Abstract

Evolution of artificial neural networks (ANNs) has been investigated for a decade or so with a wide variety of approaches seen in the literature. Application of evolutionary methods to the ANN design problem has been shown to produce good results and a number of systems have been proposed that enable the automatic discovery of suitable networks for the solution of test problems. Most systems so far described apply the evolutionary method to only part of the design, or they are otherwise limited in the types of network they can produce. This paper proposes a more comprehensive evolution of network design, enabling all aspects of the network to evolve and all varieties of networks to be discovered. Early testing of this system shows the possibilities of this method, generating commonly recognized solutions to benchmark test problems as well as discovering novel designs. Further experimentation and expansions of the system are also described.

1. Introduction

Neural networks have proved to be useful tools in a wide variety of problems that are difficult to solve (e.g. combinatorial optimization, classification, prediction etc) using traditional methods. The choice of which type of network to use to solve the problem at hand is not always straightforward and often requires a degree of experience or the application of heuristics or ‘rules of thumb’. Likewise the specific design of the chosen network which will most efficiently and/or successfully tackle the problem is often far from easy to obtain. This conundrum of network type and design choice can be formulated as search through network-architecture space, a space which can be vast if few limitations are placed on the permitted topologies and connectivities. When viewed in this way, it seems attractive to use evolutionary methods to better conduct this search, as these approaches have been shown to be very effective as global search or optimization techniques (Goldberg, 1989).

Many different evolutionary approaches have been proposed and most, if not all, have reported some advantage over the alternatives to evolutionary design (i.e. experience-based choice, trial and error etc). As well as architecture design, evolutionary methods have also been applied to the learning algorithm of the networks (most often evolving the weights on connections) in order to alleviate the perceived disadvantages of backpropagation and other gradient-based learning rules. Perhaps most usefully the evolutionary method has been used to optimize the entire network (connectivity, weights, size etc), which enables the evaluation of evolved networks to be done with less noise attributed to a pre-defined learning rule and/or architectural constraint.

This paper describes a system in which the networks are encoded such that all components of the network may be acted on by the evolutionary process. Hence, neural networks of any connectivity, including recurrent connections may be generated and each node may have any one of a number of transfer functions associated with it. Also described is a mechanism for introducing the possibility of discovering modular neural networks by including a replicating operator in the network representation. This operator results in parts of the encoding being expanded following prescribed rules, creating a larger network. Inclusion of this functionality defines representations using this feature as indirect encodings (Yao, 1999).

Two evolutionary methods are used to search the possible networks described by the encoding scheme. A traditional genetic algorithm (GA) approach is used where a population of networks is evaluated for fitness before undergoing selection, crossover and mutation. Custom designed crossover and mutation operators are applied, each designed to allow a full search of the combinations whilst favouring the generation of smaller networks. The much debated problems of using the recombination operator (crossover) (e.g. Angeline et al. 1994) lead many to eschew its use and favour alternative methods that avoid it (e.g. evolutionary programming, EP). EP is similar to the steady state GA (SSGA) in which only a small portion of the population is replaced at each generation, resulting in the more rapid selection of fitter
individuals (Syswerda, 1991), however, EP does not use a crossover operator. This type of approach is also attempted here along with a steady state GA that does employ recombination.

Accompanying the genetic operators is a so-called ‘partial-training’ method (Yao, 1996), this feature is introduced to add a fine-grained search to the global search carried out by the main GA or EP. With each discovery of a new architecture and weight distribution, the partial training algorithm will conduct a local search starting from the discovered weights in an attempt to find an improvement in the fitness of the network. In this manner a network that is discovered close to the solution may be recognized as an acceptable design via a local weight search.

Experimentation using this system remains in the early stages, but testing using the common benchmark logic problems of exclusive-or (xor), symmetry and parity is presented. Generalization ability is thus not explored to date and no sophisticated regularization method is included in the fitness function, pruning mechanism or elsewhere. However it is hoped that the tendency to produce smaller sized networks will help in preserving generalization and avoiding overfitting problems.

2. Network Representation

The network is encoded into a representation to enable easier operation of the evolutionary process. This encoding (commonly called the chromosome in genetic algorithm work) is what is evolved in the process and the individuals – neural networks - are generated (or developed (Gruau, 1992)) from these encodings. The scheme used here is one in which each network is represented by a string of characters, with each character or group of characters representing either a node, connection, weight or instruction to repeat some or all of the preceding characters’ meaning. An example string with definitions and the resulting network is shown in figure 1.

