Simple Genetic Algorithm as a Learning Method for Associative Memory

Akira Imada, and Keijiro Araki
Nara Institute of Science and Technology
email: akira-i@is.aist-nara.ac.jp, araki@is.aist-nara.ac.jp

abstract

Up to now little work has been done on the application of Genetic Algorithms to Hopfield networks. In this paper, we propose a Genetic Algorithm as a learning method for associative memory system with Hopfield neural network. The connection weights are randomly determined at the start, then the network do not have an ability to store any pattern. However the network comes to store certain number of patterns after applying the Genetic Algorithm. We reported elsewhere also a Genetic Algorithm for Hopfield network though, here we redesigned it as simple as possible.

1 Introduction

In 1982 Hopfield [1] proposed a neural network model as an associative memory system. He used Hebbian rule [2] to make connection matrix memorize a set of patterns. He estimated the memory capacity to be at most 15% of the number of neurons with computer simulations. Later the capacity was verified more analytically by Amit et al. [3] with spin-glass theory, and by Amari et al. [4] with statistical neurodynamical method. Since Kohonen [5] proposed several types of connection matrices for associative memory, many researchers have successfully improved the capacity in various ways. For example, Yanai et al. [6] tried that by introducing sparse connectivity, and Morita [7] by replacing the sigmoid function with a nonmonotonic transfer function. Those mechanisms, however, are not well understood yet. Here we tried to make Hopfield network memorize some number of patterns by using only a Genetic Algorithm.

Genetic Algorithm is a kind of searching technique [8] [9]. Since the late 1980's, Genetic Algorithms have been extensively used to search the optimal set of connection weights of neural networks and/or their optimal architectures. We can see the early versions of those researches, for example, in the proceedings of a workshop: Combinations of Genetic Algorithms and Neural Networks [10]. Genetic Algorithms were used at first only for determining connection weights, (see for example [11] [12]), however the Genetic Algorithms as a learning method were not so effective compared to other algorithms like Back Propagation [15]. So the interests were shifted toward the exploration of neural network architectures (see [13] [14] for example). However, to represent architectures effectively, more sophisticated Genetic Algorithms had to be considered. For example, grammatical representations of the architecture of neural networks were proposed [15], [16]. Recently more biological approach was reported [17].

All of those researches were for layered neural networks, and only a few were for Hopfield networks [18] [19], probably because of the fact that Hopfield net-

work has a fixed architecture. However, as described above, sparce connectivity has a possibility to enhance the original associative memory capacity of Hopfield network. Therefore, Genetic Algorithms are also valid for Hopfield neural networks to explore their architectures as well as to determine their connection weights.

In an earlier papers, we showed that a Genetic Algorithm can enlarge the capacity of Hebb-rule associative memory by pruning some connections [18]. We also showed that randomly generated connection matrices can learn some patterns using a Genetic Algorithm without any learning rules [19].

In this paper, we applied a much simpler Genetic Algorithm to Hopfield neural networks, and we have found that it can also make the neural networks memorize some number of patterns.

The overall goal for this research is twofold. One is to know if we can use the Genetic Algorithm as a more effective learning method than those proposed so far, and the other is to clarify the process of this learning mechanism of the Genetic Algorithm.

2 Experiment

We applied a simple Genetic Algorithm to fully connected associative memory model of Hopfield, to be more specifically, an autoassociative discrete-time asynchronous-update binary-state network. We set the size of the network to 49 neurons in this paper considering computational expense. Before applying the Genetic Algorithm the connection weights are chosen randomly either from -1 or 1, therefore the network can not store any patterns initially. The objective of the Genetic Algorithm is to make the network store some number of patterns.

In the following part of this section, we present a brief review of associative memory, and then describe our Genetic Algorithm.

