1998 International Computer Symposium
Workshop on Artificial Intelligence
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

AUTOMATIC DESIGN OF NEURAL NETWORKS BASED ON GENETIC ALGORITHMS

CHEN Ching-Han

Department of Electrical Engineering
I-Shou University, Kaohsiung, Taiwan, R.O.C.
Email : neotech@ms19.hinet.net

ABSTRACT

Nowadays, the theories, models, methods and tools concerning neural networks are approaching complete and mature. Nevertheless, there still exists a main difficulty for industrial applications. That is how to design optimal network architecture according to every specific problem. The task includes optimization of network size, network topology, connection weights between neurons etc. This paper proposes an automatic design methodology of neural networks based on genetic algorithms. We analyze firstly the building blocks of neural networks in order to obtain design specifications. Then, we develop a genetic encoding method, after which the evolution process is elaborated for finding the optimal neural network. The results of our experiments reveal that our methodology is superior to the error back-propagation algorithm both for its executing efficiency and performance.

1. INTRODUCTION

There have been a number of successful cases using neural networks as an automation technology, for example, applications in the complex non-linear information processing, pattern recognition, time series prediction, intelligent control, etc. The success of those applications depends mostly on the design of the specific network architecture for specific problem rather than the learning algorithms for the networks [1]. However, an efficient and systematic method for neural network architecture design doesn't exist till nowadays. Let's take the generally applied multi-layer perceptron as an example. The topology (number of hiding layers, number of neurons and connections between neurons) before learning and the learning parameters are decided by experiences or the try-and-error method. Concerning the connection weight, it is generated by way of learning like the famous error back-propagation method. But, some of the neural network models are not governed by differential equations. The recurrent neural network is the one with which it is impossible to obtain connection weights via the error back-propagation method [1].

The aforementioned difficulty restricts the applications of neural networks. Especially, in the field of industrial automation, the developer of a neural network application system always needs to rapidly design a prototype according to the problem's particularities and its environment. The prototype has to satisfy the optimization of both performance (problem solving capability) and efficiency (executing speed). Aiming at this problem, this paper will propose a methodology using genetic algorithms (GA) in the automatic design of neural networks.

Presently, nearly all the commercial neural network development systems follow the procedure of design-evaluate-test cycle [2]. At the stage of architecture design, the network structure, the connection topology, the transfer function of neurons, the learning algorithm and the learning parameters must be determined previously. The stage of evaluation aims at the simulation of neural network and the evaluation of its performance via training data. At the final stage, the tests are carried out with testing data. If the result is not satisfactory, the architecture has to be modified, which means that a new design cycle restarts. Such design procedure implies that the designer of neural network architectures searches at random or through some heuristics among all possible network configurations for the optimal one. If a network configuration can be formalized or parameterized, the original design problem is regarded as a parameter optimization problem.

Beside other optimization techniques, the GA owns two important properties: first, it forms a global optimization method by way of stochastic search, which is different from local optimization methods like conjugate gradient method [1]; secondly, it adopts multi-agent searching strategy enabling the parallel processing performance, which is different from the single-agent searching strategy like simulated annealing method [3]. Those two properties make GA capable to find the optimal solution in the high-dimensional searching space. The sheer number of recent researches integrating GA and neural networks proves the possibility of combining these two techniques [3][4][5].

Nevertheless, there exist several obstacles in combining GA and neural networks for applications [6][7]. First of all, the applied problem must be formalized into input-output mapping model of neural networks. The possible neural network architectures corresponding to this model constitute a constrained searching space. Before seeking for the optimal neural network architecture, it is important to elaborate the encoding method, then to decide the required precision of the solution. The higher the precision is, the longer the code will become so the space and the time needed for searching will be increased.

On the contrary, if the precision is too low, the suitable solutions may probably be missed. Finally, it comes to choose or to design new genetic operators such as selection, crossover, mutation and other problem related genetic operators.

This paper is divided into five sections. In section 2, we will propose neural network building blocks, which provide systematically the neural network design specifications. In section 3, we will firstly determine the neural network/genetic encoding mapping. Then, we will define the performance evaluation function of neural networks and elaborate the evolutionary searching procedures. The section 4 demonstrates an experiment to compare the conventional approach and ours. A conclusion is given in the last section.

