Basin of Attraction of Hebb-rule Associative Memory as it Evolves

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

We are applying genetic algorithms to fully connected associative memory model of Hopfield. In an earlier paper, we reported that a Genetic Algorithm can enlarge storage capacity of Hebb-rule associative memory. The associative memory capacity obtained in that experiment was around 33% of the number of neurons, while capacity of the original Hebb-rule matrix is 15% at most. However the higher the storage capacity, the smaller the basin of attraction in general. In this paper, we present a new version of the previous work to try to control the basin size of the Hebb-rule associative memory. To our knowledge, this is the first attempt to address the size of basin of attraction of associative memory by evolutionary processes.

Key Words

Associative memory, Hebbian learning, Storage capacity, Basin of attraction, Genetic algorithm

1 Introduction

Associative memory is a process which stores and recalls information by distributing it among neurons. One of the most appealing features is its capability to tolerate noisy and/or partial input, though it has a limited capacity. We are thinking of the possibility to use it as a memory organ of digital organisms of our Artificial Life research in the future. However, as a first step, we are now investigating basic behavior of the function of associative memory under simple evolutionary processes.

In 1982, Hopfield (1982) proposed a neural network model as an associative memory system. He used Hebbian rule (Hebb, 1949) to make connection matrix store a set of patterns. He estimated the storage capacity to be at most 15% of the number of neurons with computer simulations. Since then many researchers have been trying to enlarge the storage capacity in various ways (see e.g., Yanai, Sawada, & Yoshizawa, 1991; Morita, 1993).

However, the enlargement of capacity spoils the error correcting capability of the network. The size of basin of attraction (we will describe this later in detail) decreases as the number of storage pattern increases. In fact, Personnaz, Guyon, & Dreyfus (1986) shows that the basin of attraction of a stored memory falls sharply as the number of patterns approaches to a half of the number of neurons for auto-associative memory. In this paper we wish to explore the trade off as storage capacity vs basin size with using genetic algorithm.

Genetic algorithms are a kind of searching techniques (Holland, 1975; Goldberg, 1993). 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 (Schaffer, Whitley, &

Eshelman, 1992; Yao, 1993) and references quoted therein).

However all of these researches were for layered neural networks, and applications of genetic algorithm for Hopfield networks remain few so far, 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 (Yanai, Sawada, & Yoshizawa, 1991). 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 randomly generated connection matrices can store some number of patterns using a genetic algorithm without any learning rules (Imada & Araki, 1995a). We also showed that a genetic algorithm enlarges the capacity of Hebb-rule associative memory by pruning some connections (Imada & Araki, 1995b). In these two experiments, we did not refer to the basin of attraction of the associative memory. In this paper, we redesign the genetic algorithm to try to control the size of basin of attraction.

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 mechanism of learning on a population level through evolution.

The following section provides a brief review of the associative memory. Section 3 provides an overview of the implementation of our genetic algorithm. Then results of the effect of fitness evaluations with noisy inputs on the basin size are described in Section 4. Finally some conclusions are given.

2 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}'s define a so called energy function on an N-dimensional bipolar state space, i.e.,

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

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 is 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.

When input is one of the stored patterns which is given small noise within the size of basin of attractor, this input pattern relaxes to the original stored pattern after several steps of the update. If we give no noise to the input, on the other hand, i.e., 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. However the goal in this paper is to obtain larger size of basins rather than the higher capacity.

To know more about associative memory, see for example (Hassoun, 1993).

3 Implementation of our genetic algorithm

In this section we describe the implementation of our genetic algorithm briefly. As in Figure 1, our genetic algorithm proceeds as follows:

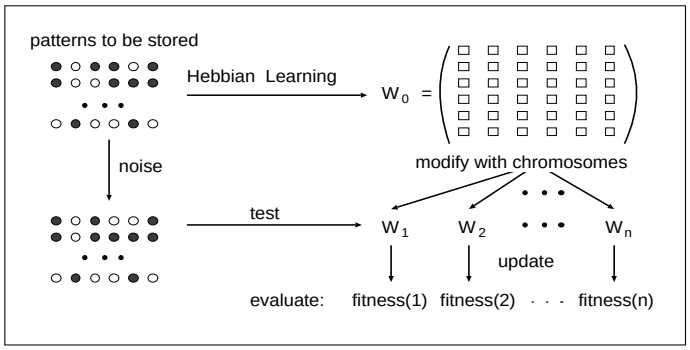

Figure 1: Schematic representation of our genetic algorithm.

