# Using attribute grammars for the genetic selection of backpropagation networks for character recognition

Roger A. Browse[a,b], Talib S. Hussain[a] and Matthew B. Smillie[a]

[a]Department of Computing and Information Science, [b]Department of Psychology
Queen's University, Kingston, Ontario, Canada K7L 3N6

## ABSTRACT

Determining exactly which neural network architecture, with which parameters, will provide the best solution to a classification task is often based upon the intuitions and experience of the implementers of neural network solutions. The research presented in this paper is centered on the development of automated methods for the selection of appropriate networks, as applied to character recognition. The Network Generating Attribute Grammar Encoding (NGAGE) system is a compact and general method for the specification of commonly accepted network architectures that can be easily expanded to include novel architectures, or that can be easily restricted to a small subset of some known architecture. Within this system, the context-free component of the attribute grammar specifies a class of basic architectures by using the non-terminals to represent network layers and component structures. The inherited and synthesized attributes indicate the connections necessary to develop a functioning network from any parse tree that is generated from the grammar. The attribute grammar encoding is particularly conducive to the use of genetic algorithms as a strategy for searching the space of possible networks. The resultant parse trees are used as the genetic code, permitting a variety of different genetic manipulations. We apply this approach in the generation of backpropagation networks for recognition of characters from a set consisting of 20,000 examples of 26 letters.

## 1. INTRODUCTION

Neural networks have been shown to be a suitable and effective means of solving classification problems such as character recognition. There is, however, a great variety of neural network architectures, and it is often difficult to determine exactly which one, with which parameters will provide the best, or even a reasonable solution. The enormous space of possible networks, along with the usually very long training times, precludes exhaustive search of the space of possible networks. Some analytic techniques exist that may help narrow the search, but generally the greatest reliance is upon the intuitions and experience of the implementers of neural network solutions.

The research presented in this paper is centered on the development of automated methods for the selection of appropriate networks[1,2,3], as applied to a character recognition problem. We have developed a formal specification for both classes of, and individual neural networks. The specification provides a representation that is successfully manipulated with genetic search algorithms[4] to determine the particular architectures that perform best on a task. In this paper we describe our approach applied to a large space of backpropagation networks in the task of recognizing characters.

## 2. NETWORK GENERATING ATTRIBUTE GRAMMARS

In order to search through a space of possible networks, one must first develop the ability to specify a concise set of useful networks, and to provide the ability to generate functioning instances from that set. One would never wish to consider the entire space of all connected graphs. What is required is a compact and general method for the specification of commonly accepted network architectures that can be easily expanded to include novel architectures, or that can on the other hand be easily restricted to a small subset of some known architecture. In order to facilitate search through the space of networks, the specification method must be accompanied by a process that can generate functioning network instances. For this purpose, we have devised the Network Generating Attribute Grammar Encoding (NGAGE) system[5,6].

Attribute grammars[7] consist of a context-free grammar (CFG) base in which the productions are supplemented with the ability to compute values of both synthesized and inherited attributes which may be associated with the terminal and non-terminal symbols. The use of such attributes extends the representational power of the context-free grammar. NGAGE uses the context-free component of the attribute grammar to specify the structure and sequence of signal passing within a neural network. Figure 1 depicts the context-free part of the grammar that we used to generate backpropagation networks for the character recognition task, and Figure 2 the meanings of the symbols used in the grammar.

**S → i o f B**
*Specifies a network's external I/O ports.*
**B → C N**
*Adds a control structure to a topology of basic processing nodes.*
**C → s t z**
*Specifies a topology of control nodes.*
**N → U T M**
*Ensures that the backpropagation module has the correctly sized input and output layers*
**M → L M | L**
*Specifies the number of hidden layers.*
**T → T L | T**
*Specifies the size of a layer.*
**T → n**
*Specifies an unconnected set of backpropagation nodes.*
**U → m**
*Specifies an unconnected set of simple, unweighted pass-through nodes.*

