Evolving Both the Topology and Weights of Neural Networks *†

Zhengjun Pan Lishan Kang Sixiang Nie‡
Institute of Software Engineering, Wuhan University
Wuhan 430072, P.R.China
‡Department of Mathematics, Shantou University
Shantou, Guangdong 515063, P.R.China

Abstract

Evolutionary algorithms(EAs) have been applied to designing and training artificial neural networks(ANNs), independently or in various combinations with other algorithms like back propagation and simulated annealing, which can be distinguished into the evolution of connection weights, of architectures and of learning rules. In this paper, we present an evolutionary approach to designing neural networks both for their topology and connection weights. For we directly represent a candidate network as a graph with weighted edges and vertices, the genetic operators can be designed to be quite natural and effective. For the mutation operator, we introduced the temperature of an individual on which the mutation based can protect fitter individuals while enlarge the search neighborhood of unfit ones. Although we do not use an other fine-tuning procedure to train the connection weights of the networks in each generation, the experiments demonstrate that the algorithm can design an optimal network for some problems successfully and efficiently.

1 Introduction

Designing an artificial neural network (ANN) involves both learning an appropriate topology of nodes and links and training its connection weight values. This learning procedure usually requires many training runs with different choices of ANN structures and parameter settings, which can be regarded as a kind of optimization to find the best specific structure and parameter settings for the network. The parameters to be selected may include both constants, such as the rate of gradient descent, and constraints, such as ranges of initial distribution of connection weights. Once the structure and parameters have been decided, the next is then to optimize the connection weights usually by a gradient descent algorithm such as back propagation to yield the best performance of the ANN over the training data.

Many efforts have been made towards the automatic design of ANN architectures recently. Connectionist approach to this task fall into two categories, the constructive algorithm and the destructive algorithm[2][4]. Roughly speaking, a constructive algorithm starts with a simple network and adds nodes and links as warranted, while a destructive algorithm initially assumes a large network and prune off unnecessary nodes and links during training. In these methods, once an

*This work was supported in part by National Natural Science Foundation of China and National 863 High Technology Project of China.
†in: *Parallel Algorithms and Applications* , Vol.9, 299–307, 1996.

architecture has been explored and determined to be unfit, a new architecture is adopted and the old becomes topologically unreachable. Such structural hill climbing methods are susceptible to becoming trapped at structural local optima, which make the endeavor trying to optimize its connection weights to be little sense. In addition, constructive and destructive algorithms only investigate restricted topological subsets rather than the complete class of network architectures.

Evolutionary algorithms (EAs) have been applied to designing and training ANNs, independently or in various combinations with other algorithms like back propagation and simulated annealing. Yao has thoroughly reviewed the extensive literature on this aspect in [13]. He distinguishes the approaches to evolving ANNs into the evolution of connection weights, of architectures and of learning rules.

When evolving the connection weights of an ANN with a given topology, the network is encoded into either a binary or real-valued string by concatenating the connection weights one after the other. Once the representation scheme has been decided, a GA is used to evolve a population of these strings which are randomly generated initially, but the ANN architecture, which include the number of hidden layers, of hidden nodes and the links among all nodes, is predefined and fixed during the evolution. Since each individual in the population encodes an ANN with a completely specified set of weights, its evaluation is relatively simple. The specific encoding of every connection weight also makes the fitness evaluation quite easy. However, it has the disadvantage that the length of the strings grows more rapidly than the number of nodes in the network, so the dimensionality of the search space is very large. Though standard GAs are very robust and effective for global search, they are very slow in local fine-tunning once a promising region of the search space has been explored. Therefore, hybrid approaches to incorporate a local search procedure like back propagation into the evolution follow naturally. GAs can first be used to explore a good region quickly, then local search is employed to find a near optimal solution in this region.