2. ANALYSIS OF NEURAL NETWORK BUILDING BLOCKS

The purpose of analyzing neural network building blocks is to acquire a modular and hierarchical network architecture with which we can easily modify, extend or build more complex neural network architecture. In addition, it is facile to extract design specifications from building blocks. The biggest challenge of this task comes from the assurance of the independence of every building block and the consideration of a good communication interface between building blocks in order to facilitate the design and the synthesis of complex networks.

Figure 1 shows a single neuron building block. When N incoming stimulating signals reached the neuron, they will be multiplied by N connection weights w_i, i = 1, 2,...N. The summation of the multiplication will be added up by a bias b and the result is obtained after the transfer function $f()$.

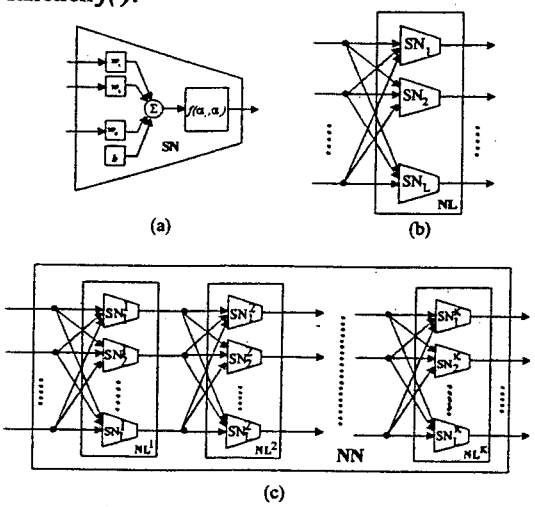

Figure 1 Neural network building blocks

We define $f()$ to be a parameterized transfer function:

$$y = f(x) = \frac{\alpha_2}{1 + e^{-\alpha_1 x}} \qquad (1)$$

where α_1 is used to modify the form of the function and α_2 its scale. x and y are respectively the input and the output. This function can be considered to be a generalized sigmoidal function. When $\alpha_1 = 1$ and $\alpha_2 = 1$, we obtain the normal sigmoidal function:

$$y = f(x) = \frac{1}{1 + e^{-x}} \qquad (2)$$

A neural layer is composed of L parallel single neurons (see Fig.1(b)). A feedforward multi-layer neural network is composed of K neural layers connected in serials. The output of each layer will become the input of the following layer (see Fig. 1(c)).

From the building blocks of the neural network, we can easily extract their design specifications. For each neuron, we have a set of design parameters P_{SN} including:

array of weights : $\{ w_i, i = 1, 2, ..., N\}$;
bias : b ;
adjustable parameters of the transfer function : α_1 以及 α_2

For each neural layer, we obtain thus a set of design parameter P_{NL}

P_{NL} : $\{ P_{SN}^j, j = 1, 2, ..., L\}$

Finally, we get a set of design parameters P_{NN} of the neural network

P_{NN} : $\{ P_{NL}^h, h = 1, 2, ..., K\}$

3. AUTOMATIC DESIGN OF NEURAL NETWORKS

The conventional design method of neural networks relies on the try and error method or on the heuristic method to determine firstly the transfer function of neural networks and the learning parameters of learning algorithms. Then, the analytical method is applied to compute connection weights between neurons, which is the so-called learning. In this paper, we use GA in the search of the optimal neural network configuration comprising the weight of each single neuron, the bias and adjusting parameters of the transfer function, in order to achieve the automatic design.

3.1 GA and the principles of the optimal search

The typical GA performs their evolution process in a binary searching space noted $\Omega = \{0,1\}^N$. GA is used to find the maximum of the performance evaluation function F in Ω. The domain is positive real :

F : $\Omega = \{0,1\}^N \rightarrow R^+$

An element of Ω is called an individual or a chromosome. Here, the chromosome is structured by a binary string with length N_b. A population Π is composed of N_{POP} chromosomes. The genetic operators of GA make the population Π_i of the i^{th} generation evolve to the offspring generation, noted population Π_{i+1}. Each generation is regarded as an iteration in GA. For discrete dynamic systems, it is a time step. The three most often used genetic operators are selection, crossover and mutation [8].

An evolution process can be described as follows:

I. Initialization of the population

Normally, individuals of the initial population Π_0 are generated at random. Every individual is assigned a value arbitrarily from {0,1}. But in certain cases, we could also take some known near-optimal solutions for the initial population.