(1) At first, a weight matrix W_0 are determined by Hebbian rule, i.e., each component of the matrix w_{ij} is

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

As we will describe later, we simulate with 49 neurons of Hopfield network, hence the size of W_0 is 49x49. This original weight matrix remains unchanged during evolution. **(2)** Then 256 chromosomes are produced randomly. The chromosome has a fixed length of 2401(=49x49) alleles, and these alleles are chosen randomly from $\{-1, 0, 1\}$, where the probability of choosing either -1 or 0 is set to 0.02 here. As is mentioned in procedure(3) below, each component of the original weight matrix w_{ij} is multiplied by one of these alleles. Allele 0 implies to prune the connection which is multiplied, and -1 to reverse excitatory/inhibitory connection. We denote the i-th allele of the n-th chromosome as $c^{(n)}(i)$. **(3)** Each chromosome modifies the original weight matrix W_0, which produces 256 weight matrices slightly different from W_0. The modification was made as follows:

$$w_{ij}^{(n)} = w_{ij} \cdot c^{(n)}(49i + j) \qquad (i, j = 0, 1, 2, \cdots, 48; \quad n = 1, 2, 3, \cdots 256),$$

where $w_{ij}^{(n)}$ denotes i-j component of the n-th weight matrix in the population. 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. **(4)** Evaluation of fitness value f for each

individual weight matrix is made with a set of randomly pre-determined patterns ξ^μ (see preceding section). 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 time-consuming fitness evaluation above in order to avoid these oscillated solutions. **(5)** Two parent chromosomes are chosen uniformly at random from the upper 40% of the population which is ranked by fitness. Then those are recombined. **(6)** Recombinations are made with uniform-crossover. We tested several types of crossover including one- and two-point-crossover, and observed that the uniform-crossover outperformed the others. Furthermore, the offspring from each crossover are each mutated in times, where mutation rotates the value of randomly chosen bit in chromosome $c^{(n)}(i)$ cyclically i.e.,

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

The mutation rate is set to 0.01 in this paper. **(7)** Unless highest fitness value reaches the value of 1.000 nor generation exceeds 12000, the individuals in upper subpopulation (40%) survive to constitute the next generation with their offspring(60%), and the processes from (2) to (7) are repeated.

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

4 Results and Discussions

In this section, the basins of attraction of the original Hebb-rule associative memory of Hopfield are shown at first as a basis for comparison. Then we describe the effect of fitness evaluations with noisy inputs of stored patterns on the basin size. The results of the genetic algorithm are categorized into three groups, small, medium, and large set of stored patterns. All these simulations were carried out on networks with 49 neurons.

Basin of attraction of Original Hopfield model

Since we evolve Hebbian learned weight matrix in this paper, we present, at first, how the basin size is affected with number of stored patterns in original Hebb-rule associative memory.

In Figure 2, we show the degree of tolerance to noisy input for original Hebb-rule associative memory, by plotting the similarity of updated neuron-states of noisy input to its complete pattern as a function of number of noisy bits given to the input. The network starts out with an initial configuration of neuron states (-1's and 1's) and this configuration changes in discrete time steps according to asynchronous update as was mentioned in Section 2. When one of noisy input of stored patterns is given to the network, we obtain a configuration of the state of neurons after updating long enough time steps (= 490 here). And this is compared to the initial configuration of complete version of its input. The comparison is made by using cosine of angles between two vectors which represent the two configurations above. These are averaged both over all inputs of stored patterns and over several runs (= 20 here).

Although this kind of graph can be seen in the standard literatures of associative memory, we wanted the experimental data in the situation close to the implementation of our genetic algorithm. Hence the measurements were made with networks of 49 neurons by the same way of updating of the neuron states as the one mentioned in Section 2. Therefore we must note that this degree of similarity corresponds to the fitness value in our genetic algorithm, except that the final steady state within 2 Hamming distances from the stored pattern is considered to be relaxed in a fixed point.

We can see in the figure that the Hebb-rule weight matrix can store all the patterns as fixed

points up to 10 patterns, although the degree of tolerance for noisy inputs decreases as the number of stored patterns increases. For more than 10 patterns, on the other hand, the network can not store all the patterns as fixed points. Furthermore, for over 13 patterns, the performance drops drastically. (The number of patterns stored as fixed points for 11, 12, 13, 14, 15 and 16 patterns is 9, 10, 6, 4, 5 and 3, respectively.)