Figure 1: Context-free grammar for back-propagation networks

| | |
|---|---|
| **S** | Start symbol: Describes a complete network. |
| **B** | Represents a functioning back-propagation module. |
| **C** | Represents a topology of control neurons that implement back-propagation control. |
| **N** | Represents a topology of processing neurons |
| **M** | Represents a topology of 'hidden' processing neurons |
| **L** | Represents a layer of 'hidden' processing neurons |
| **T,U** | Represent a set of neurons |
| **i**,**o**,**f** | Externally accessible input, output and feedback ports, respectively. |
| **n** | A neuron which computes 'A' signals until it receives an 'S' signal; computes 'F' signals only when it receives an 'S' signal and until it receives a 'Z' signal; and modifies internal weights only when it receives a 'Z' signal. |
| **m** | A neuron that simply passes on the single 'A' signal it receives. |
| **s** | A neuron which outputs an 'S' signal if none of its incoming 'A' signals have changed since the previous cycle. |
| **t** | A neuron which outputs a 'T' signal if none of its incoming 'F' signals have changed since the previous cycle. |
| **z** | A neuron which outputs a 'Z' signal if it receives both an 'S' and 'T' signal. |

Figure 2: Description of grammar symbols

In our grammar, the non-terminal symbols generally represent different aspects of the neural network structure, while the terminal symbols usually represent specific neurons. Straightforward changes in the depicted class of neural network architectures can be accomplished with simple changes in the productions of the grammar. For example, the grammar of Figure 1 specifies all backpropagation networks with at least one hidden layer. The production **M → L M** is applied once for each additional hidden layer. The class of networks described by the grammar could be expanded to include the class of perceptron-like networks with no hidden layer simply by adding the production **N → U T.**

The full specification of a class of neural network architectures requires more specific constraints than are available in the context-free component. For example, one may wish to specifically restrict the number of hidden layers. Such semantic restrictions are not generally able to be included within context-free grammars, and require either the use of context-sensitive grammars or the addition of a semantic processing capability. Within our NGAGE system, this capability is provided by extending the context-free grammar to be an attribute grammar. For example consider the addition of attributes to the production **M→ L M** as shown in Figure 3. As productions are applied, individual non-terminals in the resulting parse tree are assigned attributes as specified in the production. Multiple occurrences of a particular non-terminal in a production

requires the use of subscripts to distinguish them. In Figure 3, an inherited attribute *max-layers* is previously associated with the **M** non-terminal from the left hand side of the production. This value has been inherited through from the root node of the grammar, where the original restriction was imposed. Whenever a new instance of **M** is created with this production, it inherits a decremented value of *max-layers*, to a minimum of 0. Thus each instance of **M** has a value code ranging from 0 to the original value of *max-layers*. Each layer that has a non-zero code is initialized with the inherited value of the attribute *max-size*, permitting their construction into a layer of neurons. In a similar way, the inherited attribute *max-size* is used to restrict the number of nodes that may appear in any particular level.

**$M_1$→ L $M_2$**
(inherited)
**L**.*max_size* = $\mathbf{M_1}$.*max_size*
$\mathbf{M_2}$.*max_layers* = ***max***((**$M_1$**.*max_layers* - 1), 0)
$\mathbf{M_2}$.*max_size* = **if** $\mathbf{M_2}$.*max_layers* > 0, **then** $\mathbf{M_1}$.*max_size*
**else** 0

**M→ L**
(inherited)
**L**.*max_size* = **M**.*max_size*

Figure 3: Use of attributes to limit number of layers

**$M_1$→ L $M_2$**
(synthesized)
$\mathbf{M_1}$.*in_nodes* = **L**.*all_nodes*
$\mathbf{M_1}$.*out_nodes* = **if** $\mathbf{M_2}$.*out_nodes* non-empty, **then**
$\mathbf{M_2}$.*out_nodes* **else** **L**.*all_nodes*
$\mathbf{M_1}$.*all_nodes* = **L**.*all_nodes* ∪ $\mathbf{M_2}$.*all_nodes* ∪
***sat_nodes*** (**L**.*all_nodes* , $\mathbf{M_2}$.*in_nodes*)
$\mathbf{M_1}$.*connections* = $\mathbf{M_2}$.*connections* ∪
***sat_connect***(**L**.*all_nodes* , $\mathbf{M_2}$.*in_nodes*)

Figure 4: Use of attributes to compute connectivity

Once the parse tree is constructed, and the inherited attributes evaluated, the layout of the backpropagation network's neurons will be complete. In order to become a complete specification it is necessary to designate the connections that will be present in the network. This is accomplished through the addition of synthesized attributes. Synthesized attributes work in the opposite manner to inherited attributes, computing values based on the values of attributes on the right hand side of the production, and assigning them to symbols on the production's left hand side. The attribute evaluations in Figure 4 collect information about the network nodes and connections from lower levels in the tree and combine them with connections created within the production, and passing on that specification of connections as attributes of the **M** non-terminal on the left hand side of **M→ L M**. Through a continuation of this same process, the information about all the connections is eventually accumulated at the root node of the grammar.