II. Performance evaluation

It is brought out by the calculation of evaluation function for each individual.

III. Selection

By way of duplication, an identical but temporary population Π' is created from the population Π_i. The number of duplication of each individual is proportional to its performance.

IV. Crossover

We take randomly from Π' pairs of individuals for crossover operation. The individuals of each pair exchange their bit values to produce offspring which form the population Π''.

V. Mutation

The mutation operation alters randomly certain bit values of the individuals with a small probability.

VI. Production of the next generation $\Pi_{i+1} = \Pi''$

When the $(i+1)^{th}$ generation is produced, check if the evolution termination criterion is reached. If not, take Π_{i+1} for the population of i^{th} generation and go back to step II.

Genetic operators are of decisive importance for GA's performance. They possess double functionality [9] : on one hand, they permit the evolution process taking place around the better individuals (exploitation) ; on the other hand, they enable the evolution process to explore virgin domains (exploration). Among the three operators mentioned above, selection enables the current best solution to go on exploiting. Mutation permits the exploring of solutions in global searching space. Crossover is gifted with both mechanisms. It makes the individuals of the offspring generation differ largely from those of the parent generation. However, the exploration happens only in a limited region around individuals of the parent generation.

3.2 Genetic encoding of neural networks

The binary string is the most common data structure used in GA [9]. Therefore, we adopt it for neural network encoding. We enumerate design parameters of neural network in series and translate them into binary string according to the required precision, so we will get a binary encoded individual.

For a real-value parameter, the length of its binary string depends on the required precision. Assume a variable x whose value domain is $[a, b]$. If the required precision is T decimal places for x, it implies that x will have $(b-a)\times 10^T$ resolution ranges. Thus, The needed length of the binary string m can be obtained via the following formula :

$$2^{m-1} < (b-a)\times 10^T \le 2^m - 1 \qquad (3)$$

As for decoding the binary string into real-value, the formula is :

$$x = a + \text{decimal}(\text{binary\_string}) \times \frac{b-a}{2^m - 1} \qquad (4)$$

Figure 2 demonstrates how a single neuron is encoded into a binary string. In this figure, each design parameter is encoded in four bits. The total length of the binary string is $N_{SN}=(N+3)\times 4$, where N is the number of the single neuron weights. Regarding the single neuron with L layers, the total length of the binary string is $N_{NL}=L\times N_{SN}$. As for a neural network with K layers, its total length of the binary string is $N_{NN}=K\times N_{NL}$.

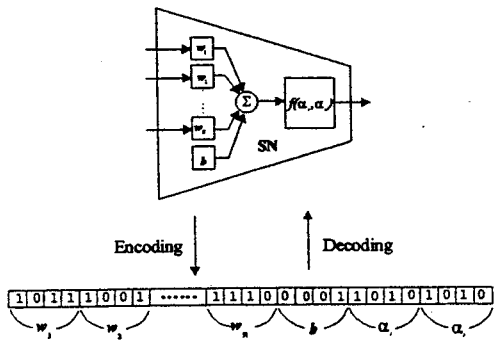

Figure 2. Encoding of a single neuron by a binary string

3.3 Evolution process of neural networks

The population is the object of GA process. From the viewpoint of the optimization, if the number of individuals constituting the population is large, it is favorable for finding the global optimum, but the convergent speed will therefore be diminished. Since our purpose is to find the optimal neural network architecture, we generate several neural networks to form a population and apply the genetic operators including selection, crossover and mutation in the population to search step by step the optimal neural network configuration.

We use a set of observed data $\{I_i, O_i, i = 1,2, ..., M\}$, where I_i is the feature vector, O_i is the associated observed output, M is the number of data records. Taking feature vectors for input and applying them in the neural network, we will get a set of inferred output $\{T_i, i =$

1,2, ..., M}. The average error function of the neural network can be expressed as :

$$err(x) = \frac{1}{2(M \times N_{out})} \sum_{i=1}^{M} \sum_{j=1}^{Nout} [O_{i,j}(x) - T_{i,j}]^2 \quad (5)$$

where $O_{i,j}$ is the j[th] observed output of the i[th] data and $T_{i,j}$ is the j[th] neural network inferred output of the i[th] data ; N_{out} is the number of output data.