Associative Memory

Hopfield network which has N neurons can store some number of N-bit bipolar patterns, if all the connection weights are determined adequately. The patterns are n-dimensional random vectors whose elements take the values ± 1 randomly with equal probability. The μ-th pattern is denoted by

$$\xi^\mu = (\xi_1^\mu, \xi_2^\mu, \cdots, \xi_i^\mu, \cdots, \xi_N^\mu),$$

where, ξ_i^μ is the i-th bit of the μ-th pattern which takes the value of either -1 or 1. To store these p patterns, each connection weight has to be adequately adjusted. If we use Hebbian rule, for example, the weight w_{ij} is determined as follows:

$$w_{ij} = \sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu.$$

This process of determining w_{ij} is referred to as *learning*.

These w_{ij} define a so called energy function on an N-dimensional bipolar state space, i.e.,

$$E(S) = \frac{1}{2} \sum_{i,j(i \neq j)} w_{i,j} S_i S_j,$$

where S moves over all possible combinations of N dimensional vector $\{-1, 1\}^N$ and S_i is the i-th component of the state S. If the number of stored patterns

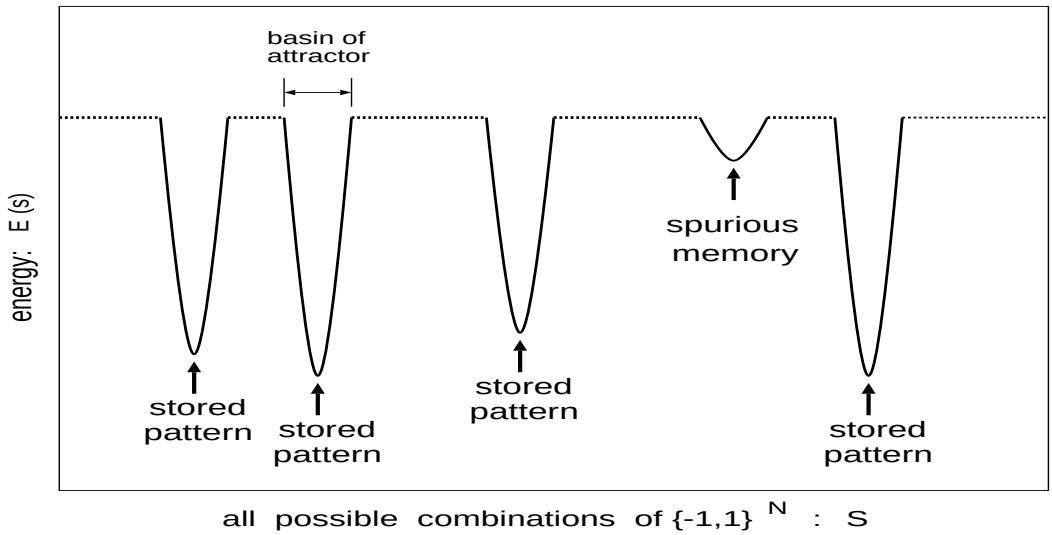

Figure 1: **Conceptual diagram of energy function**

are small enough, this function takes a local minimum where S corresponds to one of stored patterns as shown in Figure 1 (Note that abscissa is not really one-dimensional). In this sense the stored patterns are said to be *attractors* and the size of this local minimum are referred to as *basin of attractor*. As the number of stored patterns p increases, spurious minima become non-negligible and finally these local minima are randomized.

The retrieval of one of stored memories is as follows. The state of each neuron is asynchronously updated by

$$s_i^{\mu}(t+1) = sgn(\sum_{j=1}^{N} w_{ij} s_j^{\mu}(t)),$$

where $s_i^{\mu}(t)$ is the state of the i-th neuron at time t when the μ-th pattern is given to the network. For each neuron, the weighted sum of its inputs is computed, and if this is greater than or equal to zero, the state of the neuron takes the value of 1, and -1 otherwise. At each step of updating, a neuron is chosen according to pre-assigned order (once in a cycle), instead of chosen randomly. For the remainder of this paper, the term "update" will refer to this asynchronous update.