Figure 5 shows the entire attribute grammar for the class of backpropagation networks used in the experiments described in this paper[8]. In order to construct a parse tree representation for an instance network, the productions of the grammar are applied, then the inherited attributes are computed in order to impose the semantic restrictions that are summarized in the initial values of the inherited attributes associated with the starting symbol of the grammar. Next, the synthesized attributes are computed, resulting in a complete description of the connections and nodes of the network encoded in the final value of the synthesized attributes of the starting symbol. There has been extensive research into the possible strategies for efficiently computing the attribute values with a parse tree constructed from the base context-free grammar[9].

Once the parse tree is has all of its attributes computed, it is a complete neural network specification. It can then be interpreted by a process that will carry out the network's processing. The interpreter has no knowledge of operations of backpropagation. The interpreter only contains knowledge of the models of processing for individual neurons. All the neurons and connections necessary for feedforward operation, for backpropagation of error signals, and for sequencing of firing of the layers of the network are encoded in the specification produced as parse trees by the NGAGE system.

**S → i f o B**
(inherited)
**B**.*in_size* = 16
**B**.*out_size* = 26
**B**.*max_layers* = 3
**B**.*max_size* = 30
**B**.*increment_size* = 5
(synthesized)
**S**.*all_nodes* = **B**.*all_nodes* ∪ **S**.*in_nodes* ∪ **S**.*feedback_nodes* ∪ **S**.*out_nodes*
**S**.*in_nodes* = ***node***(i) × 16
**S**.*feedback_nodes* = ***node***(f) × 26
**S**.*out_nodes* = ***node***(o) × 26
**S**.*visible_controls* = **N**.*visible_controls*
**S**.*connections* = **N**.*connections* ∪ ***1to1connect***(**S**.*in_nodes*, **N**.*in_nodes,'A'*) ∪ ***1to1connect***(**S**.*feedback_nodes*, **N**.*out_nodes, 'F'*) ∪ ***1to1connect***(**N**.*out_nodes*, **S**.*out_nodes, 'A'*)

**B → C N**
(inherited)
**C**.*nodes_to_control* = **N**.*all_nodes*
**N**.*in_size* = **B**.*in_size*
**N**.*out_size* = **B**.*out_size*
**N**.*max_layers* = **B**.*max_layers*
**N**.*max_size* = **B**.*max_size*
**N**.*increment_size* = **B**.*increment_size*
(synthesized)
**B**.*all_nodes* = **N**.*all_nodes* ∪ **C**.*new_nodes*
**B**.*in_nodes* = **N**.*in_nodes*
**B**.*out_nodes* = **N**.*out_nodes*
**B**.*visible_controls* = **C**.*final_controls*
**B**.*connections* = **N**.*connections* ∪ **C**.*connections*

**N → U T M**
(inherited)
**U**.*size* = **N**.*in_size*
**T**.*size* = **N**.*out_size*
**M**.*max_layers* = **N**.*max_layers*
**M**.*max_size* = **N**.*max_size*
**M**.*increment_size* = **N**.*increment_size*
(synthesized)
**N**.*all_nodes* = **U**.*all_nodes* ∪ **T**.*all_nodes* ∪ **M**.*all_nodes* ∪ ***sat_nodes***(**U**.*all_nodes*, **M**.*in_nodes*) ∪ ***sat_nodes***(**M**.*out_nodes*, **T**.*all_nodes*)
**N**.*in_nodes* = **U**.*all_nodes*
**N**.*out_nodes* = **T**.*all_nodes*
**N**.*connections* = **N**.*connections* ∪ ***sat_connect***(**U**.*all_nodes*, **M**.*in_nodes*) ∪ ***sat_connect***(**M**.*out_nodes*, **T**.*all_nodes*)

**$M_1$ → L $M_2$**
(inherited)
**L**.*max_size* = $\mathbf{M_1}$.*max_size*
**L**.*increment_size* = $\mathbf{M_1}$.*increment_size*
$\mathbf{M_2}$.*max_layers* = ***max***(($\mathbf{M_1}$.*max_layers* - 1), 0)
$\mathbf{M_2}$.*max_size* = **if** $\mathbf{M_2}$.*max_layers* > 0, **then** $\mathbf{M_1}$.*max_size* **else** 0
$\mathbf{M_2}$.*increment_size* = $\mathbf{M_1}$.*increment_size*
(synthesized)
$\mathbf{M_1}$.*in_nodes* = **L**.*all_nodes*
$\mathbf{M_1}$.*out_nodes* = **if** $\mathbf{M_2}$.*out_nodes* non-empty, **then** $\mathbf{M_2}$.*out_nodes* **else** **L**.*all_nodes*
$\mathbf{M_1}$.*all_nodes* = **L**.*all_nodes* ∪ $\mathbf{M_2}$.*all_nodes* ∪ ***sat_nodes*** (**L**.*all_nodes* , $\mathbf{M_2}$.*in_nodes*)
$\mathbf{M_1}$.*connections* = $\mathbf{M_2}$.*connections* ∪ ***sat_connect***(**L**.*all_nodes* , $\mathbf{M_2}$.*in_nodes*)