For the evolution of network architectures, most researchers concentrate on the evolution of ANN connection topology, i.e., the number of nodes in an ANN and the links among them. In this case, the representation scheme only encodes the connection pattern of an untrained neural network, directly or indirectly as Yao called, rather than encodes the specific connection weights. The fitness of the network must then be evaluated by training its connection weight values on the data set with an extra training procedure like BP or GA itself. The advantage in comparison with the first approach is that the search space is relatively small, particularly if we only encode connection parameters by an indirect encoding scheme. However, an entire network training procedure must be carried out for each individual in the population for each generation during the evolution. This will cost a lot of computational time. Moreover,it is not wise because there is only a minor modification for the structure of an offspring network from its parents in many cases. There do are studies attempting to coevolve both the topology and connection weights under the GA framework, but the network architectures are still restricted within a predefined network and the evolution of weights is their distribution but not actual values[10].

In this paper, we directly represent a neural network by a weighted graph, and apply some specific genetic operators on them. The experiments indicate that this approach can yield efficient yet robust procedures for designing neural networks both for their topology and weights.

2 Network Model and Algorithm

2.1 Network Model

We shall in this paper concentrate on the layered feed-forward ANNs— one of the most popular models of ANNs, and we only consider the strictly layered feed-forward ANNs with full connection, i.e., the networks whose nodes within each layer are not connected and that the nodes in layer i

receive their inputs only from the nodes of layer $i-1$ and provide their outputs only to the nodes of layer $i+1$ and the nodes in two neighboring layers are all connected. The neurons in the network are linear threshold elements exhibiting the sigmoid or hard limit transfer function. Both the inputs and outputs are allowed to be continuous valued for a wide range of applications.

Given a training set of input patterns $\{X^1, X^2, \cdots, X^p\}$ and the associated target output vectors $\{T^1, T^2, \cdots, T^p\}$, the structure of a network and its connection weights as well as its threshold values (bias terms) defines the output of the network to each presented pattern. If the output vectors are m-dimensional,the least square error function giving the error between the actual output vectors $\{Y^1, Y^2, \cdots, Y^p\}$ and the target outputs is defined by

$$E(net) = \sum_{k=1}^{p} \sum_{i=1}^{m} (t_i^k - y_i^k)^2 \tag{1}$$

where t_i^k, y_i^k are the ith component of T^k and Y^k respectively.

This least square error function $E(net)$ measures the performance of the current network over the given training data set. Then the learning procedure can be regarded as an optimization process to minimize the objective function $E(net)$. If the network structure is expected to be as simple as possible, we can add an extra term $C n_p$ in the right of (1), where C is a user-defined constant and n_p is the number of network parameters.

2.2 Genetic Algorithms

Genetic algorithms (GAs) have been first introduced as a general adaptation scheme for a population in which each individual represents a feasible solution for the involved problem.[5][7] In our case, each individual in the population P is a neural network with both specific connection topology and its weight values. In each generation of the evolution, the networks are first evaluated by a user-defined fitness function $f : P \longrightarrow R$. Through some selection (selecting more fit individuals) and genetic operators, which are based on the fitness of the individuals, a new population is formed. The new generation usually has some more fit networks. After some number of generations, the best individual is the networks hopefully solving the task. The structure of a genetic algorithm can be sketched as follows:

Step 1: Randomly generate an initial population $P(0) := \{net_1, \cdots, net_n\}, t \leftarrow 0$;

Step 2: Compute the fitness of each individual of the population $P(t)$;

Step 3: Create a new population $P(t+1)$ by applying the reproduction, crossover, mutation and/or other genetic operators. These operators are applied to the individuals in the population selected with a probability based on fitness.

Step 4: While the termination criterion has not been satisfied, $t \leftarrow t+1$ and goto step 2.

Applying genetic algorithms to a specific task entails the definition of the encoding structure, the design of the genetic operators which are suitable for the representation, and then the evolution itself driven by a GA. In the case of evolving ANNs, Yao in [13] has completely discussed the strengths and weaknesses for several of these alternatives.

Our choice for the representation scheme differs from those already presented in the literature, for the reason that we emphasize mainly on the scalability of networks and the accumulation of knowledge during the evolution. We directly represent neural networks by weighted graphs, and present some specially designed genetic operators.

1) Representation Scheme :

Perhaps the most natural representation of a feed-forward neural network is a weighted graph. Each weight of the graph, which is either a connection weight or a threshold value (a bias term) of the represented ANN, is encoded directly by a real number. This makes the length of a genetic string to be much smaller than that encoded by a binary string. Moreover, the float representation, which makes it unnecessary to predefine the precision of solutions as the binary coding schemes, can improve fine-tuning of the solutions[11].