We define the performance evaluation function of neural network as follows :

$$g(x)=1-err(x) \quad (6)$$

The evolution process of the population of neural networks is formalized by the following algorithms :

Step 1. Initialization of the population

a. Set the number of generations N_{GE} for terminating the evolution.
b. Set the number of neural networks N_{POP} for the population.
c. Set the probability of crossover p_{cross} and mutation p_{mut}.
d. Generate randomly N_{POP} neural networks with binary string v_i, i = 1, 2, ..., N_{POP}.

Step 2 Performance evaluation of neural networks

a. Use formula (4) to transform v_i into real-value parameter x_i , i = 1, 2, ..., N_{POP}.
b. Construct the neural networks with x_i and test them with observed data. Use formula (6) to evaluate x_i .
c. Assign the evaluation function value $g(x_i)$ to the performance fitness of the neural networks, that is fitness(v_i) = $g(x_i)$.

Step 3. Selection

Use roulette wheel method [8] to carry out the following steps :

a). Find out firstly the performance fitness of the whole population of neural networks :

$$F = \sum_{i=1}^{N_{pop}} fitness(v_i) \quad (7)$$

Then, calculate the probability of being selected for each neural network v_i:

$$p_i = \frac{fitness(v_i)}{F} \quad (8)$$

Finally, according to order of the neural networks, calculate the accumulated probability :

$$q_i = \sum_{k=1}^{i} p_k \quad (9)$$

After that, execute steps b) and c) with each neural network.

b). Generate a random number from [0, 1].

c). If $r \le q_1$, choose the first neural network ; otherwise, choose the k[th] neural network to make $q_{k-1} < r \le q_k$

Step 4 Crossover

According to the crossover probability p_{cross}, it is anticipated that crossover of $p_{cross} \times 100\%$ neural networks will take place.

a). For each neural network v_i, generate a random number r_i from [0, 1]. If $r_i \le p_{cross}$, take v_i for a parent neural network. The parent neural networks selected are matched to form pairs.

b). For each pair of neural networks v_i 和 v_k, use one-point-cut method [8] to choose randomly a position so that the portions of the two neural networks beyond this position to the right can be exchanged to the offspring v'_i and v'_k.

Step 5 Mutation

Among the whole population, $p_{mut} \times N_{NN} \times N_{POP}$ bits will be selected for a mutation, where N_{NN} is the number of bits of a neural network and N_{POP} , the number of neural networks of the population. We execute the following mutation operations :

a). Generate $N_{NN} \times N_{POP}$ random numbers r_j (j = 1, 2, ..., $N_{NN} \times N_{POP}$) from [0, 1]. If $r_j < p_{mut}$, note the position for mutation as pos = j .

b). From each mutation position pos , we are able to find its neural network i and its location bit_no in the neural network.

i = quotient(pos / N_{NN});
bit_no = remainder (pos / N_{NN});

Then, flip the bit value of the location.

Step 6 : verify the termination criterion

If the iterations reach the number N_{GE} , terminate the evolution and save the neural network with the optimal performance. Otherwise, return to step 3 and continue the evolution of the next generation.

4. EXPERIMENTATION

In order to validate the automatic design methodology of neural networks that we have proposed, we use Fisher's iris data [10] to automatically generate a feedforward multi-layer neural network. Besides, we train a back-propagation neural network with the same topology via this set of data. Finally, we compare the efficiency and the performance of these two approches.

Fisher's iris data is a set of experimental data widely used in the comparison of diverse recognition models. It has 150 records of data with three categories (each category contains 50 data). Each record of data possesses four feature values and an observed category. In our experiment, the observed category is transformed into three observed outputs. Category 1 corresponds to [1, 0, 0] ; category 2 corresponds to [0, 1, 0] ; category 3 corresponds to [0, 0, 1].

The back-propagation network adopted is an ameliorated version of the back-propagation advanced by Minia and Williams [11]. They used the learning rate adaptation and momentum adaptation techniques to accelerate the convergence of the learning algorithms and to increase the precision. Since the back-propagation network takes sigmoid function for the neurons' transfer function, we must normalize the original data to [0, 1]. The result of

momentum , $\Delta\eta$: momentum change factor per learning cycle; α : learning rate, $\Delta\alpha$: learning rate change factor per learning cycle.