**M→ L**
(inherited)
**L**.*max_size* = **M**.*max_size*
**L**.*increment_size* = **M**.*increment_size*
(synthesized)
**M**.*in_nodes* = **L**.*all_nodes*
**M**.*out_nodes* = **L**.*all_nodes*
**M**.*all_nodes* = **L**.*all_nodes*
**M**.*connections* = {}

**$L_1$→ T $L_2$**
(inherited)
**T**.*size* = ***min***($\mathbf{L_1}$.max_*size*, $\mathbf{L_1}$.*increment_size*)
$\mathbf{L_2}$.*max_size* = ***max***(($\mathbf{L_1}$.*max_size* - **T**.*size*), 0)
$\mathbf{L_2}$.*increment_size* = $\mathbf{L_1}$.*increment_size*
(synthesized)
$\mathbf{L_1}$.*actual_size* = **T**.*size* + $\mathbf{L_2}$.*actual_size*
$\mathbf{L_1}$.*all_nodes* = **T**.*all_nodes* ∪ $\mathbf{L_2}$.*all_nodes*

**L→ T**
(inherited)
**T**.*size* = ***min***(**L**.max_*size*, **L**.*increment_size*)
(synthesized)
**L**.*actual_size* = **T**.*size*
**L**.*all_nodes* = **T**.*all_nodes*

**T→ n**
(synthesized)
**T**.*all_nodes* = ***node***(n) × **T**.*size*

**U→ m**
(synthesized)
**U**.*all_nodes* = ***node***(m) × **U**.*size*

**C → s t z**
(synthesized)
**C**.*final_controls* = **C**.*new_nodes*
**C**.*new_nodes* = {***node***(s), ***node***(t), ***node***(z)}
**C**.*connections* = ***control_connect***(**C**.*new_nodes*, **C**.*nodes_to_control*)

Figure 5: Attribute grammar for back-propagation networks

## 3. GENETIC MANIPULATION OF NEURAL NETWORK PARSE TREES

The genetic operators such as mutation and crossover cannot reasonably be applied directly to a neural network structure, for example replacing randomly selected neurons with other neurons. To apply such operators in this way would be to disrupt the network's functionality that led it to the performance level that marked it for selection. The genetic operators must manipulate meaningful and identifiable components of the network's functionality in order to produce a subsequent improved population. The individual non-terminal nodes of parse tree associated with each network generated by NGAGE provide identified replacement and substitution points that retain the network's functionality. Figure 6 depicts an example of a parse tree generated from the grammar of Figure 5. The subtree in the dotted square represents the portion of the parse tree that corresponds to a topology of processing nodes. Since the grammar does not provide multiple productions expanding the **S**, **B**, **C**, **N**, **T** and **U** symbols, the dotted subtree is the only portion that may vary from tree to tree. Genetic manipulations upon the **M** and **L** symbols of that subtree correspond to meaningful changes in the number and size of hidden layers in the network, respectively.

