Lamarckian Evolution of Associative Memory

Akira Imada
akira-i@is.aist-nara.ac.jp

Keijiro Araki
araki@is.aist-nara.ac.jp

Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-01 Japan
phone: +81-7437-2-5083

Abstract

There have been a lot of researches which apply evolutionary techniques to layered neural networks. However, their applications to Hopfield neural networks remain few so far. We have been applying Genetic Algorithms to fully connected associative memory model of Hopfield, and reported elsewhere that the network can store some number of patterns only by evolving weight matrices with a simple Genetic Algorithm. In this paper we present that the memory capacity can be enhanced by incorporating Lamarckian inheritance to the Genetic Algorithm.

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 store a set of patterns. He estimated the memory capacity to be at most 15% of the number of neurons with computer simulations. Since then many researchers have been trying to enlarge the memory capacity in various ways (see e.g., [3]). Here we are trying that by means of a Genetic Algorithm.

Genetic Algorithms are a kind of searching techniques ([4] [5]). 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 (see e.g., [6] [7]).

Genetic Algorithms were used at first only for determining connection weights, (see e.g., [8] [9]), however the Genetic Algorithms as a learning method were not so effective compared to other algorithms like Back Propagation [10]. So the interests were shifted toward the exploration of neural network architectures (see e.g., [11] [12]). 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 [10], [13]. Recently more biological approach was reported [14].

All of these researches were for layered neural networks, and only a few were for Hopfield networks [15] [16], probably because of the fact that Hopfield network has a fixed architecture. However, sparse connectivity has a possibility to enhance the original associative memory capacity of Hopfield network [17]. 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 enlarges the capacity of Hebb-rule associative memory by pruning some connections [15]. We also showed that randomly generated connection matrices can learn some patterns using a Genetic Algorithm without any learning rules [16].

In this paper, we redesigned the Genetic Algorithm so that we can incorporate the individuals' Hebbian learning effect to their chromosomes, i.e., Lamarckian inheritance. And we found that it can enlarge the memory capacity.

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

specific, an auto-associative 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 are small enough, this function takes a local minimum where S corresponds to one of stored patterns. 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. In this paper, a neuron is chosen according to pre-assigned order (once in a cycle) at each step of updating, 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 [3].

Our Genetic Algorithm

In this subsection we describe the implementation of our Genetic Algorithm briefly. As in Figure 1, our Genetic Algorithm proceeds as follows:

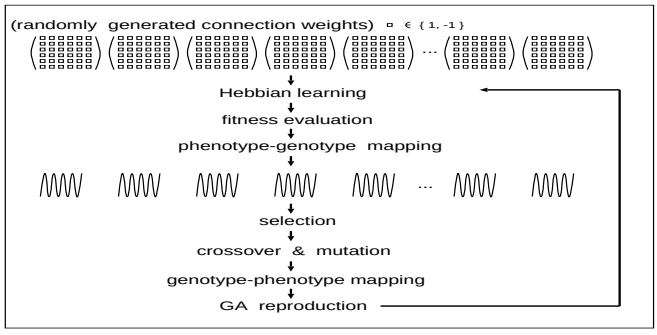

Figure 1: **Overview of our GA**

(1) At first, 256 weight matrices(49x49) W^ν are produced randomly so that each component of the matrices w_{ij}^ν are chosen from $\{-1, 1\}$. Hence the matrix does not store any patterns at this moment. We must note 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. **(2)** Each weight matrix learns a set of predetermined random patterns ξ^μ

by Hebbian learning rule as follows.

$$w_{ij} = w_{ij} + \lambda \sum_{\mu=1}^{p} \xi_i^\mu \xi_{j\neq i}^\mu.$$

We set the value of λ to 0.02 here. These components of Hebbian learned weight matrices are incorporated directly to their chromosomes in procedure(4) below. Hence we call this evolution as Lamarckian. **(3)** Evaluation of fitness value f for each individual weight matrix is made with a set of predetermined random patterns ξ^μ (see preceding subsection). First we calculate the total of all the inner products (overlap) of initial states and 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, and we observed that was enough. We must note that fitness 1.000 implies that all the initial patterns are stored 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, some solutions might be limit cycle. We adopted the fitness evaluation above in order to avoid these oscillated solutions. **(4)** Each chromosome is simply produced from corresponding weight matrix by unfolding the components i.e., each allele value of the chromosome is:

$$c^\nu(i + N \cdot j) = w_{ij}^\nu, \quad i, j = 0, 1, 2, \cdots, (N-1).$$

Therefore the chromosome has a fixed length of 2401(=49x49) alleles. **(5)** Two parent chromosomes are chosen uniformly at random from the upper 40% of the population which is ranked by fitness. **(6)** Recombinations are made with uniform crossover. We tested several types of crossover including one- and two-point-crossover, and observed that uniform crossover outperformed the others. Furthermore, mutation rotates the value of randomly chosen bit in chromosome $c^{(n)}(i)$ cyclically i.e.,

$$(1) \rightarrow (-1), \quad (-1) \rightarrow (0), \quad (0) \rightarrow (1).$$

We must note that zero components of weight matrices are introduced by this mutation operation. The mutation rate is set to 0.01 in this paper. **(7)** The chromosomes are again mapped to weight matrices with reverse process of (2). **(8)** Unless highest fitness value reaches the value of 1.000 nor generation exceeds 12000, the processes from (2) to (8) are repeated. Then individuals in upper subpopulation (40%) survive to constitute the next generation with their offsprings(60%).

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

3 Results and Discussions

To see the effect of Lamarckian inheritance in our Genetic Algorithm, we first experimented without using any learning algorithm. In this case, the network were able to store up to 9 patterns as fixed points. For 10 patterns, we repeated 30 runs with different random number seed, however we did not obtain the weight matrix which stored all these 10 patterns. In Figure 2, we show representative sample of fitness vs generation for 9 and 10 patterns. On the other hand, with Lamarckian inheritance, the weight matrices which can store more patterns emerge. In Figure 2 the example of 15 patterns for Lamarckian evolution are also shown. In this case, perfect solution was obtained at 838-th generation, while the no-learning version for 9 patterns described above converged at 6931-th generation. We can see that the Lamarckian inheritance improves the convergence speed as well as the memory capacity.

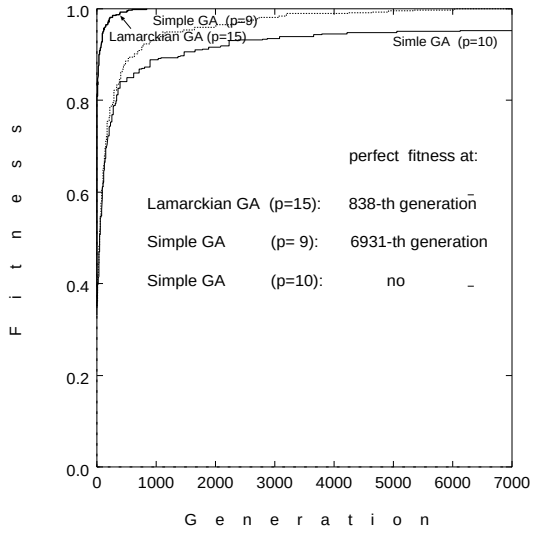

Figure 2: **Fitness vs Generation**

To see the difference between the two versions, we investigated the zero density in the weight matrices and their degree of symmetry as evolution proceeds. Here we define the degree of symmetry η of the matrix w_{ij} by