When input is one of stored patterns which is given small noise within the size of basin of attractor, this input pattern relaxes to original stored pattern after several steps of the update. In this simulation, however, we do not give any noise to input pattern. In this case, when one of a stored patterns is given to the network, all the states remain unchanged from the start, and the patterns are said to be memorized as *fixed points*. We evaluate the ability of weight matrix by the upper bound of the number of patterns stored as fixed points.

To know more about associative memory, see for example [20].

Genetic Algorithm

In this subsection we describe our Genetic Algorithm briefly and the specific details will follow them.

As in Figure 2 our Genetic Algorithm proceeds as follows:

(1) A population of random weight matrix is created (**1st generation**).

(2) The fitness of each matrix is tested (**fitness evaluation**).

(3) Two parent matrices are selected from upper subpopulation(**selection**).

(4) The chromosomes corresponding to two parent matrices are recombined to produce one child (**crossover**).

(5) This child matrix is replaced with the one in the lower subpopulation.

(6) (3)-(5) are repeated until all the individuals in the lower subpopulation are replaced with new offsprings.

(7) Mutation is added to all the chromosome of offsprings (**mutation**).

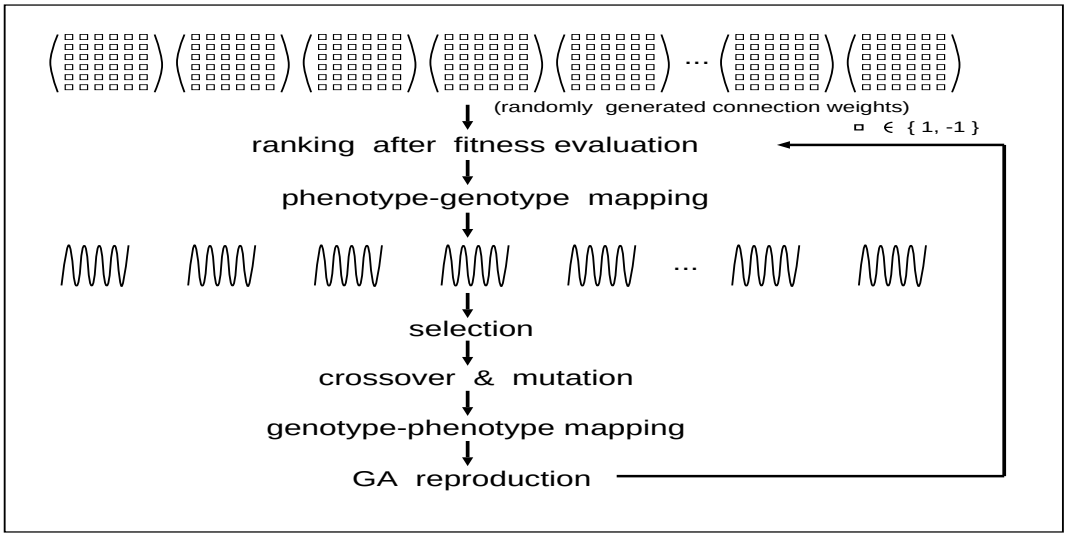

Figure 2: **Schematic representation of our Genetic Algorithm**

population

We observe that the larger the population number, the higher the performances. In these experiments, however, we set it to 256 just because of our computer resources. Each individual is a weight matrix(49x49), and in the first generation all the elements are chosen randomly from -1 or 1

fitness evaluation

As shown in Figure 3, the fitness f of each matrix is evaluated as follows. First we calculate the total of all the inner products (overlap) of initial states and the