The representation scheme is intended to permit all possible connectivities, providing the largest amount of architectural variety available for discovery. The inclusion of the recurrent symbol ‘R’ technically defines this representation as an ‘indirect encoding’ because the rules for expanding this symbol abstract some of the detail of the network away from the encoding. There is no detail of layering or other architectural relationships between groups of nodes in this encoding, each node is simply listed with the connections and associated weights that are incoming to that node. Output nodes are defined within the fitness function of the evolutionary method (the number of which is supplied by the user) and consequently no output information is contained within the network representation.

Connections are associated with a node, each node has listed after it the incoming connections. The node from which each connection emanates is defined by counting a number of nodes along the string equal to the number indicated in the connection. Connections listed for a node may indicate a connection beyond the end of the string, in this case the count is looped around such that in a 3-node string if the first node has a connection from ‘node 5’, it is actually connected from node 3. This enables nodes to
have connections from nodes that appear before them in the string and indeed to have a connection to/from themselves.

Alternative activation functions are possible using this encoding and these are represented using different node symbols. The standard sigmoid is S, the gaussian is G, a PI unit (weighted product of connections through a sigmoid (Zhang, 1994)) is P and a hard-limiter is H. Variations on these four types may be considered, but the variety available using only these four is currently considered broad enough.

The motivation behind the choice of encoding scheme was mainly one of understandability and representation potential. Also considered was the advantage of keeping those components associated with each node (i.e. the bias and inputs) local to that node in the string. This helps to preserve useful features or combinations of features when crossover occurs and so aids the effective operation of a genetic algorithm. In the literature, both indirect (e.g. Kitano, 1990) and direct encoding schemes have been widely employed, with some debate over which approach is most appropriate. Certainly the issue of scalability is crucial and larger networks clearly require very large chromosomes in a direct encoding scheme, but can be relatively compactly defined using an indirect one. However, indirect methods often impose some degree of bias on the available architectures of the network and for small sized networks a direct encoding allowing all possible combinations is considered superior. Modularity is also an issue here, the generation of complex networks from smaller, simpler networks is an advantage and indirect encodings that can facilitate this (Gruau, 1992). The final choice here represents something of a compromise, allowing the possibility to use or ignore the ‘R’ component in the encoding. This leaves the user the choice to as to whether to investigate potentially modular designs or to keep a strictly direct approach. In the experiments presented in this paper, the ‘R’ component has not been used however it is hoped that this feature will enable the discovery of interesting modular designs in the future, especially when considering the evolution of larger networks.

3. Fitness Function

Each network, represented by its encoding, must have its fitness evaluated. Firstly, an actual neural network must be generated from the encoding and this is done inside the fitness function. No information is contained within the encoding as to the general architecture of the network it represents. That is, no layering or other input-output path is provided. Only the number of inputs and outputs are supplied to the fitness function and so it is within this function that the output node(s) are defined. The output of the network for each training example is ascertained by applying the input vector and then evaluating the activation of each node in turn, continuing to do so until stability is reached or a time-out has been exceeded. The output node(s) are determined during the application of the first input vector by choosing the last node(s) to become stable as the output node(s). The output node(s) are then fixed for the remaining training examples. Unstable networks are allocated a fitness of zero.

Presently the fitness of the networks is tested without using a sophisticated fitness equation or different training, testing and validation sets. For training and testing using only the logic problems of xor, symmetry and parity, it is only required to use the entire set of possible patterns to both train and test each network. More challenging problems requiring some generalization would require a more complex fitness evaluation procedure but none is presented here.

4. Evolutionary Method and Operators

The system contains the capability to use both recombination (crossover) and a more sophisticated mutation procedure. The three types of evolutionary method used are a traditional genetic algorithm type approach, a steady-state GA and an SSGA without crossover (similar to an EP design). A general evolutionary procedure is shown in figure 2. In the GA the selected ‘mating pool’ is identical in size to the original population and replaces it after crossover and mutation have taken place. With the SSGA, only two selected chromosomes enter the mating pool, and these replace the worst in the original population, if their fitnesses exceed those of the worst members. Additionally, even if the SSGA mating pool pair are not fitter than the worst original individuals they will still replace them if they are larger (in terms of nodes), as increased size means larger potential for fitness. In this manner this method is similar to some EP approaches.

