Structure Design of Neural Networks Using Genetic Algorithms

Satoshi Mizuta
Takashi Sato
Demelo Lao
Masami Ikeda
Toshio Shimizu
Department of Electronic and Information System Engineering,
Faculty of Science and Technology, Hirosaki University,
3, Bunkyo-cho, Hirosaki 036-8561, Japan

A method for designing and training neural networks using genetic algorithms is proposed, with the aim of getting the optimal structure of the network and the optimized parameter set simultaneously. For this purpose, a fitness function depending on both the output errors and simpleness in the structure of the network is introduced. The validity of this method is checked by experiments on four logical operation problems: XOR, 6XOR, 4XOR-2AND, and 2XOR-2AND-2OR; and on two other problems: 4-bit pattern copying and an 8×8-encoder/decoder. It is concluded that, although this method is less powerful for disconnected networks, it is useful for connected ones.

1. Introduction

A neural network (NN) is a kind of processing device, which is applied to various problems such as pattern recognition, encoding/decoding, image compression, and so on. When we solve a certain problem by using a NN, we usually go through the following two phases of procedure: the first step is to design the structure of a NN corresponding to the problem under consideration, and the next is to train the network so that appropriate outputs can be derived from it. In most cases, however, since the first phase is executed by an empirical way or "trial-and-error," it is quite difficult to obtain a fully optimal structure of the network.

One of the methods to overcome such difficulties is designing the structure by genetic algorithms (GAs) [1]. Although various studies have been made on designing NN structures by using GAs [2–8], the research is presently still in progress.

Designing NNs by using GAs is roughly classified into two approaches. In the first approach, GAs are used only in designing the structure of a NN, and then training the network is carried out by another method such as back propagation. In the second, GAs are used in both structure design and training, where the optimal structure of the network and parameters for it are given simultaneously after the GA process is completed. In this article we propose a method of the second approach.

In designing NNs, a fitness function plays an important role. Fitness functions used in most of the earlier works, however, depend only on the output errors and/or their convergence. In implementing a NN it is preferable to make the network simple. In this study, therefore, we adopt a fitness function that depends on not only output errors but also the simplicity of network structure. Such a fitness function has been used in [6], but it is rather complicated and it is thought that the execution time can be improved. A fitness function in an even simpler form should be investigated.

In section 2 we define the NN investigated in this study. In section 3 we describe a method for representing a NN as a chromosome. After that we refer to the genetic operations in section 4, and show the details of the experiments and the results in section 5. In this study we carried out experiments on four logical operation problems: XOR, 6XOR, 4XOR-2AND, and 2XOR-2AND-2OR; and also on two other problems: 4-bit pattern copying and an 8×8-encoder/decoder. These are commonly used to check the validity of a network. At last, in section 6, we give conclusions.

2. Architecture of the neural network

In this study we use layered feed-forward networks with one hidden layer. Figure 1 shows an example of the network. Circles depict neurons, all of which have the value of “0” or “1”. Arrows indicate connections, the strength of which, called *weight*, has a real-number value. We consider connections between hidden neurons only from left to right, in other words, from the neuron with a smaller suffix to that with a larger one.

The values of hidden neuron H_i and output neuron O_k are calculated by

$$\begin{aligned} H_i &= \theta\left(\sum_j w_{ij}^I I_j + \sum_{j<i} w_{ij}^H H_j - t_i^H\right) \\ O_k &= \theta\left(\sum_i w_{ki}^O H_i - t_k^O\right) \end{aligned} \tag{1}$$

where w_{ij}^I is the weight from input neuron I_j to hidden neuron H_i, w_{ij}^H is the weight between hidden neurons, w_{kl}^O is the weight from hidden neuron H_i to output neuron O_k, $t_i^H (t_k^O)$ is the threshold (all of which are fixed to 1.0 for simplicity in this study), and $\theta(x)$ is a step function