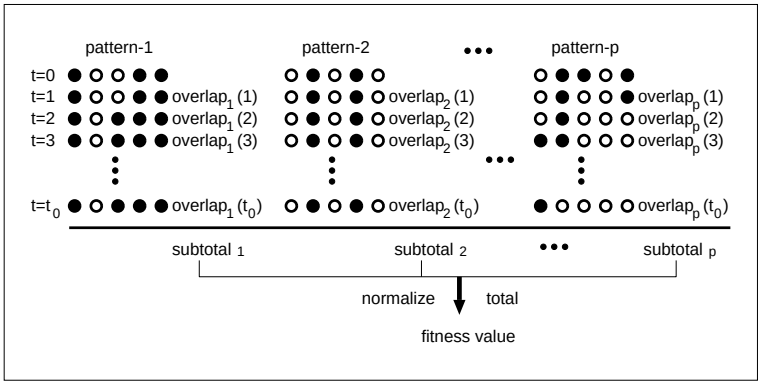

Figure 3: **Fitness evaluation**

states at each time of update not more than certain time t_0. And then these are summed up over all sets of initial patterns, i.e.,

$$f = \frac{1}{p \cdot (t_0 - 1) \cdot 49} \sum_{\mu=1}^{p} \sum_{t=2}^{t_0} \sum_{j=1}^{49} \xi_j^\mu s_j^\mu(t),$$

In this paper, t_0 is set to 98 i.e., two times the number of neurons. We must note that fitness 1.000 implies that all the initial patterns are memorized as fixed points, while fitness less than 1.000 includes many possible cases.

If we evaluated the fitness at one point of updating time instead, we might obtain a limit cycle solution as shown in Figure 4. We adopted the fitness evaluation above in order to avoid these oscillated solutions.

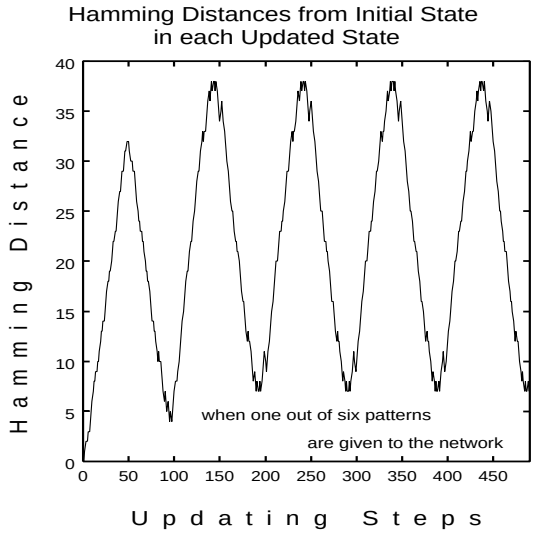

Figure 4: **Limit cycle when fitness is evaluated at one point in updating**

selection

Two parent matrices are chosen uniformly at random from the upper 40% of the population which is ranked by fitness. We refer to this upper 40% of the population as upper subpopulation and the rest 60% as lower subpopulation.

chromosome and recombination

As is depicted in Figure 5(a) chromosome is simply produced by unfolding elements of weight matrix, i.e., each allele value of the chromosome is:

$$e(i + N \cdot j) = w_{ij}, \quad i, j = 1, 2, \cdots, N.$$

Recombinations are made with uniform crossover. We tested three types of crossover: uniform-crossover, one-point-crossover and two-point-crossover, and observed that uniform crossover outperformed the other two.

mutation

Mutation changes the value of randomly chosen bit in chromosome $e(n)$ as in Figure 5(b). The mutation rate is set to 0.01 in this paper.

constructing next generation

Individuals in upper subpopulation (40%) survive to constitute the next generation with their offsprings.

These operations and values of parameters were determined mainly on the basis of trial and error.

