic surface is divided into four sub-squares, which are subdivided further, until a prescribed granularity is obtained (see Chang, Bein and Angel [1] for an application of quad-trees to geometric design in geometric parallel computing.) Our approach is similar to direct encoding in that genetic material is interchanged from different quadrants of the the parents (see Figure 1.) The difference is that our graph grammar is designed facilitate interchange genetic material in parallel at various level of subdivision.

*Department of Computer Science, University of Nevada, Las Vegas, NV 89154. Email: bein@cs.unlv.edu. Research supported by a UNLV SITE grant.

†Department of Computer Science, University of Nevada, Las Vegas, NV 89154. Email: hendrick@cs.unlv.edu.

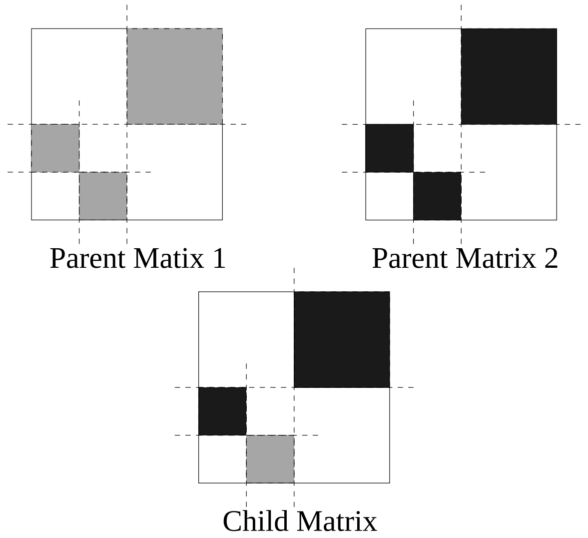

Figure 1: Exchange of genetic material at various levels of subdivision.

2 A Nested Graph Grammar

We are using a graph-generation grammar to define a square two-dimensional matrix M of zeros and ones. As usual, the grammar has a starting symbol S, and terminal symbols, which we denote by lower case letters $a, b \ldots$. The terminal symbols represent 2×2 matrices of zeros and ones, which are the ultimate building blocks of M. All rules involving non-terminal symbols are of the form $\mathcal{X} \to \begin{matrix} \mathcal{A} & \mathcal{B} \\ \mathcal{C} & \mathcal{D} \end{matrix}$, which we call *standard form*. (Note that $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$ could have repeated symbols.) Finally, for the terminal symbols there are rules of the form $\mathcal{X} \to \begin{matrix} \alpha & \beta \\ \gamma & \delta \end{matrix}$, where α, β, γ, and δ are terminal symbols, *i.e.* lower case letters representing 2×2 matrices of zeros and ones.

Since S is the start symbol, in the initial rule we have $\mathcal{X} = S$. For each of the symbols $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$ there are rules in standard form, which in turn have further rules, giving the grammar a nested structure. We say that a symbol which can be reached by applying i consecutive rules has nesting depth i. If the largest nesting depth is n, then the resulting matrix $\mathcal{A}$ has dimension $d = 2^{n+1}$. Symbols at depth i represent $\frac{d}{2^i}$-dimensional square sub-matrices of M.

3 The Graph Grammar Genome

Such a graph grammar can be represented as an array, which lists the rules in a specific order. Because the genome is divided into rules four symbols wide, designing a crossover is easy: The crossover, which we call the *standard rule crossover*, selects a set of rules from one parent and replaces those rules in the second parent to generate a new child. One could consider other crossovers, which involve interchanging genetic material between two individual rules, but the standard rule crossover gave the best results in our study. For mutation, a certain (low) percentage of the symbols is changes at random.

We remark that the ordering which we use is guided by the nested structure of the grammar: The first four locations in the array hold the starting rule. The remaining array is divided into four equal sections, each section representing the letters $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$ in that order. Recursively, each of these sections is further subdivided in the same manner. Using this approach, running through the grammar to generate the matrix can then be accomplished by a straightforward recursion.

4 Phenotype

Given, then, a matrix M defined via a graph grammar, we next derive the phenotype, *i.e.* the ANN architecture. We note that here we are restricting our interpretation of M to feed-forward ANN architectures, as in [6]. However, this is in no way a necessary restriction, as the graph grammar can, in principle, produce any 0-1 matrix M. Therefore, one can force a certain class of structures, by redefining how the ANN architecture is derived from M.