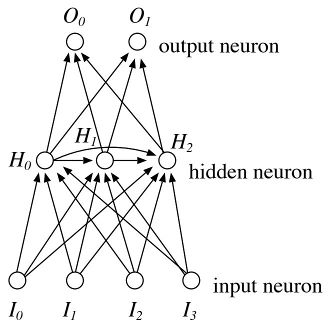

Figure 1. NN with one hidden layer.

defined as

$$\theta(x) = \begin{cases} 1 & \text{if } x \geq 0; \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The number of neurons in the input layer and that of the output layer are fixed in accordance with the problem considered. Therefore, the number of hidden neurons, the number of connections, and values of weights are to be determined with GAs.

3. Representation scheme of the network

An architecture of the network is represented by a chromosome constructed from a header and a body. Figure 2 illustrates the header. The header retains the ID-number and the fitness (defined in section 4.1) of the network and a set of ID-numbers of the output neurons included in the network. The body, on the other hand, keeps data attributed to the hidden neurons. Figure 3 shows a part of a body corresponding to a single hidden neuron. It includes the ID-number of the hidden neuron, a set of the ID-numbers of the connected neurons, and the weights of their connections.

The chromosomes are initialized as follows: at first, the number of input and output neurons are determined according to the problem con-

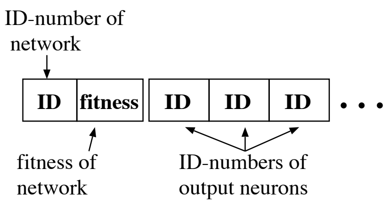

Figure 2. Header of the chromosome.

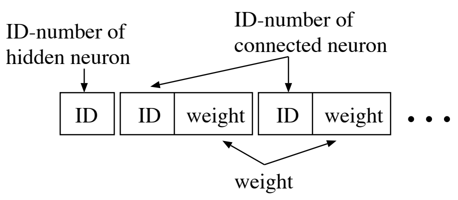

Figure 3. Body of the chromosome corresponding to a single hidden neuron.

sidered, and the neurons are assigned ID-numbers in ascending order. Next, the number of hidden neurons is randomly chosen for each individual, and the hidden neurons are assigned ID-numbers unique to each individual. Here, the population size is chosen to be 100, and it is fixed throughout the GA operation. Then, all the neurons are maximally connected with each other, and the connections are assigned weights which are randomly selected within the range between –1.0 and 1.0. Note that direct connections between input and output neurons are forbidden in this study.

4. Genetic operations

In this study we consider two kinds of genetic operations: selection and mutation. Since the length of the chromosome is variable, we do not take crossover into account.

4.1 Selection

As the selection scheme, the so-called "roulette strategy" is adopted; an individual that should survive to the next generation is selected by a probability proportional to its fitness, and this procedure is repeated as many times as the population size.

As mentioned before, GAs are used in both designing the structure and training the network. The fitness function, therefore, should depend on not only the output errors but also on the simpleness in the structure of the network. Thus, the following criteria are imposed on formulating a fitness function.

1. The more correct outputs given by the network, the larger the fitness.
2. The more hidden neurons, the smaller the fitness.
3. The more connections, the smaller the fitness.

From these criteria, the fitness function is constructed as

$$f = C\left(1 + \frac{\alpha}{2H + S}\right), \tag{3}$$

where C is the number of correct outputs in bit units given by the network, H is the number of hidden neurons, S is the number of connections normalized by the maximum number of them when the neurons are fully connected, and α is the contribution of the simpleness in the structure of the network.

Here, we should explain in detail how the actual value of α is determined. The minimum composition of the network has one hidden neuron fully connected with all of the input and output neurons, where $H = 1$ and $S = 1$. The fitness of this network is calculated from equation (3) as $f = C(1 + \alpha/3)$. On the other hand, in the maximum composition of the network, there are an infinite number of hidden neurons, where $f = C$, because $H \to +\infty$. Here, we impose a constraint that the fitness of the minimum composition is equal to or less than that of the maximum composition making one more correct answer, that is, $C(1 + \alpha/3) \leq C + 1$. Taking $C = P/2$, where P is the total number of outputs in bit units corresponding to a given problem, we get a condition $\alpha \leq 6/P$. We take the upper limit for the contribution of the simpleness at its maximum,

$$\alpha = \frac{6}{P}. \tag{4}$$

Note that, in order for the individual with the largest fitness to survive to the next generation more effectively, the fitness calculated by equation (3) is enhanced by multiplying the factor 3.