For the purpose of both evolving the topology and the connection weights, we permit the networks in a population to have different topologies, i.e., the graphs need not to be isomorphic (i.e.,they may have different number of nodes and/or links).

2)Genetic Operators :

The evolutionary process of a population in GAs is driven by genetic operators after the Darwinian principle of survival of the fittest and the naturally occurring genetic operations [5][7]. The most widely used genetic operators are reproduction, crossover and mutation.

To perform genetic operators, one must select individuals in the population to be operated on. The selection strategy is chiefly based on the fitness level of the individuals actually presented in the population. There are many different selection strategies based on fitness. The most popular is the fitness-proportionate selection. The tournament selection and the rank selection are two alternative selection strategies[1][5][6]. In this paper, we will use the fitness-proportionate selection strategy. If $E(net_i, t)$ is the value of objective function $E(net_i)$ of individual net_i at generation t, the fitness measure $f(net_i, t)$ is computed from $E(net_i)$ as follows:

$$f(net_i, t) = \frac{1}{1 + E(net_i, t)} \tag{2}$$

Then the probability that individual net_i will be selected into the next generation of the population as a parent to be operated on by genetic operators is

$$p(net_i, t) = \frac{f(net_i, t)}{\sum_{j=1}^{N} f(net_j, t)} \tag{3}$$

where N is the population size, i.e., the number of individuals in the population.

After a new population is formed by selection process, some members of the new populations undergo transformations by means of genetic operators to form new solutions (a recombination step). Because of intuitive similarities, we only employ during the recombination phase of the GA three basic operators: reproduction, crossover and mutation, which are controlled by the parameters p_r, p_c and p_m (reproduction probability, crossover probability and mutation probability), respectively.

Because of a distinct representation scheme, the genetic operators used in our case will be quite different from the traditional ones.

i) *Reproduction*: This is the standard roulette wheel reproduction operator[5]. It consists of two steps. First, a single individual is selected from the population according to the fitness-proportionate selection method based on its fitness. Second, the selected individual is copied from the current population into the new population (i.e., the new generation) without alteration. In order to get the best-so-far network, a elitist strategy (preserving the best individual of the last generation in the new generation) is used in our later experiments.

ii) *Crossover*: The crossover operation creates variation in the population by producing new networks that consist of parts taken from each parent. The crossover operator starts with two parental networks and produces two offspring for each two parents. The two parental models are also chosen from the population using the fitness-proportionate selection method. The operation begins by independently selecting several weights in each parent to be the crossover fragments. Then the two offspring are produced by swapping these fragments for each other.

iii)*Mutation*: Our mutation operator operates on one individual which is also selected by the fitness-proportionate selection method based on (normalized) fitness. It begins by selecting an individual within the population, then one of the following operations is performed with equal probability:

- deleting some nodes and/or links together with their weights in the hidden layers.
- inserting some nodes and/or links together with their weights in the hidden layers.
- randomly selecting a node or a link and changing its weight w to be $w + \triangle(b - w, T)$ or $w - \triangle(w - a, T)$ with equal probability, where $[a, b]$ is the domain of network parameters and

$$T = 1 - \frac{1}{E(net) + 1} \tag{4}$$

is called the mutation temperature of the selected individual *net* which is related to the concept of temperature in simulated annealing, and the function $\triangle(y, T)$ returns a value in the range $[0, y]$ such that the probability of $\triangle(y, T)$ being close to 0 increases as T decreases, i.e., the closer the network *net* to the expected network is, the smaller the change of *net* is. This property causes the mutation operator to search the space uniformly when the performance of *net* is "bad", and very locally when the performance of *net* is almost as good as we expect. The function $\triangle(y, T)$ used in our later experiments is

$$\triangle(y, T) = y \cdot (1 - r^{T^2}) \tag{5}$$

where r is a random number from $[0, 1]$. Figure 1 displays the value of $\triangle(1, T)$ for two temperatures. It clearly indicates the behavior of this operation. This operation is a little bit similar to the non-uniform mutation in [11], but our experiments demonstrate that it is more effective by using the temperature defined in (4), which is usually not same for networks in a population.

