Proceedings of the 14th
KACC, Octorber 1999

Neural Network Architecture Optimization and Application

Zhijun Liu and Masanori Sugisaka

Department of Electrical and Electronic Engineering
Oita University
700 Dannoharu, Oita 870-1192, JAPAN
Phone: +81-975-54-7831, Fax: +81-975-54-7841
zjliu@cc.oita-u.ac.jp, msugi@cc.oita-u.ac.jp

Abstract

In this paper, genetic algorithm (GA) is implemented to search for the optimal structures (i.e. the kind of neural networks, the number of inputs and hidden neurons) of neural networks which are used approximating a given nonlinear function. Two kinds of neural networks, i.e. the multilayer feedforward [1] and time delay neural networks (TDNN) [2] are involved in this paper. The synapse weights of each neural network in each generation are obtained by associated training algorithms. The simulation results of nonlinear function approximation are given out and some improvements in the future are outlined.

1. Introduction

We have some simulation results [3], [4], [5] on the function approximation application based on the neural network with fixed structures, in which, we tried one by one manually to search for the most suitable neural network structure for an multi-input one output nonlinear process in a temperature control system. Finding an alternative to automatically optimize the neural network architecture is our recent research emphasis.

To design the optimal architectures of artificial neural networks (ANNs), one attempt is implementing the evolutionary algorithms, e.g. evolutionary programming (EP) [6], genetic algorithm. There are also various designing methods relating to the automatically searching for the optimal architectures of ANNs, e.g. constructive and pruning algorithms [7].

In this paper, GA based algorithm is proposed to search for the optimal architecture of ANNs which are used to approximate a given nonlinear function [8], [9]. The chart flow of this method is illustrated by Fig. 1.

2. ANN Structures and Training Algorithms

We mainly consider two kinds of ANNs in this paper, i.e. the general multilayer feedforward neural network and time delay neural network (TDNN). For the former one, the training algorithm is the improved back propagation algorithm. For the training algorithm of TDNN, please refers to [2]. To simplify the problem, we make some limitations on TDNN structure, i.e. for any input node and hidden node in TDNN, there is only one step time delay. Fig. 2 illustrates one TDNN with two input nodes, 3 hidden nodes and one output node.

a. Randomly generate the first generation with population size N_p.

While (not terminated) do:

b. Decoding the individual of population (genotype) into the related neural networks (phenotype)

c. Carry out the training process for neural networks decoded above and calculate the fitness for each phenotype.

d. Carry out the GA operators upon the genotypes according to their fitness.

e. Termination judgement.

Fig.1 Chart flow of optimization algorithm

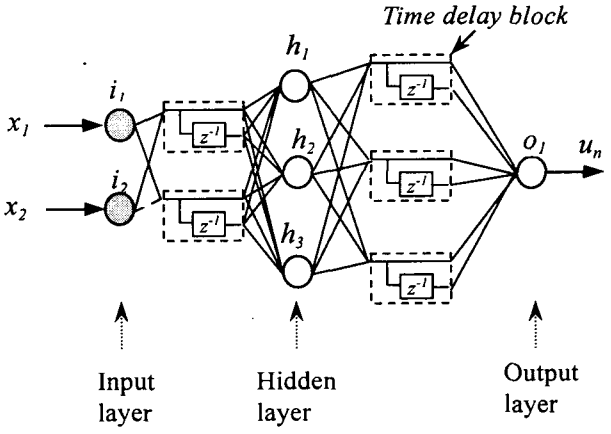

Fig. 2 One example of TDNN

3. GA Based Optimal Algorithm

3.1 Fitness Function

The fitness of each fully trained network is calculated using Eq. 1.

$$f = \frac{1}{E}(1 + \alpha(1 - \frac{N_i}{N_{i\,max}}) + \beta(1 - \frac{N_h}{N_{h\,max}})) \qquad (1)$$

where f is the fitness of the neural network, E is the error of ANN, α and βare coefficients implying the influence of inputs number and hidden neuron number, N_{imax} and N_{hmax} is the maximum number of inputs and hidden neurons, respectively. In this paper, several selection of E are available to calculate the fitness of each ANN, e.g. average absolute error (AAE), root mean square error (RMS) and mean square error (MSE).

3.2 Encoding Algorithm

The first step before using GA operators is encoding the neural networks into binary strings called chromosome. A chromosome's characteristic is determined by the genes which are presented by binary bits in this paper. There are numerous encoding algorithms in neural network optimization. Because we emphasis our research on the neural network architecture optimization, the encoded strings just contain the architecture information of ANNs, i.e. the synapse weights of ANNs will not be involved in the encoded strings. The job of obtaining suitable weights is fulfilled by using the ANN training algorithms.