4.2 Mutations

The following operations are taken into consideration as mutation processes: addition and deletion of a hidden neuron, deletion of a connection, and modification of weights. These mutation processes are performed on all individuals excepting the 20 with the largest fitness in order for the superior individuals to be preserved in the next generation.

Addition of a hidden neuron. A hidden neuron is added to each network with a probability P_{ha}, and it is assigned the largest ID-number. The added neuron is fully connected to the other neurons also, where all the new weights are randomly chosen ranging from -1.0 to 1.0.

Deletion of a hidden neuron. One of the hidden neurons in each network is randomly selected and deleted with a probability P_{hd}. In this case, all the connections linked to and from the deleted neuron are also deleted.

Deletion of a connection. One of the connections that are attached to a hidden neuron is randomly selected and deleted by a probability P_{cd}. This operation is performed on all hidden neurons in each network.

Modification of weights. Individuals, the number of which are randomly chosen ranging from 1 to 80, are selected, and each weight in the selected networks is modified by a probability P_{wm}. The modification is performed by adding δ to the weight, where δ is randomly chosen ranging from $-w_m$ to w_m. If the absolute value of the modified weight exceeds an upper limit w_l, the weight is truncated within the range between $-w_l$ and w_l.

5. Experiments and results

To evaluate our method, we carried out the following experiments: XOR, 6XOR, 4XOR-2AND, 2XOR-2AND-2OR, 4-bit pattern copying, and an 8×8-encoder/decoder problem.

The networks in 6XOR, 4XOR-2AND, and 2XOR-2AND-2OR experiments have two input and six output neurons, and they are trained to generate outputs corresponding to each logical operation. As for the 8×8-encoder/decoder problem, the network has eight input and eight output neurons. Among the eight input neurons only one takes "1" and the others take "0" at the same time, and the network is trained to produce the same pattern as the input. The network in 4-bit pattern copying is the same as that in the 8×8-encoder/decoder problem except that the number of the input and output neurons is four and the training set includes the whole patterns which are represented by four binary digits.

Table 1 shows the parameters which are relevant to mutations, where "Value a" is applied to XOR, 6XOR, 4XOR-2AND, and 2XOR-2AND-2OR experiments, and "Value b" to the 4-bit pattern copying and 8×8-encoder/decoder problems.

5.1 XOR

Figures 4 and 5 show the convergence and the resultant network that has the maximum fitness in the XOR problem, respectively. The numerical values attached to the connection lines indicate their weights.

Parameter	Meaning	Value	
		a	b
P_{ha}	probability of adding a hidden neuron	0.008	0.01
P_{hd}	probability of deleting a hidden neuron	0.008	0.001
P_{cd}	probability of deleting a connection	0.0001	0.0001
P_{wm}	probability of weight modification	0.5	0.7
w_m	range of weight modification	0.5	0.3
w_l	upper limit of absolute value of weights	2.0	2.0

Table 1. Parameters relevant to mutations.

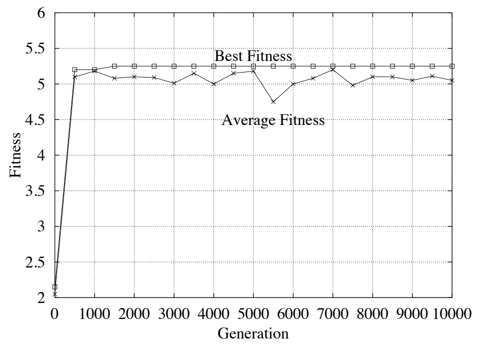

Figure 4. Convergence in the XOR problem.