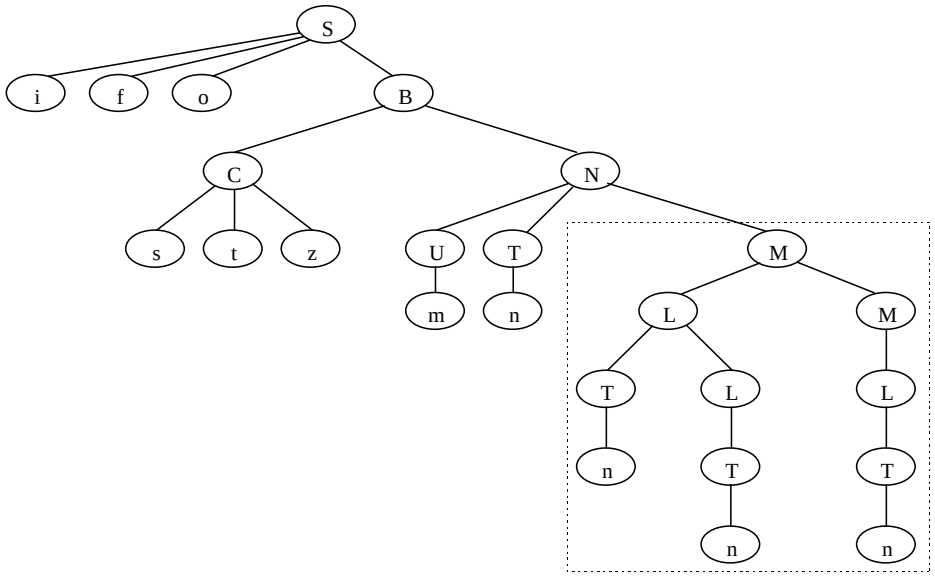


Figure 6: Sample parse tree of grammar

Consider first the issue of generating a population of neural networks from the attribute grammar specification. A representative population can be easily obtained using the simple strategy of starting with the root node of the grammar as the sentential form and expanding each non-terminal in the sentential form using equal probabilities for the alternative applicable productions. It is interesting that individual choices in the grammar can be assigned probabilities that will influence the population in predictable ways. For example, the productions **M → L M** and **M → L** are used to create the hidden layers. If each time the non-terminal **M** is to be expanded, these two productions are used with probabilities 0.7 and 0.3, then less than 10% of the network population will have more than two hidden layers, a condition that would likely be desirable in a population of backpropagation networks.

Next, consider the genetic manipulation of the members of the population. The use of the parse tree as a genetic code offers several advantages. Firstly, it provides a strong typing regime for genetic manipulations[10,11]. The genetic operators that are used must be defined to offer guarantees about the scope of the populations. Secondly, operations involving some particular non-terminal symbol may be more effective than others in generating viable results from genetic operations. It is possible to keep track of the results of applying genetic operators to each of the non-terminals, and use a reinforcement learning scheme to influence the probability of applying future operations to that symbol. This may favor the types of neural structures that are the most successful. As a simple example, in a particular application the number of hidden layers may not greatly influence performance, but the size of the layers might. In such case, the non-terminals which lead to variation in layer size will be favored for genetic manipulation. Thirdly, from each node in the parse tree, it is also possible to determine its role in the overall structure of the network by examining such characteristics as how far the node is from the root node. This offers the opportunity to tune the application of the genetic algorithm to favor either large or small scale

modifications. The scale of the favored modifications may vary through the course of the genetic search.

We have defined two genetic operators that take all of these ideas into account. Each symbol in the grammar is assumed to have an initial probability of selection. Subtree crossover is defined as follows. Given two parent parse trees, all the non-terminal symbols that exist in both trees are extracted. Those symbols that have a non-zero probability of selection are considered in a random, weighted selection. The result is the selection of a non-terminal symbol from the grammar that exists in both trees and is a valid crossover point. Then, from each tree, the set of nodes with matching symbols are extracted, along with their depths in the tree. Finally, a random selection of one of those nodes is made for each tree. This selection may be uniform, biased directly according to depth (i.e., those that are deeper are more likely), or biased according to inverse depth (i.e., those that are higher in the tree are more likely). The subtrees rooted by the chosen node in each tree are swapped to produce the offspring. Since crossover can only occur at subtrees which have the same root symbol, this crossover operator guarantees that the two newly created examples could have been generated from the defining grammar. Figure 7 illustrates an example of subtree crossover. The two dotted subtrees in the parents, which have the same non-terminal symbol, are exchanged to produce two new offspring. Note that the hidden layer sizes of the offspring are different than those of the parents.