As usual, we interpret the matrix as the adjacency matrix of a directed network. For an architecture requiring i input nodes, the first i nodes in the matrix with ones in their diagonals are chosen as the input nodes. In the same way, for j output nodes, the last j nodes are selected. Any edges incoming to the input nodes are ignored, as well as any outgoing edges from the output. In a breadth first manner, nodes are added one layer at a time. The algorithm continues to add nodes and layers until no more can be added. Any connections between the last layer added and the previously chosen output nodes are then inserted, completing the network architecture.

5 Weighted Networks

Given a population of ANN architectures and a set of training data, a fitness value for every individual must be computed. For feed-forward networks this is usually accomplished by standard backpropagation [2]. Instead, we use a secondary genetic algorithm to evolve weights. The genetic operators are as implemented by Montana and Davis (see [6]).

For the primary algorithm the fitness of a

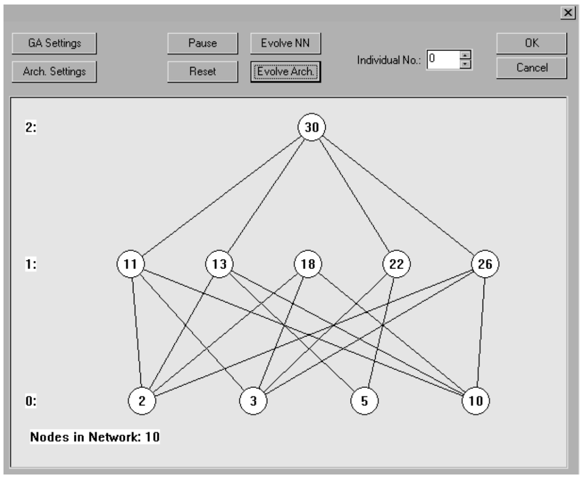

Figure 2: ANN Architecture for the Diamond

graph grammar chromosome uses the score obtained via the secondary algorithm. That fitness can be further modified to contain information about the number of nodes, edges and layers. This information is then used by the primary genetic algorithm to prefer smaller networks throughout the evolution.

6 Results

We implemented our approach under GALib, an object-oriented library for genetic algorithms, which was recently developed by Matthew Wall at MIT [7]. We performed test runs to develop best architectures with two simple x-or variants and a more complicated geometric problem.

X-or has two input nodes and one output node which fires exactly if one input node is stimulated. The x-or3 problem simply counts the number of stimuli on the input; if the number of stimuli is odd, the output was to fire. The genetic algorithm always found correct and smallest architectures for the x-or problems in a few generations.

For the geometric problem, the network received three floating valued inputs, representing x, y, and z coordinates. A fourth input node is used as a threshold node. The network was to calculate if the point lay within a three-dimensional diamond in space, satisfying the equation $|x| + |y| + |z| \leq 2.0$. If the point satisfied the inequality, the output was to fire.

We generated a set of 200 random points with values between -2.0 and 2.0 and considered graph grammars with up to 32 nodes. To ensure that approximately the same number of points fell inside and outside the region, we modified the probabilities of the random number generator accordingly. Resulting networks were then tested on unbiased data. The resulting best architecture of Figure 2 had only 10 nodes and performed significantly better than other fixed architectures of similar size with complete connection among layers.

References

[1] Long C. Chang, Wolfgang W. Bein, and Edward Angel. Surface intersection using parallelism. *Computer-Aided Geometric Design*, 11:39–69, 1994.

[2] Richard M. Golden. *Mathematical Methods for Neural Network Analysis and Design*. MIT Press, Cambridge, Massachusetts, 1996.

[3] Hiroaki Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4:461–476, 1990.

[4] Geoffrey F. Miller, Peter M. Todd, and Shailesh U. Hedge. Designing neural networks using genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.

[5] Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, Massachusetts, 1996.

[6] David J. Montana and Lawrence D. Davis. Training feedforward networks using genetic algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1989.

[7] Matthew Wall. *GAlib: A C++ Library of Genetic Algorithm Components*. Cambridge, Massachusetts, version 2.4 edition, 1996.