Details of the various stages are described below:

2.1 Initial Population

A population of specified size (usually 50) of network encodings is generated randomly. Nodes can be of any kind, connections can be from any node or input. The smaller the network size (max size is user-specified) the more chance that a node will have a significant number of connections from the inputs. Weights and biases are randomly assigned between -5.0 and +5.0.

2.2 Partial training

This can be either random weight replacement, where a random search is conducted for a better weight
distribution, or a more sophisticated GA training. The genetic algorithm training searches for improved weights starting from an initial population generated by perturbing the original weight distribution. A standard GA proceeds for a specified number of generations. The most fit weights being retained.

2.3 Pruning
Connected nodes are traced back from the defined output node (determined during fitness evaluation) to the inputs. Any nodes not encountered are deleted.

2.4 Diversity and Fitness Scaling
It is desirable to maintain a healthy diversity in a population and to ensure that selection consistently picks from amongst the best individuals. Measuring the diversity and appropriately adjusting the probability of mutation (adaptive mutation) and scaling the fitness before selection (roulette) helps to achieve these aims.

2.5 Selection
GA – Remainder stochastic sampling without replacement (Goldberg, 1989) – similar to roulette wheel selection but each individual is guaranteed a number of selections according to the integer part of their probability of selection. Selection is elitist as fittest individual is always selected.

SSGA – Rank-based selection

2.6 Crossover
This operator proceeds by simply swapping parts of paired strings. The crossover points are required to be at nodes, such that partial strings consist only of whole node-descriptions. Single point crossover is used, but as the strings may be of different lengths then points in both strings must be selected. When using the GA-method the fittest pair is immune to crossover (and mutation) ensuring that maximum fitness can never decrease between generations.

2.7 Mutation
Each chromosome in the mating pool may undergo mutation. There are a number of different mutations possible.

If the probability of mutation (as defined by the diversity) is satisfied the chromosome is exposed to possible mutation node by node. Each node has a 30% chance of being mutated in some way. There are three kinds of possible mutation, mutation of weights, mutation of connections and mutation of nodes.

Mutation of Weights
Each weight value (including the bias) of the node is randomly reassigned (within the same range)

Mutation of Connections
Each connection to that node has a further 30% chance of mutating by either changing its origin, being deleted or having another random connection added.

Mutating, deleting and adding are randomly selected with probabilities in the ratio 5:3:2.
Mutation of Nodes

The node mutates in one of three ways. It either has its activation function randomly changed to one of the alternatives, it can be deleted, or it can reproduce itself (is ‘split’). For the last two changes the connections of other nodes in the string are automatically altered to preserve the meaning they had before the node mutation occurred. Connections of other nodes may also need to be deleted or new ones generated.

Activation function mutation, node deletion and node addition are randomly selected with probabilities in the ratio 3:3:2.

Mutation of weights, connections or the whole node is selected randomly with probabilities in the ratio 5:3:2. The ratios of probabilities are chosen such that the mutation operator will tend to explore weight distributions before architecture changes and will prefer reducing complexity (deleting connections/nodes) to increasing it. However all combinations remain possible.

In addition to node mutations any ‘R’ genes indicating parts of the string to be reproduced may also be altered. R genes can be deleted or have their lifetimes (how many times the indicated section should be replicated) or sizes (how many previous nodes should be repeated) mutated. R genes can also be randomly created between nodes with a small probability.

These operators allow any topology to be reached from any other via crossover and/or mutation, so the whole space of possibilities may be searched. This space is very large and this requires the evolutionary method to be efficient in order to discover solution networks within acceptable time scales.

5. Experiments

The system as described has been tested using the well-known logic problems xor, symmetry and parity which are often used as benchmarks for assessing training schemes for neural networks.

Most experimental runs have been used in testing the functionality of the system and have employed the xor problem. Using this as the test problem comparisons could be drawn between the use of the different evolutionary methods. The usefulness of allowing any activation function type could also be assessed and the general performance of the system gauged. The results from this testing are seen in tables 1 and 2.

(These runs used population sizes of 20, maximum 20 generations (200 SSGA – same no. of evaluations) and 20 generations of partial training GA).