Table 1. Back-propagation neural network training by iris data

Learning parameters of back-propagation neural network				Result		
η	$\Delta\eta$	α	$\Delta\alpha$	Training cycle	Training time(sec)	error(×100%)
0.2	0.998	0.2	0.999	5,000	15.4	1.43
1.0	0.995	0.5	0.990	5,000	15.4	1.45
2.5	0.990	0.8	0.985	10,000	30.8	1.35
5.0	0.985	1.0	0.980	15,000	46.2	1.34

We apply the automatic design procedures of the neural networks in the same experimental data. The result is shown in table 2. We adopt for every experiment p_{mut} =0.01 and p_{cross}=0.6 .

Table 2. Automatic design of neural networks

Evolution parameters		Optimal neural networks obtained		
N_{POP}	N_{GE}	Evolutionary time(sec)	error(×100%)	performance index(×100%)
20	50	6.2	0.78	99.22
40	100	15.6	0.58	99.42
50	50	12.4	0.65	99.35
100	100	58.3	0.53	99.47

The comparison between table 1 and table 2 reveals that our method is much better than the conventional error back-propagation algorithm both for the executive efficiency and for the performance.

5. *Conclusion*

Our automatic design methodology of neural networks is caracterized by the following virtues :

- The neural network building blocks are of modular and hierarchical structure. It is rather easy to organize and to modify them. Moreover, the transformation between neural networks and genetic encoding is linear and reversible. We can thus extend this methodology to design neural networks with different topology like recurrent neural network, bi-directional associative memory, etc.
- The single neuron uses adjustable transfer function instead of the classical fixed transfer function. In neural network, the transfer functions of neurons may be different from each other for this reason. Such kind of neural networks will be more suitable for non-linear modeling and the models obtained will be more precise than conventional neural networks.
- The α_1 and α_2 of the formula (1) are able to adjust the form and the scale of single neuron's transfer function. That's why the training data need not to go through the normalization process. In applications like simulation and prediction of dynamic systems, this architecture is obviously more appropriate than the conventional one.
- In our design methodology, the evolutionary computing replaces the analytical learning algorithms. The search is carried out by parallel and multi-directional ways and is thus capable of escaping from local optimum to find the global optimum.
- The length of genetic codes is variable, which means that the designer of neural networks can increase the number of bits according to the required precision. On the contrary, if the efficiency is considered to be an important factor, the number of bits is to be decreased. For hardware implementation of the neural networks, this functionality may be helpful in the optimization of performance/surface.

6. *Acknowledgement*

The support received from the National Science Council, Taiwan, R.O.C. under the grant No. NSC-87-2212-E-214-003, is gratefully acknowledged.

References

[1] Hecht-Nielsen, R, *Neurocomputing*. Reading, MA, Addison-Wesley, 1990.

[2] Wong, F. & Tan C., "Hybrid Neural, Genetic and Fuzzy Systems," in *Trading on the Edge* (Deboeck G. Ed.), John Wiley & Sons, Inc., 1994.

[3] Kirkpatric S., Gelatt C. and Vecchi M., "Optimization by simulated annealing", Science, Vol.220, No.4598, pp.671-680.

[4] Xin Yao, "Evolutionary Artificial Neural Networks",*Int. Journal of Neural Systems*, 1993, 4, 3, pp.203--222.

[5] Frederic Gruau, "Genetic Synthesis of Boolean Neural Networks witha Cell Rewriting Developemental Process", *Combinations of Genetic*

Algorithms and Neural Networks, pp. 55-74, IEEE Computer Society, 1992.

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

[7] G. F. Miller, P. M. Todd and S.U. Hegde, "Designing neural networks using genetic algorithms", In J. D. Schaffer, ed., *Proceedings of the Third International Conference on Genetc Algorithms*, Morgan Kaufmann, 1989.

[8] David E. Goldberg, Genetic algorithms in search, optimization and machine learning, Addison-Wesley Inc., 1989.

[9] Zbigniew Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 1994.

[10] R. Fisher, "The use of multiple measurements in taxonomic problems", *Annals of Eugenics*, vol. 7, part II, pp. 179-188, 1936.

[11] A. A. Minia and R. D. Williams, "Acceleration of back-propagation through learning rate and momentum adaptation", *Proceedings of the International Joint Conference on Neural Networks-90*, Vol.I, pp.676-679.