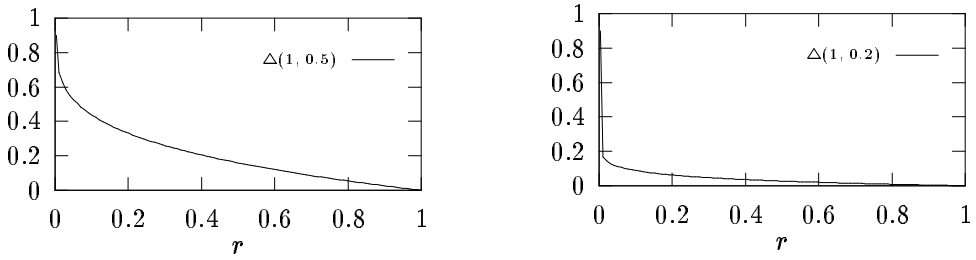

Fig 1: The value of $\triangle(1, T)$ for two temperatures.

3 Experimental Results

In this section, we shall test our proposed design technique on a 4-input and 1-output parity problem, where the output should be 1 if there is an odd number of 1s in the input pattern, 0 otherwise. The difficulty of the problem is due to the fact that input patterns differing in only one bit require opposite outputs.

As we know from Kolmogorov theorem[8], three layer perceptrons (i.e., networks with two hidden layers of processing elements) can distinguish arbitrarily complex decision regions. In our experiments, the networks have no more than two hidden layers and all outputs in the networks are continuous valued.

3.1 Parameter Settings

The algorithm are controlled by 8 parameters, which are the population size N, crossover probability p_c,mutation probability p_m,reproduction probability p_r, the maximum generation G, the maximum number of nodes in a hidden layers m, the domain of network parameters $[a, b]$, and C, a user-defined parameter for controlling network complexity.

In our later experiments, the common parameter settings are $N = 40, G = 2000, p_r = 0.1, p_c = 0.6, p_m = 0.3, [a, b] = [-10, 10]$.

3.2 Results

We carry out a set of runs to evaluate the effectiveness of the algorithm. The results are shown in Figure 2 and Figure 3. Each result was obtained through 5 random runs.

Figure 2 shows the mean performance of the 5 best networks with $C = 0.005, m = 8$. Although we do not use an other fine-tuning procedure to train the connection weights of the networks in each generation, it is demonstrated that the algorithm can still learn a fit network for the problem efficiently and quickly.

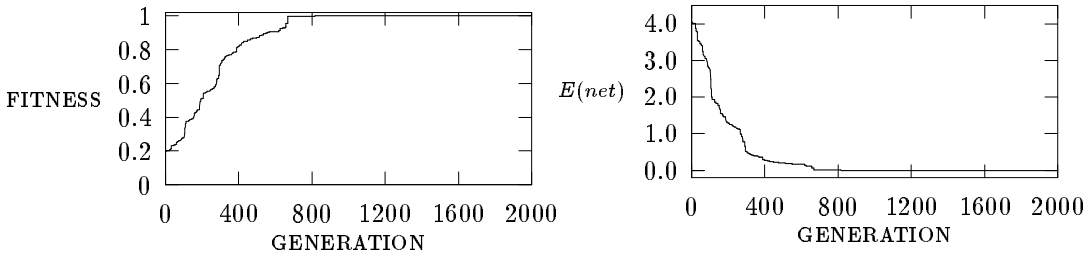

Fig 2: Mean performance of the 5 best networks ($c = 0.005, m = 8$).