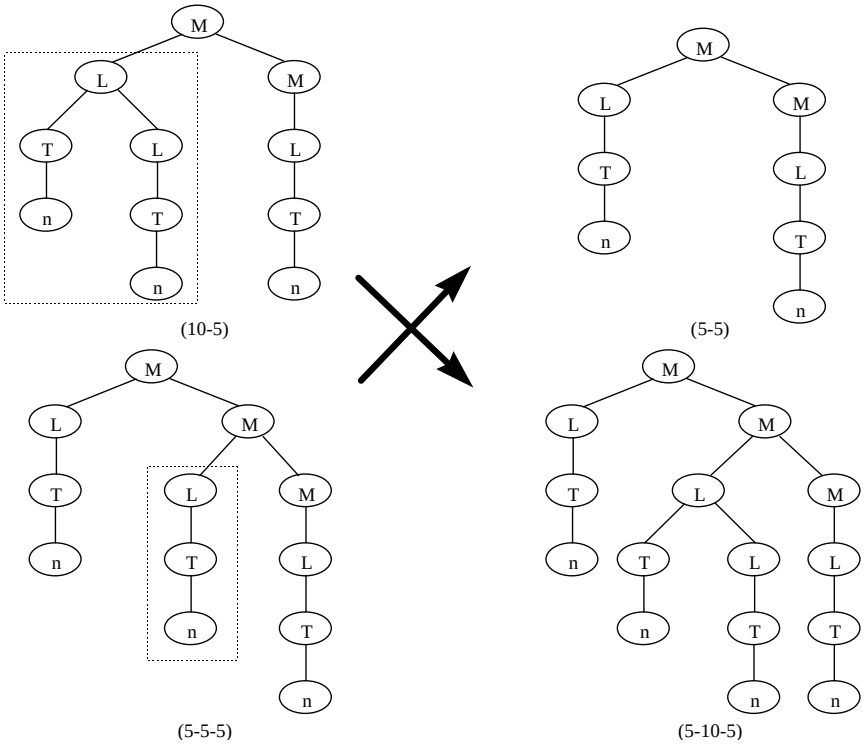


Figure 7: Illustration of subtree crossover applied to parse trees

The subtree mutation operator is defined similarly (see Figure 8). A node in the single parent tree is selected using the same procedure of selecting a non-terminal symbol first and a matching node next, based upon depth. The selected subtree is then replaced by a new tree created using the same generation process that was used to create the population, but with the root node in the generation set to the selected non-terminal. This guarantees that the mutated network falls within the class of networks that is described by the grammar. Figure 8 illustrates an example of subtree mutation. The dotted subtree in the parent is replaced by a new subtree randomly generated from the grammar and rooted by the same symbol. Note that the resulting offspring has a different number of hidden layers than its parent.

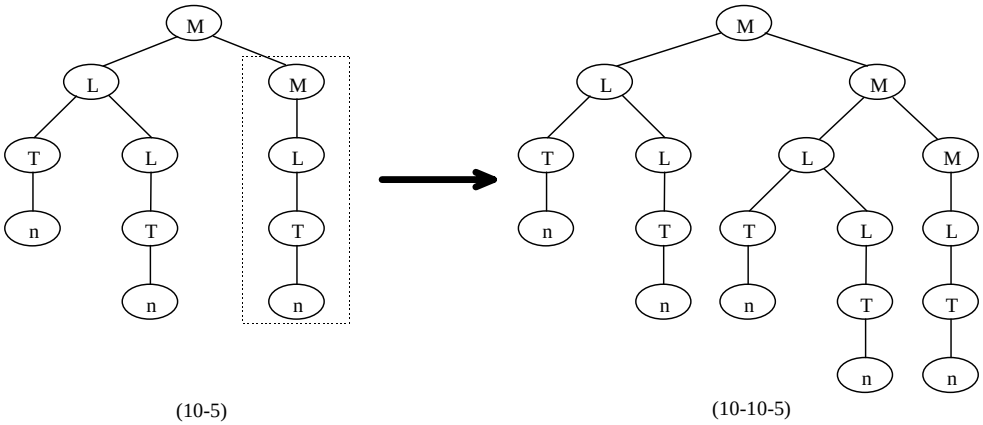


Figure 8: Illustration of subtree mutation applied to parse trees

## 4. GENETIC MANIPULATION OF BACKPROPAGATION NETWORKS

### 4.1. Experimental Paradigm

All experiments were carried out using the same NGAGE grammar and the same set of initial attributes. Specifically, the grammar of Figure 5 was used with root attribute values of 3 for *max_layers*, 30 for *max_size* and 5 for *increment_size*. All rules were assigned equal probability of being chosen. The space of networks created by this grammar is that of back-propagation networks with one to three layers of hidden nodes and up to 30 nodes in each layer, in increments of 5 nodes. A total of 258 networks that satisfy these conditions are possible.