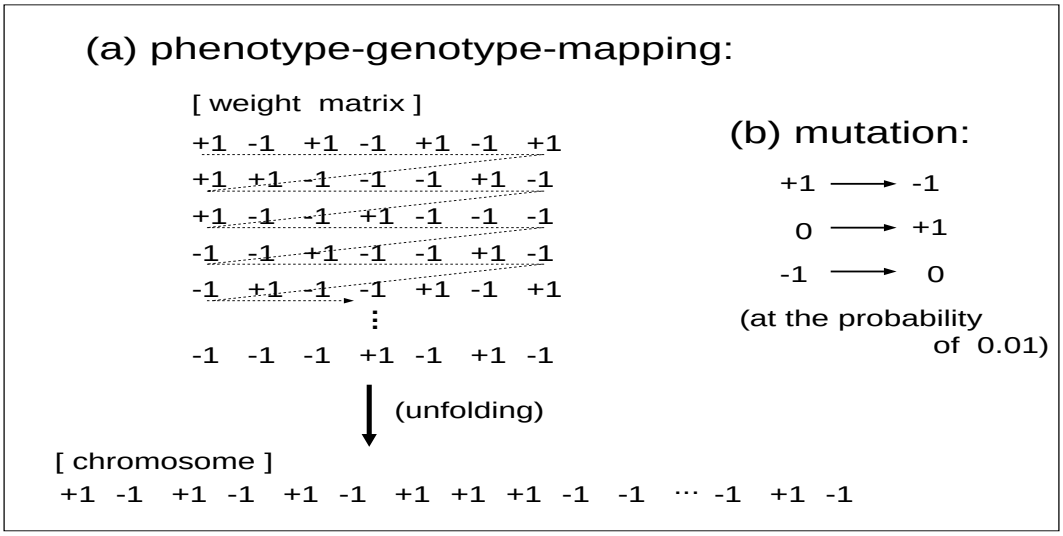

Figure 5: **Chromosome and mutation operation**

3 Results and Discussions

The simulation results are highly dependent on random number seed at the start. Hence each experiment was repeated several times with different random

number seed. Figure 6 shows representative samples of fitness vs generation from several runs. By representative sample we mean the sample from the run in which convergence was fastest.

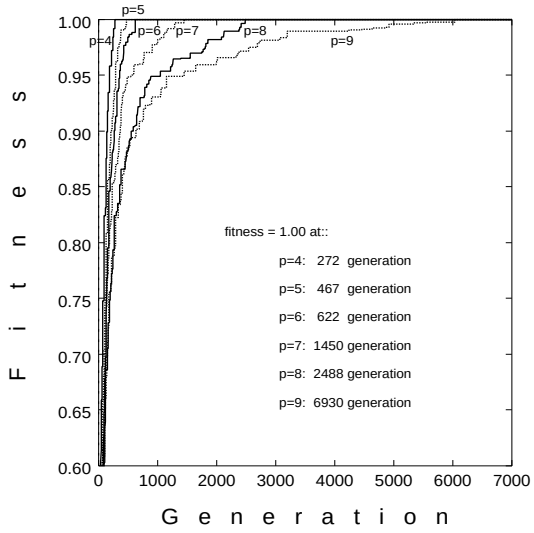

Figure 6: **Fitness vs Generation**

We can see in Figure 6 that the weight matrices of fitness 1.000 were obtained up to nine patterns. For ten patterns, on the other hand, we did not obtain the individual of fitness of 1.000 even after 12000 generations, in spite of the fact that we repeated each simulation 30 times with the same parameters except for random number seed at the start.

So we may conclude that the simple Genetic Algorithm finds weight matrix which memorize nine patterns by evolving the randomly determined matrix at the start. Since these simulations are made with 49 neurons, the ratio of the number of stored patterns to the number of neurons is about 0.18. This ratio is slightly higher than that of Hebb-rule associative memory (0.15). However the structure of matrix is totally different from each other. The weight matrix of Hebb-rule Hopfield network is symmetric with zeros in the diagonal. To see how these connection matrices change through evolution, in Figure 7 the degree of symmetry of the connection matrix (left) and the density of zero in the connection matrix (right) are plotted against generation number (for 9 patterns). Here the degree of symmetry η of a matrix was defined by

$$\eta = \sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}w_{ji} / \sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}^2.$$

The connection weights in this paper are chosen randomly from {-1, 1} in advance. Therefore the matrix is totally asymmetric and do not have zero element at the start. However as matrix is evolved to memorize some patterns, symmetry is recovered to some extent and the matrix comes to include some zero elements. Both ratios asymptotically approach to the value of about 0.20. We must note that these zeros are introduced by the mutation operation described above.