<table>
<thead>
<tr>
<th>Table 1. Results of xor runs using GA as evolutionary method, size of networks is in terms of number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sigmoid Only</td>
</tr>
<tr>
<td>% success</td>
</tr>
<tr>
<td>Smallest</td>
</tr>
<tr>
<td>Average</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2. Results of xor runs using SSGA as evolutionary method, size of networks is in terms of number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sigmoid Only</td>
</tr>
<tr>
<td>xover</td>
</tr>
<tr>
<td>% success</td>
</tr>
<tr>
<td>smallest</td>
</tr>
<tr>
<td>average</td>
</tr>
</tbody>
</table>

Although the number of runs using the xor problem was not high, conclusions may still be drawn from the results. Firstly all bar one run was successful in finding a solution and most found the solution extremely quickly. Secondly most configurations found the minimum solution of one hidden node (2 nodes in total), despite no tailoring of the fitness function to favour small sizes. This suggests that in general the system works as intended, discovering efficient solution networks. Also results suggest that the GA appeared to perform slightly better than the SSGA (with or without crossover). The GA generally found fitter solutions after a similar number of evaluations as successful SSGA runs. However the fastest runs were, as expected, SSGA runs as this method evaluates only two individuals before checking for convergence, rather than the whole population as in the standard GA. The only failed run (i.e. one that did not find a solution in time) was a SSGA run, and this occurred due to a swamping of the population with one individual. Mutation failed to rescue this population before the deadline, despite the high probability of mutation due to the adaptive mutation used. Even with diversity-preserving measures the dominance of one individual appeared to be the main drawback of the SSGA method.

Also of note is the improved performance gained when allowing all types of activation function to be available. This speeded up convergence in all cases, with a solution always found in the initial population of the GA and in less than 10 evaluations using the SSGA. Clearly this freedom allows a great many more possible solutions to be found.

The symmetry and parity problems were run less often during the testing of the system, but those runs that were made highlighted the same areas as the xor runs. Again the system was nearly always successful, but the main
problems occurred when using the SSGA to solve 4-bit parity. As with xor, the population would suffer from domination by a single individual, by comparison, the GA method always discovered a solution. Also, as with the xor, the additional freedom of activation function choice improved results. The similarity with xor is of course unsurprising since symmetry and parity are merely extensions of the xor problem.

6. Discussion
The functionality of the network encoding and genetic operator combination in this system offers the possibility to evolve a large range of networks. The ability to have networks with various activation functions seems immediately encouraging in the test results. More work on how the various activation functions are selected under which evolutionary pressures (i.e. test problems) would be interesting. Whilst the amount of testing is, so far, limited the suggestion is that the traditional GA method may outperform a steady-state GA and a crossover-free method. However, the exact construction of the method of mutation will certainly have an important influence and alternative mutation operations could very well better suit an EP-like approach. Certainly continued investigation into the evolutionary approach is appropriate although the indications are that any competing-conventions problems may not be too serious.

The benefit of a partial training method seems undoubted. Improvements were seen when using even the crudest of local searches with each new individual. Clearly though, there is a balance to strike between making an effective local search with each new evaluation and permitting the local search to become too exhaustive and time-consuming, dominating the run-time, for no extra gain.

7. Conclusions
The system presented in this paper represents the initial stages of the development of a promising tool for the design of neural networks. It offers possibilities of network variety unseen in the literature and the indirect encoding scheme can provide modularity. Testing has been limited, but results on the logic problems of xor, symmetry and parity have shown the potential of this system. Comparisons between the competing evolutionary methods have proved inconclusive but it seems apparent that the use of a recombination operator may not in fact be disadvantageous.

Speed and efficiency measurements have not been recorded whilst the system remains in its raw state but it is hoped that the system will prove its efficiency once full implementation and ‘tuning’ is complete.

8. Future Work
Complete implementation of the R-component that permits the production of modular networks is ongoing and a full testing of the system over more rigorous test problems will also be carried out. Further investigation of the appropriate evolutionary method is also planned and a parallel implementation of this method is proposed. It is envisaged that the complete system will be used to investigate how neural networks solve problems by examining which designs are favoured in various problem domains and how they are reached from initial random networks.

Acknowledgement
This work is supported by EPSRC award no. 0030545X

References


Yao, X. (1999)“Evolving Artificial Neural Networks” Proceedings of the IEEE, vol 87, no. 9, 1423-1447.