The general description of encoding method is shown by Fig. 3.

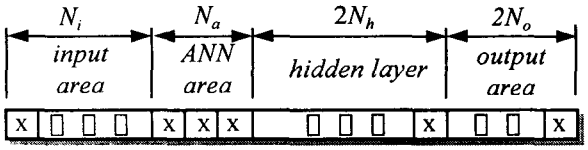

Fig. 3 The chromosome presentation

Fig. 3 shows one chromosome related to one neural network architecture, where x is one binary bit representing '0' or '1'. The chromosome consists of four parts, i.e. input area, ANN area, hidden layer area and output area, respectively. The length of each parts is N_i, N_a, $2N_h$ and $2N_o$ bits, respectively, where N_i, N_h and N_o is the number of inputs, hidden neurons and outputs of ANN, respectively.

In input area, one bit represents one input in input layer of neural network, with '1' representing the existence of the input and '0' representing the absence of the associated input. In the second portion of the string, three bits are assigned to represent different kinds of ANNs, with '001' representing multilayer feedforward neural net-works, '010' representing TDNN, etc. In hidden layer and output areas, two bits represent one neuron because we consider three kinds of active functions for each neuron, with '00' represents the absence of the neuron, '01'--- the neuron with linear function, '10' --- the neuron with sigmoid function, '11' --- the neuron with hyperbolic tangent function, respectively.

Fig. 4 shows one of multilayer feedforward neural networks (phenotype) generated by GA randomly, with the capacity of 4 inputs, 5 neurons in the first and second hidden layer, one neuron in output layer.

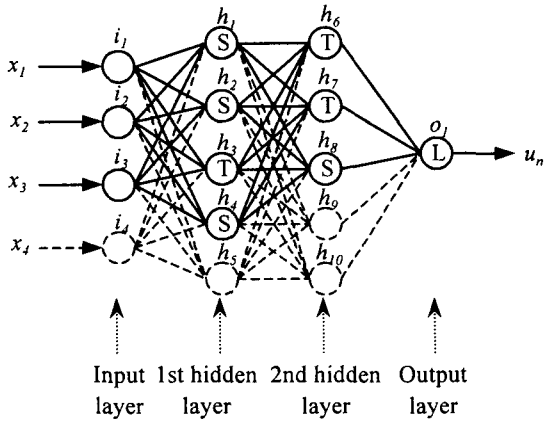

Fig.4 One of multilayer feedforward neural networks generated by GA

In Fig. 4 the dotted lines imply the absence of the associated input joints or the neurons and the related connections. The solid lines imply the existence of neurons and connections. Using the encoding technique illustrated in Fig. 3, the related chromosome (genotype) of neural network is shown by Fig. 5. Because the maximum bit length for the first and second hidden layer is defined, so it is easy to distinguish them in associated chromosome. See the dotted line in Fig.5.

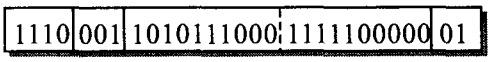

Fig.5 The chromosome of neural network

3.3 GA Operators

Three GA operators, i.e. reproduction, crossover and mutation are implemented one generation by generation to search for the optimal architecture of neural network.

3.3.1 Reproduction

One commonly used technique is roulette wheel reproduction. It can be regarded as allocating pie-shaped slices on a roulette wheel to population members, with each slice proportional to the member's fitness. There are two drawbacks to this method: it is possible that some of the best individuals may not be reproduced at all, and thus their genes may be lost. Also, it is possible that the genetic operators alter the best chromosome's gene so that whatever was good about them is destroyed.

To improve the properties of roulette wheel reproduction, an alternative called steady state reproduction is used in this paper. In this method, P_r percent of the best

individuals will just be copied into the new generation, while the remained (100- P_r) percent individuals will carry out the crossover and mutation process. This method removes the drawbacks in roulette wheel, the best individuals will always reproduce because they are simply copied into the new generation and their genes are not changed by GA operators.

3.3.2 Crossover

Crossover is one of important operators in GA unlike the case in evolutionary programming (EP), in which crossover is not implemented and only mutation is carried out.

There are different possible crossovers in GA literature, e.g. one point crossover and two point crossover. For one point crossover, one part of the parent chromosomes are exchanged at the randomly selected point, another part is kept the same as before. The simplicity of one point crossover makes it be widely used in GA operators, but its drawback is obvious, that is the possible searching space is limited because only one point is chosen.