The data set used is the 'Letter Recognition Database' available from the Machine Learning Repository at the University of California, Irvine. This data set has been provided as a benchmark set for learning algorithms[12]. It consists of 20000 total exemplars of the 26 letters of the English alphabet. The letters were derived from 20 fonts that were randomly distorted to form black-and-white images. Each image was analyzed to produce an exemplar of 16 numerical attributes such as the position and dimensions of the bounding box and the number of black pixels. Previous results[12] have achieved a testing performance rate of 82.7% using Holland-style adaptive classifier systems that were trained on 16000 exemplars and tested on 4000. In our experiments, the data was randomly split into 3 sets: a training set of 16000 exemplars, a testing set of 2000 exemplars, and a hold out set of 2000 exemplars. The same sets were used in all experiments.

To speed up the genetic experiments and avoid unnecessary re-computations, a database of trained instances of all possible networks was first created. The learning rate of all networks was 0.05. Each network was trained for 200 epochs on the training data, with the data presented in random order in each epoch. At the end of 200 epochs of training, the performance of the network on the testing set was determined. This value, in terms of the percentage of letters correctly classified, was then used as the fitness value for that network configuration and stored in the database. Each network configuration was trained and tested only once.

The evolutionary algorithm used was a fixed-population size, steady state genetic programming algorithm, with fitness proportional roulette wheel selection, subtree mutation and crossover operators (as described above), and worst-one-out competitive replacement. In other words, during each generation, a single genetic operator is first selected randomly. Each operator may have a different probability of occurring. We used a mutation rate of 10% and a crossover rate of 90%. The parents to which the operator is applied are selected randomly in proportion to their fitness values. Those that are most fit are most likely to be parents. In our algorithm, we did not permit the same individual to be both parents for crossover. After application of the genetic operator, the fitness of the resulting offspring is determined. In the case of mutation, that fitness is then compared to that of the least fit individual in the population. If the new individual's fitness is better, then it replaces that worst individual. Otherwise, no change is made to the population and the algorithm proceeds to the next generation. In the case of crossover, the fitnesses of the two resulting offspring are compared to those of the two worst individuals in the population. Only the best two out of the four individuals survive to the next generation. Note that this will

result in a monotonically increasing population fitness.

In the application of the genetic operators, symbol reinforcement was an option. When reinforcement was not used, the M and L symbols both had a likelihood of 0.5 of being selected. When reinforcement was used, the M and L symbols started with a likelihood of 0.5, but that value was modified after each genetic operation. An improvement in fitness of the offspring over the parents led to an increase in the probability of selection (e.g., 0.5 to 0.55) while a fall in performance led to decrease in the probability (e.g., 0.5 to 0.45).

### 4.2. Experimental Conditions

Six experimental conditions were carried out using the genetic program. The only variation between conditions was whether symbol probability reinforcement was used or not, and whether depth bias was uniform, direct or indirect. In each condition, a population size of 10 was used and the algorithm was run for 100 generations. This ensured that the genetic algorithm could not exhaustively search the entire space. At most, it could search 200 individuals, with a high likelihood of overlap among those individuals. After 100 generations, the hold-out performance of the most fit individual was evaluated. This configuration was run 5 times for each experimental condition, with a different initial population each time. Note that each condition started with the same 5 initial populations.

As a performance check, a final set of 5 runs was made. The average number of novel individuals examined over all the genetic programming runs was calculated. Five runs were then made in which this number of different individuals were directly generated randomly using the grammar. The hold-out performance of the individual with the best fitness score in each run was then evaluated.

### 4.3. Results

The results for each experimental condition are summarized in Table 1. The average number of novel individuals examined in the genetic runs was 48, and this value was used in the random runs. The last two rows of the table present results that are computed based upon our knowledge of the fitnesses for the entire space of networks. Based on the information in our database, we were able to rank every network configuration according to its fitness. A rank of 1 was assigned to the network with the optimal fitness, 2 to the second best network, and so on. Possible rank values ranged from 1 to 223, since networks with the same fitness were assigned the same rank. The network with the highest fitness value was one with a hidden layering of (30-25-25), which had a fitness of 82.0%. The networks had a minimum fitness of 17.4% and average fitness of 63.2%. Inspection of the fitness values stored in the database also revealed that the network structures that performed well tended to have large first hidden layers.