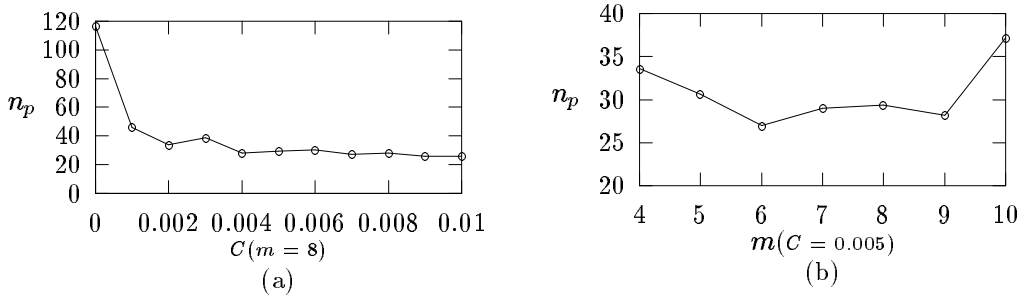

Fig 3:The influence of the parameter C and m on n_p.

In order to test the influence of the parameter C and m on network complexity (we measure it by n_p—the number of network parameters which includes the weights and biases), we also make a lot of runs with C ranging from 0 to 0.1. For the parity problem, if $C > 0.02$, then Cn_p will be a main term in $E(net)$ as evolution goes on, which can lead to a searching trend on networks with small n_p and getting stuck in local minima. Figure 3(a) shows the results with C from 0 to 0.01, we find from our experiments that if C is about 0.005, the algorithm is more stable and with little chance of early convergence. From Figure 3(b), we can conclude that there is no noteworthy influence of m on n_p. However, larger m can work out an applicable network easily but with larger computational cost.

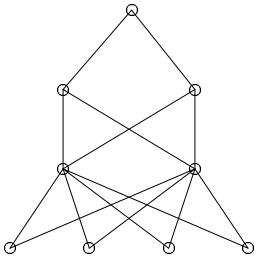

Fig 4:An optimized network topology for the parity problem.

An optimized topology for the parity problem is shown in Figure 4, which is much simpler than that in [10].
By the way, we also test the algorithm on the gasoline blending problem from [3],the XOR problem and the symmetry problem. It is indicated that our approach to designing neural networks can be used in evolving both their structures and weights successfully.

4 Conclusions

This paper presents an evolutionary approach to designing neural networks both for their topology and connection weights. Due to a special encoding procedure for candidate networks, the genetic operations can be designed to be quite natural and effective. For the mutation, we introduced the temperature of an individual on which the mutation based can protect fitter individuals while enlarge the search neighborhood of unfit ones. Although we do not use an other fine-tuning procedure to train the connection weights of the networks in each generation, it is demonstrated that the algorithm can still design a network for a problem successfully and efficiently.

References

[1] T.Bäck, Selective Pressure in Evolutionary Algorithms: A Characterization of Selection Mechanisms,Proc.of First IEEE Conf. on Evolutionary Computation, vol.1, 57–62,1994.

[2] D.Chen, C.Giles, G.Sun, H.Chen, Y.Less and M.Goudreau, Constructive learning of recurrent neural networks, IEEE International Conference on Neural Networks, vol.3,1196–1201, 1993.

[3] D.B.Fogel,L.J.Fogel and V.W.Porto, Evolving Neural Networks,Biol.Cybern.,63,487–493,1990.

[4] M.Frean, The upstart algorithm: A method for constructing and training feed-forward neural networks, Neural Computation, vol.2, 198–209, 1990.

[5] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley, 1989.

[6] D. E. Goldberg and K. Deb, A Comparative Analysis of Selection Schemes Used in Genetic Algorithms, in G. Rawlins (Ed.), Foundations of Genetic Algorithms, 69–93,1991.

[7] J.H.Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, 1975.

[8] A.N.Kolmogorov,On the representation of continuous functions of many variables by superposition of continuous functions of one function and addition,Dokl Akad VSSR 14,953–956,1957.

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

[10] V. Maniezzo, Genetic Evolution of the Topology and Weight Distribution of Neural Networks, IEEE Trans. on Neural Networks, vol.5(1), 39–53, 1994.

[11] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, Berlin, 1992.

[12] M.R.Azimi-Sadjadi, S.Sheedvash and F.O.Trujillo, Recursive dynamic node creation in multilayer neural networks, IEEE Transaction on Neural Networks, vol.4,No.2, 242–256,1993.

[13] X. Yao, A Review of Evolutionary Artificial Neural Networks, International Journal of Intelligent Systems, vol.8(4), 539–567, 1993.