In this paper, we use the two point crossover to over come the shortage of the previous one.

3.3.3 Mutation

Mutations are important for keeping a bit of wildness and random search flavor to optimization process. In this paper, we use the general random mutation method, that means each gene in a chromosome changes its value (called allele) from '0' to '1', or from '1' to '0' at the given probability P_m, where P_m is set to 0.0025 in the following simulation.

4. Simulation

4.1 Training and Testing Data

The nonlinear function being approximated is multi-input one output one illustrated by Fig.6a - Fig. 6f. Where, x_1, x_2, x_3, x_4, x_5 are inputs, y is the output of the real system. There are 666 sets of input and output data as the real inputs and output sets.

Two parts of these sets are separated, one is called the training data, another one is called testing data. The training data is used to train the neural network and the testing data is used calculate the fitness of the same neural network to sort the networks and find the best one in one generation. Every two real data in time sequence compose of the training sets, with the remaining 333 real data compose the testing data. Each generated neural network by GA operator or randomly at start stage will be fully trained using the training data and then, its fitness will be calculated and then the GA

To consider the influence from both the training and testing data, a combined fitness is employed in estimating the feature of evolved neural network, see

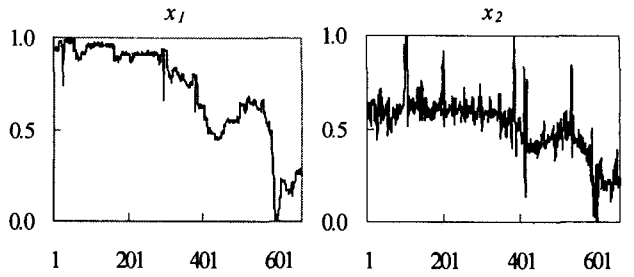

Fig.6a

Fig. 6b

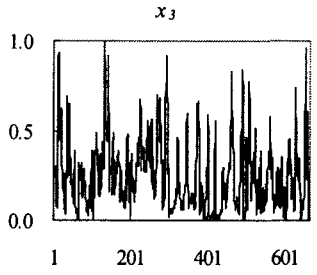

Fig.6c

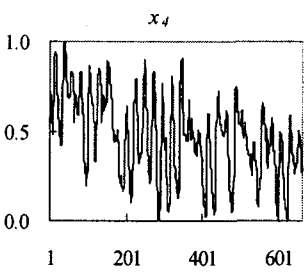

Fig.6d

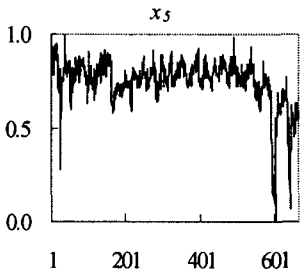

Fig.6e

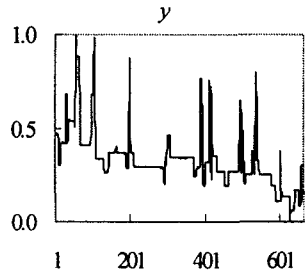

Fig.6f

Fig.6 Real inputs and outputs of the nonlinear system

$$f = \frac{\mu_1 f_{train} + \mu_2 f_{test}}{2} \tag{2}$$

where f_{train}, f_{test} is the fitness of neural network based on the training data and testing data, respectively. μ_1 and μ_2 are the influence coefficients for training and testing data, respectively. In this paper, the coefficients are selected as $\mu_1 = 0.2$, $\mu_2 = 0.8$. It is reasonable for the fitness based on the testing data has more influence in the final estimation of neural network.

4.2 Data Pre-Processing

Data pre-processing is important for the neural training.. In this paper, we employ the general dimensionless normalization technique to pre-process the real data and form the training and testing data. The data normalization is performed by Eq. 3.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3}$$

where x', x, x_{min} and x_{max} is the variable after

normalization, the original variable, the minimum of x and the maximum of x, respectively. After the data normalization, the value of training and testing data will be limited to the range of 0 - 1, see Fig. 6a - Fig. 6f.

4.3 Simulation Result

The simulation was carried out using the real data illustrated by Fig. 6a - Fig. 6f. as training and testing data sets. The system parameters are set as following.