| | Without Symbol Reinforcement | | | With Symbol Reinforcement | | | Random |
|---|---|---|---|---|---|---|---|
| | Uniform | Direct | Inverse | Uniform | Direct | Inverse | |
| Average number of novel individuals per run | 52 | 45 | 47 | 47 | 45 | 53 | 48 |
| Average number of runs until best solution first discovered | 65 | 43 | 39 | 52 | 50 | 53 | X |
| Average fitness of best individual in final population | 81.4 | 80.2 | 80.5 | 80.5 | 80.6 | 81.8 | 79.1 |
| Maximum hold-out performance over all runs | 82.0 | 81.9 | 82.0 | 82.0 | 82.0 | 82.0 | 80.0 |
| Average probability for symbol M | X | X | X | 0.81 | 0.81 | 0.79 | X |
| Average probability for symbol L | X | X | X | 0.19 | 0.19 | 0.21 | X |
| Average rank of best individual in final population | 3.4 | 12.8 | 11.4 | 11.2 | 11 | 1.6 | 18.8 |
| Number of runs optimally fit individual found | 1 | 1 | 1 | 1 | 2 | 2 | 0 |

Table 1: Experimental Results

### 4.4. Discussion

Overall, the results demonstrate that NGAGE grammars can be effectively used to specify and genetically search a space of back-propagation network architectures. At first glance, the performance on all runs seem rather close on average. The average fitness and maximum hold-out performance in particular show very little difference between experimental conditions. However, the close results are somewhat misleading since the top 50 networks in the space are all with 6% of the optimal 82.0% fitness. More revealing is an inspection of the average rank of the best individuals found. This value indicates how good the discovered solutions were in terms of the entire space of possible networks. The results clearly show that all the genetic runs were much better than the random runs. Further, the condition that is the clear winner is symbol reinforcement with inverse depth bias.

The results also show that the symbol probability values behaved quite similarly in all reinforcement conditions. They indicate a marked preference to making changes to entire layers (i.e., symbol **M**) rather than within layers (i.e., symbol **L**). The reinforcement conditions also showed a greater tendency to converge upon the optimal solution (33% of all runs versus 20%). These results indicate that the use of symbol reinforcement has promise. The suggestion is that symbol reinforcement permits the genetic algorithm to focus upon those types of structural changes that most improve fitness. Further testing on a larger problem space is required to make a stronger conclusion. An analysis of how the probabilities change over the generations, and what effect they have on biasing the genetic search is also required.

The variation in the depth bias, on the contrary, seems to have had no clear effect other than that the genetic runs that used no depth bias generally took longer to discover their best solution. This may indicate that the use of a depth bias can help speed up the genetic search. However, it may alternatively suggest that it leads to a tendency to converge prematurely, thereby trapping the genetic search in a local minimum. Finally, although the focus of our experiment has not been to emulate previous results[12], our best network did achieve a comparable hold-out rate of 82.0%.

## ACKNOWLEDGMENTS

The research reported in this paper was conducted with financial support from the Natural Science and Engineering Research Council of Canada.

## REFERENCES

1. X. Yao, “Evolutionary artificial neural networks,” *International Journal of Neural Systems*, **4**, p. 203-222, 1993.
2. H. Kitano, “Designing a neural network using genetic algorithm with graph generation system,” *Complex Systems*, **4**, p.461-476, 1990.
3. F. Gruau,. “Automatic definition of modular neural networks,” *Adaptive Behavior*, **3**, p. 151-183, 1995.
4. J.R. Koza, *Genetic Programming.* Cambridge, Mass: MIT Press, 1992.
5. T.S. Hussain and R.A. Browse, “Network generating attribute grammar encoding,” *1998 IEEE International Joint Conference on Neural Networks*, **1**, p. 431-436, 1998.
6. T.S. Hussain and R.A. Browse, “Including control architecture in attribute grammar specifications of feedforward neural networks,” *1998 Joint Conference on Information Sciences: Second International Workshop on Frontiers in Evolutionary Algorithms*, **2**, p. 432-436, 1998.
7. D.E. Knuth, “The semantics of context-free languages," *Mathematical Systems Theory*, **2**, p. 127-145, 1968.
8. R. Hecht-Nielsen, *Neurocomputing*. Reading, Mass.: Addison-Wesley, 1990.
9. G.V. Bochmann, “Semantic evaluation from left to right,” *Communications of the ACM*, **19**, p. 55-62, 1976.
10. D.J. Montana, “Strongly Typed Genetic Programming,” *BBN Tech Report #7866*, 1993.
11. T.D. Haynes, D.A. Schoenefeld and R.L. Wainwright, “Type inheritance in strongly typed genetic programming,” Chapter 18 in K.E. Kinnear, Jr. and P.J. Angeline (Eds.), *Advances in Genetic Programming 2*. Cambridge, Mass: MIT Press. 1996.
12. P.W. Frey and D.J. Slate, “Letter Recognition Using Holland-style Adaptive Classifiers,” *Machine Learning*, **6**, p. 161-182, 1991.