To see the role of these zeros in the weight matrix, we experimented with

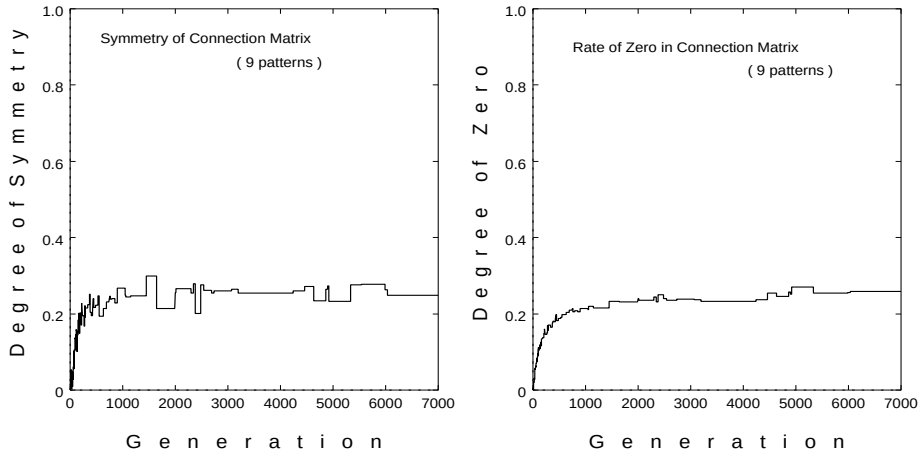

Figure 7: **Degree of symmetry and zero density**

two different mutation operators. One is to flip some bits of chromosome $e(n)$ instead of rotating them, another is to change both ± 1 to zero. Note that weight matrix can not have any zero element in the former case. The results are shown in Figure 8.

We can see in Figure 8 that zeros in the weight matrix play an important role to store patterns effectively.

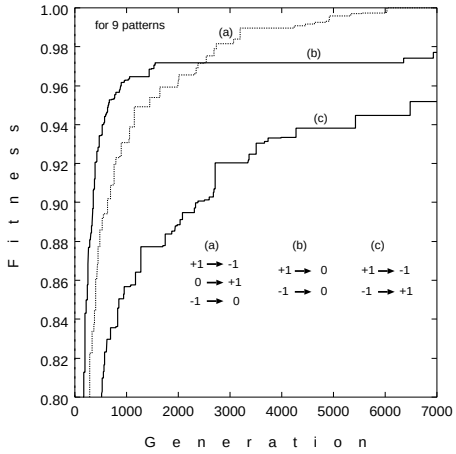

Figure 8: **The comparison of three mutations**

At first we expected weight matrix with no zero element can not store patterns perfectly, however, we observed that if the number of patterns is small enough, say four patterns, the matrix resulted from the mutation operator(c) in Figure 8 were able to store them all perfectly. On this point we can offer no more than a conjecture though, there seem to exist many local minima in fitness landscape. So far we have obtained several solutions of weight matrix which can store some number of patterns as fixed points [18] [19]. All of these solutions are different from the one of Hebb-rule associative memories, and be-

long to different local minimum.

Finally, we must note these results and analysis reported in this paper are only the beggining of our contnuing researches.