Population size N_p = 30, generation number N_g = 100, the input factor α = 0.025, the hidden neuron factor β = 0.1, reproduction factor P_r = 0.06, the mutation probability P_m = 0.002, using two point crossover method, the maximum number of hidden neurons N_{hmax} = 16, the maximum number of inputs N_{imax} = 5, the initial random weights are limited between ±0.3, the learning rate is 0.1, the momentum coefficient is 0.08. The final result are shown by Fig.7 and Fig. 8.

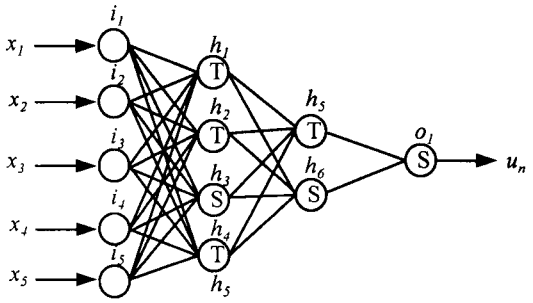

Fig. 7 The best network generated by GA

Fig. 7 is the best neural network evolved automatically by genetic algorithm with 5 inputs, 6 hidden neurons and one output.

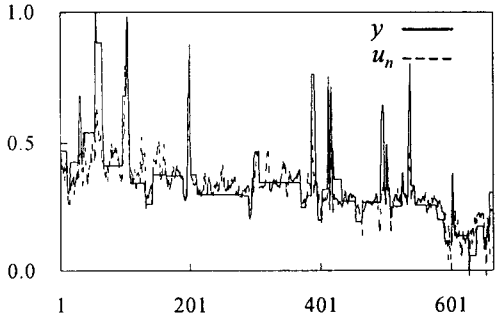

Fig. 8 The network outputs compared with real outputs

Fig. 8 illustrates the neural network output u_n comparison with the system real output y. This network is evolved in 49 generation and is trained 8921 steps, with the training average absolute error E_{train} = 0.0561, testing average absolute error E_{test} = 0.0581, the fitness of this network f= 9.19.

5. Conclusion

We made some attempts to automatically search for the optimal architecture of neural networks employing genetic algorithm. The simulation results show that the presented method is suitable in nonlinear function approximation application. According to the simulation result, a feedforward neural network with 6 hidden neurons may get the average absolute error about 0.05. In the near future, we will add other kinds of neural networks and training algorithms to expand the searching ability of this method. Also, we are considering to use this method to other applications.

References

[1] R. Parisi, E.D.D Claudio, G. Orlandi and B.D. Rao, "A generalized learning para-digm exploring the structure of feed-forward neural networks", *IEEE Transac-tions on Neural Networks,* Vol. 7, No.6, pp. 1450-1459, 1996.

[2] D.T. Lin, J.E. Dayhoff and P.A. Ligomen-ides, "A learning algorithm for adaptive time-delays in a temporal neural network", *Technical Report SRC-TR-92-59*, Systems Research Center, University of Maryland, May 1992.

[3] M. Sugisaka and Z.J. Liu, "The simulation of artificial neural network methods in cooler control using neurocomputer", *Proc-eeding of the Third Workshop of Int. Institute for General Systems Studies*, pp.140-143, Qinhuangdao, China, July 26-28, 1998.

[4] M. Sugisaka and Z.J. Liu, "The applica-tion of neurocomputer for controlling the temperature of cooler of sinter plant", *Proceeding of XIII Int. Conference on System Science*, Vol. III, pp. 352-358, Wroclaw, Poland, Sept. 15-18, 1998,

[5] M. Sugisaka and Z.J. Liu, "Artificial neural network and application in temperature control system", *Proceeding of Korea Automatic Control Conference (KACC '98)*, pp. 260-264, Pusan, Korea, Oct. 15-17, 1998.

[6] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks", *IEEE Transactions on Neural Networks,* Vol. 8, No.3, pp.694-713, 1997.

[7] G. Castellano, A.M. Fanelli and M. Pelillo, "An iterative pruning algorithm for feed-forward neural networks", *IEEE Transac-tions on Neural Networks*, Vol. 8, No.3, pp.519-531, 1997.

[8] Z.J. Liu and M. Sugisaka, "Neural network architecture optimization and application using genetic algorithm", *Proc. of the 4th Int. Symp. on Artificial Life and Robotics (AROB 4th '99)*, Vol. 2, pp. 767-770, Jan. 19-22, 1999, Beppu, Oita, Japan.

[9] Z.J. Liu and M. Sugisaka, "Optimization of neural network structures based on genetic algorithms", *11th SICE Symposium on decentralized autonomous systems*, pp. 11-14, Jan. 18-19, Nagoya, Japan.