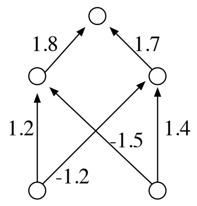

Figure 5. Resultant network that has the maximum fitness for the XOR problem. Numerical values attached to the connection lines indicate their weights.

In this study, because direct connection between input and output neurons is forbidden, the resultant network is the optimum one for the XOR problem. It is observed from Figure 4 that the network quickly converged to the optimum one at about generation 1500.

5.2 6XOR

Figures 6 and 7 show the results for the 6XOR problem. Because the networks are trained to generate six XOR outputs, a network which is simply extended from the optimum network for the XOR problem (Figure 5) to have six outputs is thought to be the optimum for the 6XOR problem (Figure 8). The number of connections in this network, however, is larger than that of the resultant network, and it has, therefore, a smaller fitness. From Figure 6, it can be seen that the network converged to the optimum one as quickly as that in the XOR problem, although the average fitness was lower in general.

5.3 4XOR–2AND

The results for the 4XOR-2AND problem are described in Figures 9 and 10. Here, numbers shown inside the circles indicate the ID-numbers of the neurons, and the weights are shown in Table 2 separately. In the

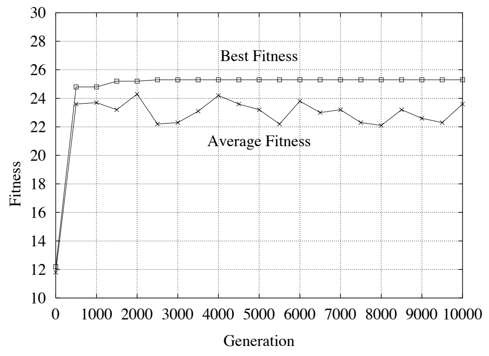

Figure 6. Convergence in the 6XOR problem.

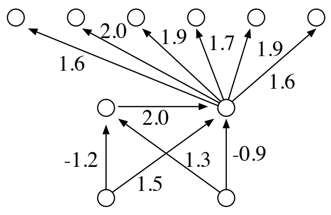

Figure 7. Resultant network that has the maximum fitness for the 6XOR problem. Numerical values attached to the connection lines indicate their weights.

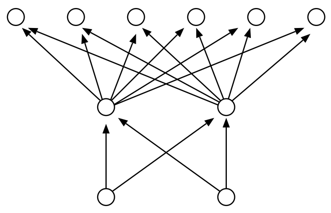

Figure 8. Network simply extended from that in Figure 5.

resultant network, neurons 8 and 9 generate an AND output and an OR output respectively, and then the two outputs are joined to produce XOR outputs in neurons 2 through 5. Because of this complexity, the convergence is about four times slower than that in the preceding two experiments.

In this network the four connection lines between neurons 8 and 2 through 5 can be replaced with a single connection from neuron 8 to neuron 9, similar to the 6XOR problem. In the framework of our algorithm, however, only one connection line per hidden neuron can be deleted at a mutation step, therefore, the probability of the occurrence of this replacement is thought to be very small. Moreover, the increase

ID-number	0	1	2	3	4	5	6	7
8	0.54	0.46	−1.02	−1.4	−1.8	−1.7	1.5	1.8
9	1.7	1.7	1.97	1.8	1.9	1.5		

Table 2. Weights for the 4XOR-2AND problem.

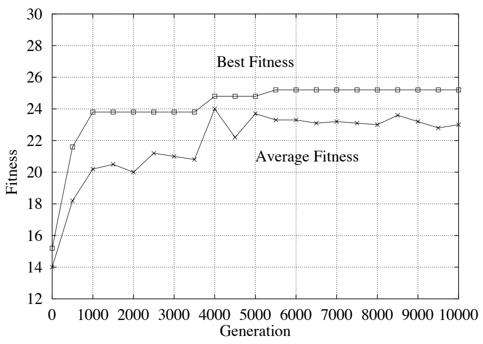

Figure 9. Convergence in the 4XOR-2AND problem.