4 Conclusion

We succeeded in making a random matrix memorize nine patterns without any learning algorithm except for the Genetic Algorithm. The memory capacity of the connection matrix obtained in this paper were 18% of the number of neurons, and this is slightly better than that of Hebb-rule Hopfield network. Therefore it may lead to useful applications to get the higher capacity of associative memory, as well as helping to clarify the process of learning mechanism of the Genetic Algorithm. The architecture of the connection matrix is quite different from that of Hebb-rule Hopfield network. The connection matrices we obtained are asymmetric and include about 20% of zeros. And the behavior of both zero density and degree of symmetry correspond to the behavior of fitness value during evolution. Hence, we conjectured that these parameters play an important roll to make them memorize some patterns effectively, and Genetic Algorithm could have its own unique learning mechanism which we have not found in the already known learning algorithms.

Acknowledgments

We thank Peter Davis at Advanced Telecommunication Research Institute(ATR). The ideas in this paper were derived from discussions with him. Thanks also go to the people at Nara Institute of Science and Technology for giving us useful discussions at various stages of this research.

References

[1] J. J. Hopfield (1982). Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences, USA* **79** 2554-2558.

[2] D. O. Hebb (1949). *The Organization of Behavior.* Wiley.

[3] D. J. Amit, H. Gutfreund, and H. Somplinsky (1987). Statistical Mechanics of Neural Networks near Saturation. *Annals. of Physics* **173** 30-67.

[4] S. Amari, and K. Maginu (1987). Statistical Neurodynamics of Associative Memory. *Neural Networks* **1** 63-73.

[5] T. Kohonen (1977). *Associative Memory: A System-Theoretical Approach.* Springer-Verlag.

[6] H. Yanai, Y. Sawada, and S. Yoshizawa (1991). Dynamics of an Auto-Associative Neural Network Model with Arbitrary Connectivity and Noise in the Threshold. *Network* **2**(3) 295-314.

[7] M. Morita (1993). Associative Memory with Nonmonotone Dynamics. *Neural Networks* **6** 115-126.

[8] J. Holland (1975). *Adaptation in Natural and Artificial Systems.* The University of Michigan Press.

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

[10] J. D. Schaffer, D. Whitley (Ed.) (1992). *Proceedings of the Workshop on Combinations of Genetic Algorithms and Neural Networks.* The IEEE Computer Society Press.

[11] D. Montana, and L. Davis (1989). Training Feedforward Neural Networks using Genetic Algorithms. *Proceedings of 3rd International Conference on Genetic Algorithms* 762-767.

[12] D. Whitley, and T. Hanson (1989). Optimizing Neural Network using Faster More Accurate Genetic Search. *Proceedings of 3rd International Conference on Genetic Algorithms* 391-396.

[13] G. Miller, P. Todd, and S. Hegde (1989) Designing Neural Networks using Genetic Algorithm. *Proceedings of 3rd International Conference on Genetic Algorithms* 379-384.

[14] S. Harp, T. Samad, and A. Guha (1989). Towards the Genetic Synthesis of Neural Networks. *Proceedings of 3rd International Conference on Genetic Algorithms* 360-369.

[15] H. Kitano (1990). Designing Neural Networks using Genetic Algorithm with Graph Generation System. *Complex Systems* **4** 461-476.

[16] F. Gruau, and D. Whitley (1993). Adding Learning to the Cellular Development of Neural Networks: Evolution and Baldwin Effect. *Evolutionary Computation* **1**(3) 213-233.

[17] S. Nolfi, and D. Parisi (1995). Evolving Artificial Neural Networks that Develop in Time. *Proceedings of 3rd European Conference on Artificial Life* 353-367.

[18] A. Imada, and K. Araki (1995). Genetic Algorithm Enlarges the Capacity of Associative Memory. *Proceedings of 6th International Conference on Genetic Algorithms* 413-420.

[19] A. Imada, and K. Araki (1995). Mutually Connected Neural Network Can Learn Some Patterns by Means of GA. *Proceedings of the World Congress on Neural Networks* **1** 803-806.

[20] M. H. Hassoun (Ed.) (1993) *Associative Neural Memories: Theory and Implementation.* Oxford University Press.