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

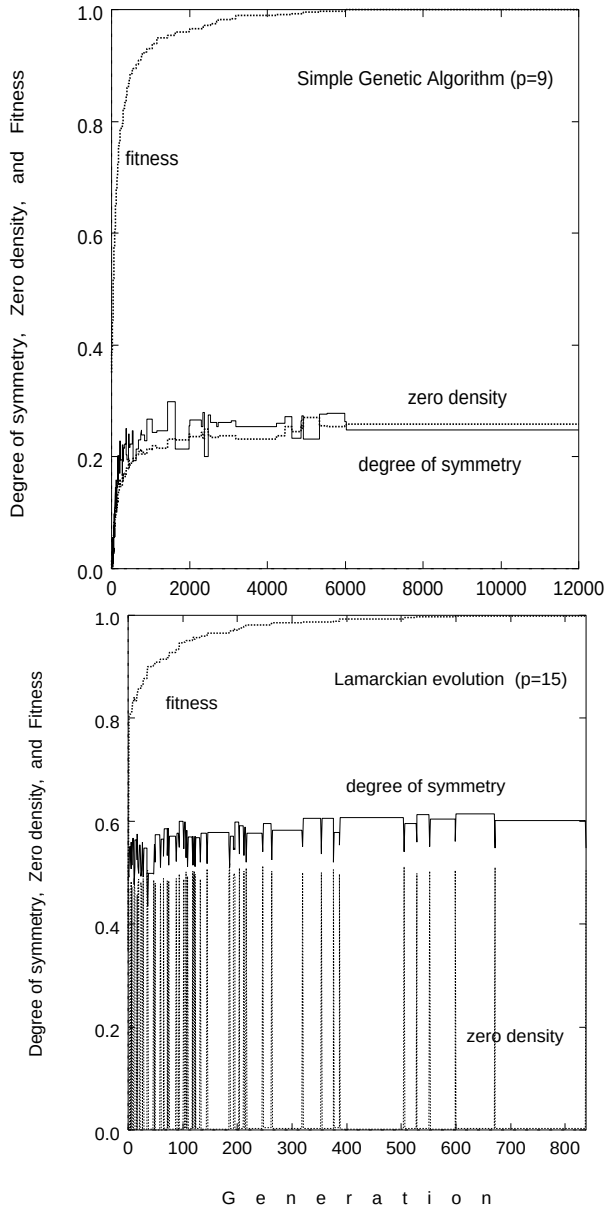

Figure 3: **Degree of symmetry and Zero density**

The results are shown in Figure 3, again with the fitness curve for convenience. The behaviors of both ratios are totally different from each other. With no-learning version (upper in the figure), both ratios are increasing from zero and asymptotically approach to the value of 0.2 approximately.

In the case of Lamarckian evolution (lower in the figure), on the other hand, both ratios start from the value between 0.3 and 0.4, because of Hebbian learning in the first generation. Degree of symmetry increases mildly toward the value of around 0.5, while zero density oscillates between two solutions of around 0 and 0.5. This shows there are two dominant species in each population which have almost a same memory capacity. They compete with each other, generation by generation, and eventually fully connected individuals dominate the population.

Original Hopfield model of associative memory with Hebbian learning has the memory capacity of 15% of the number of neurons at most. (This corresponds to 7 patterns in our simulation of 49 neurons.) Its weight matrix is totally symmetric, and all neurons are fully connected except for itself. (We must note that in the case of even number of patterns to be stored, some weight values could be zero due to Hebbian learning.)

In our Lamarckian evolution, on the other hand, we obtained weight matrix which can store 15 patterns as fixed points. We reported elsewhere [15] that both asymmetry of weight matrix and pruned connections introduced by Genetic Algorithm enlarge the capacity. Here also we may conjecture that about 50% asymmetry of weight matrix helps for the network to store more number of patterns than the original Hopfield network. However, regarding zero density, we observed that the best individual matrix have only a few zero components.

4 Conclusions

The simulations in this paper were made with 49 neurons of fully connected neural networks. This size of Hebbian-learning associative memory network would store 7 patterns as fixed points. The no-learning version of our Genetic Algorithm produced the network which can store 9 patterns. Furthermore, by introducing Lamarckian inheritance to the Genetic Algorithm, we obtain the network which can store 15 patterns. Although the improvement in capacity is not so drastic, the convergence is significantly faster in the latter case.

The architectures of weight matrices obtained in both simulations above are different. In the no-learning versions, the networks obtain the ability of storing patterns by making both density of zero in the weight matrix and degree of symmetry of the matrix asymptotically approach to the value of 0.2. In the Lamarckian evolutions,

on the other hand, degree of symmetry of the weight matrix approaches to around 0.5, and as for connections, we observed occasional appearance of best-fitness individual from species which has around 50% connectivity, while fully-connected individuals are usually dominant in the population.

The results and analysis reported here are only the beginning of our continuing research project, and many other tasks are still waiting for exploration.

Acknowledgments

We thank Peter Davis at Advanced Telecommunication Research Institute (ATR). The ideas in this paper were originally derived from discussions with him.

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

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

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

[6] J. D. Schaffer, D. Whitley, and L. J. Eshelman (1992). Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art. *Proceedings of the Workshop on Combinations of Genetic Algorithms And Neural Networks.* 1-37.

[7] X. Yao (1993). A Review of Evolutionary Artificial Neural Networks. *International Journal of Intelligent Systems.* **8** 539-567.

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

[9] 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.

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

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

[12] 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.

[13] 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.

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

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

[16] 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.

[17] 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.