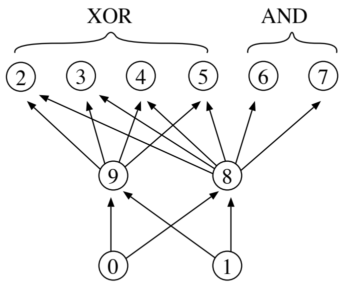

Figure 10. Resultant network that has the maximum fitness in the 4XOR-2AND problem.

in the fitness by this replacement is about 0.3%, which is too small for the optimum structure to become stable.

5.4 2XOR–2AND–2OR

Figures 11 and 12 show the results for the 2XOR-2AND-2OR problem, and the weights of the resultant network are shown in Table 3.

The optimum network was not given in this problem, in contrast with the XOR and 6XOR experiments. Neuron 8, with weights greater than 1.0 between neurons 8 and 4 and between neurons 8 and 5, could generate an AND output by itself without the connections between neurons 9 and 4 and between neurons 9 and 5. The effect of neglecting

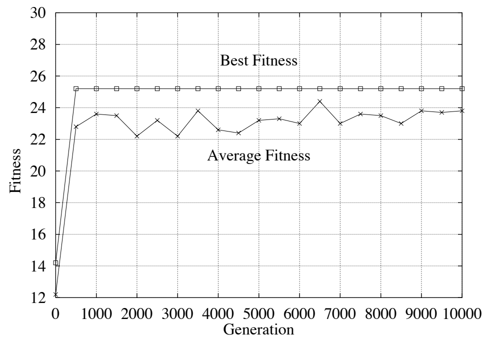

Figure 11. Convergence in the 2XOR-2AND-2OR problem.

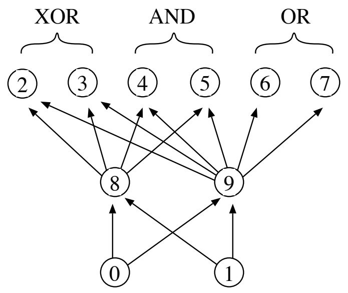

Figure 12. Resultant network that has the maximum fitness for the 2XOR-2AND-2OR problem.

ID-number	0	1	2	3	4	5	6	7
8	0.64	0.39	−1.2	−0.9	0.8	1.7	–	–
9	1.9	1.4	1.4	1.1	0.6	0.5	1.8	1.7

Table 3. Weights for the 2XOR-2AND-2OR problem.

these redundant connections to the fitness is estimated at only 0.1%. Therefore, even if an individual without these connections appears incidentally, it is difficult for the individual to survive with such a small advantage in fitness.

On the other hand, the performance of convergence in this experiment is better than that in the 4XOR-2AND problem. The reason for this phenomenon is explained as follows. In the resultant networks of Figures 9 and 11, AND and OR outputs are simple copies of output from a single hidden neuron; the appropriate values of the weights could be easily found. On the contrary, the XOR outputs must be calculated from the two hidden neurons. Seeking adequate weights for the connections, therefore, is harder than that for AND and OR outputs. The number of connections requiring this *hard training* is 8 for the 4XOR-2AND problem and 4 for the 2XOR-2AND-2OR problem. Thus, convergence in the 2XOR-2AND-2OR experiment is faster than that in the 4XOR-2AND experiment.

5.5 4-bit pattern copying

Figures 13 and 14 show the results in the 4-bit pattern copying problem, where the weights are not given. The optimum network for the 4-bit pattern copying problem in the framework of this study is illustrated in Figure 15. Although the resultant network (Figure 14) generates all

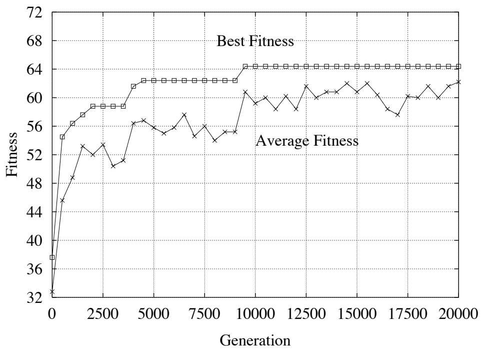

Figure 13. Convergence in the 4-bit pattern copying problem.

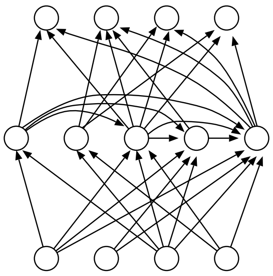

Figure 14. Resultant network that has the maximum fitness for the 4-bit pattern copying problem.

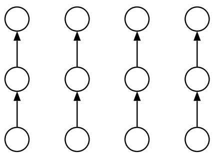

Figure 15. Optimal network for the 4-bit pattern copying problem.

of the correct outputs, the structure of the network is quite different from that of the optimum one. The resultant network has several more connections and one extra hidden neuron compared to the optimum one. The reason for this is discussed in section 6.

5.6 8×8–encoder/decoder

Figures 16 and 17 show the results in the 8×8-encoder/decoder problem. For this problem, only three hidden neurons are needed in principle, because eight different patterns can be encoded by three binary digits. In the framework of this study, however, the pattern of the hidden neurons 0-0-0 cannot be decoded to the correct output pattern, because all the outputs for 0-0-0 are calculated to be 0 under the condition that all the thresholds are fixed to 1.0 (see equation (1)). In other words, fixing all the thresholds to 1.0 forces the origin $(0, 0, 0)$ to be located on the opposite side of the direction of the normal vector on the plane separating the origin from the other points. As a result, whichever direction is chosen for the normal vector of the plane, the outputs calculated from 0-0-0 are always 0.

Thus, the network that can correctly solve the problem must have at least four hidden neurons. With four hidden neurons, all eight input patterns can be mapped onto points different from the origin in a four-dimensional space, and the patterns of hidden neurons corresponding to the points can be correctly decoded to the same patterns as the input.

Figure 18 shows one such network that can solve the problem; it is thought to be the optimum network in the framework of this study. The isolated input-hidden-output block of neurons directly copies the input value to the output. The other neurons are fully connected, and all the input patterns other than 1-0-0-0-0-0-0-0 are encoded by the right-most three hidden neurons into patterns except 0-0-0, which can be correctly decoded.

While the resultant network has four hidden neurons, it actually generated two erroneous output bits and has several more connection lines than the network in Figure 18. The reason for this is also discussed in section 6.

6. Conclusions

In this study, we investigated the capability of genetic algorithms (GAs) to design neural networks (NNs). We construct a fitness function that depends on both the ability of a network to calculate correct answers and the simpleness of its structure.

To verify the validity of our method, we applied it to six problems: XOR, 6XOR, 4XOR-2AND, 2XOR-2AND-2OR, 4-bit pattern copying, and an 8×8-encoder/decoder.

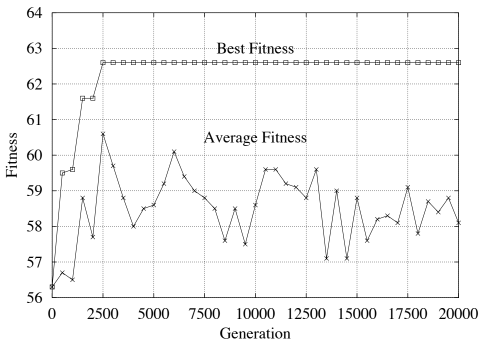

Figure 16. Convergence in the 8 × 8-encoder/decoder problem.

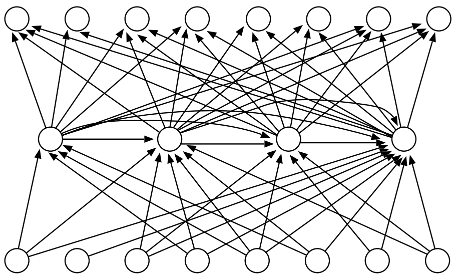

Figure 17. Resultant network that has the maximum fitness for the 8 × 8-encoder/decoder problem.

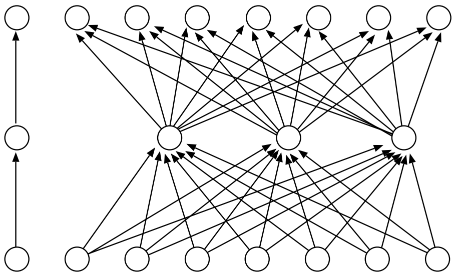

Figure 18. Candidate for the optimum network in this study of the 8 × 8-encoder/decoder problem.

In two logical operation problems out of four, XOR and 6XOR, we succeeded in getting the optimum networks. In the 4XOR-2AND and the 2XOR-2AND-2OR problems we obtained suboptimum structures of the networks.

On the other hand, in the 4-bit pattern copying and the 8 × 8-encoder/decoder problems, the resultant networks with the highest fitness are far from being the optimum ones. In the former problem the resultant network has several more connections and one extra hidden neuron, and in the latter the network generates two erroneous output bits as well as having several extra connections.

The major difference between the logical operation problems (XOR, 6XOR, 4XOR-2AND, and 2XOR-2AND-2OR) and the other problems (4-bit pattern copying and 8×8-encoder/decoder) is in the existence of isolated blocks of neurons in their optimum networks (see Figures 15 and 18). In the transition period while isolated blocks are being formed, the input patterns are partly converted into "encoded patterns" on the hidden neurons. Therefore, deleting one connection line between input and hidden neurons or between hidden neurons affects more than one output value, which may decrease the fitness by a large amount. Actually, in the network in Figure 14, deleting the connection line between the second input neuron from the left and the first hidden neuron from the right makes six erroneous outputs, which reduces the fitness by about 9%.

One way to overcome this difficulty is thought to be by increasing diversity among the individuals in a population, such as by applying more than one mutation step to an individual. This method will be adopted in the algorithm in future research.

We should include one more modification in future research. In this study we fixed all the thresholds of the neurons to 1.0. We could not get the optimum network in the 8×8-encoder/decoder problem with this constraint. However, under the condition that all the thresholds are variable, we will be able to get the optimum network for the problem, because no isolated block is needed in the optimum network in that case, and, as we have seen above, our method is suitable for exploring a connected network.

References

[1] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley, Reading, MA, 1989).

[2] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing Neural Networks Using Genetic Algorithms," *Proceedings of the Third International Conference on Genetic Algorithms* (1989) 379–384.

[3] H. Kitano, "Designing Neural Networks Using Genetic Algorithms with Graph Generation System," *Complex Systems*, **4** (1990) 461–476.

[4] D. Dasguputa and D. R. McGregor, "Designing Application-Specific Neural Networks Using the Structured Genetic Algorithm," *IEEE, International Workshop on Combinations of Genetic Algorithms and Neural Networks* (1992) 87–96.

[5] S. Bornholdt and D. Graudenz, "General Asymmetric Neural Networks and Structure Design by Genetic Algorithms," *Neural Networks*, **5** (1992) 327–334.

[6] S. Oliker, M. Furst, and O. Maimon, "A Distributed Genetic Algorithm for Neural Network Design and Training," *Complex Systems*, **6** (1992) 459–477.

[7] S. Saha and J. P. Christensen, "Genetic Design of Sparse Feed-forward Neural Networks," *Information Science*, **79** (1994) 191–200.

[8] K. P. Sankar, "Selection of Optimal Set of Weights in a Layered Network Using Genetic Algorithms," *Information Science*, **80** (1994) 213–234.