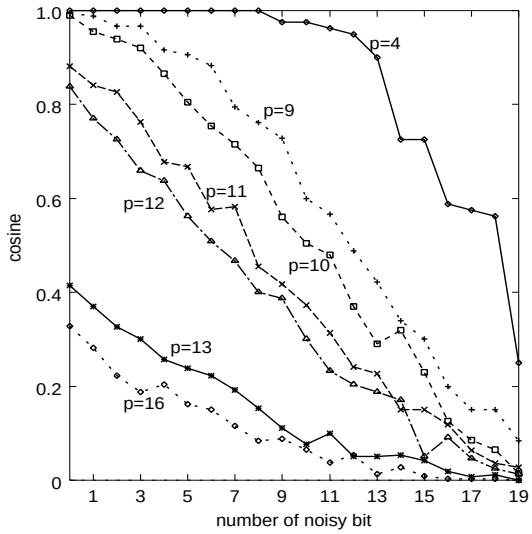

Figure 2: Tolerance to noisy input for original Hebb-rule associative memory.

4 to 10 patterns

When the number of patterns to be stored is small, as network already has a relatively large basin size, it is difficult for the genetic algorithm to search solutions of larger size of basins. Up to certain noise level, the genetic algorithm converges too quick to search them, while more than that level it never converges. We show an example for 9 patterns in Table 1. The figures in the table are the generation at which all the patterns are stored as fixed points (first two figures in the second row), or fitness values at the 12000-*th* generation if it does not converge (the last three figures in the third row). We can see in the table that up to 11 bits of noisy input, it converges to a perfect solution very rapidly, while it does not converge even after the 12000-*th* generation for over 12 bits of noises. We can observe similar type of these "limit of convergence" for the number of patterns from 4 to 10. These numbers of patterns correspond to those allowed to be perfectly stored to the original Hebb-rule associative memory (see Figure 2).

Table 1: Limit of convergence observed as the number of noisy bits increases (p=9).

number of noise	10	11	12	13	14
generation	2	10	12000	12000	12000
fitness	1.000	1.000	0.828	0.723	0.641

11 to 12 patterns

For more than storage capacity of the original Hebb-rule associative memory, even a complete input of stored patterns to the network does not necessarily relax to itself. In this case, however, genetic algorithm adjusts the weight matrix, generation by generation, so that it can store them

all as fixed points, unless the number of patterns exceeds certain limit (Imada & Araki, 1995a). However, the basin size of these attractors is not so large as in the first category. Then if we give some noise to the input patterns in evaluating fitness value, we can expect to obtain a solution of larger basin of attraction. We show an example of the results for 12 patterns in Table 2.

Table 2: Limit of Convergence observed as the number of noisy bits increases (p=11).

number of noise	1	2	3	4	5	6	7	8
generation	12	18	13	30	152	296	12000	12000
fitness	1.000	1.000	1.000	1.000	1.000	1.000	0.967	0.926

Although the "limit of convergence" still exists, the above arguments do not hold any more. The speed of convergence is long enough to search a solution of large basins (say, for 6 patterns).

In fact, we succeeded in enlarging the basin size. The result is shown in Figure 3. The ordinate of the graph is the average of cosines between the vector representing the neuron states of a noisy input after updating 490 steps and the vector representing the correct pattern. The abscissa is the number of noisy bits given to the input. (The measurements are made exactly the same way as in Figure 2.) We can see in the figure that the genetic algorithm with one noisy bit produces the matrix which store all these 11 patterns as fixed points, and the genetic algorithm with six noisy bits enlarges the basin size. We can also find a similar phenomena for 12 patterns (not shown in the figure).

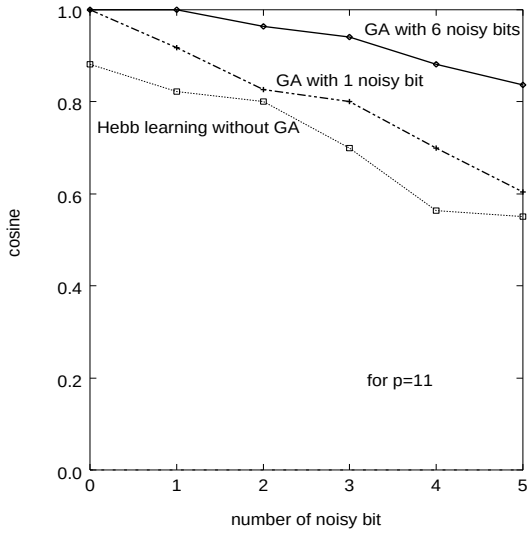

Figure 3: Tolerance to noisy input (p=11).

13 to 16 patterns

In the previous paper, we reported that a genetic algorithm can make Hebb-rule weight matrix store up to 16 patterns perfectly as fixed points (Imada & Araki, 1995b). For 16 patterns, for example, we obtained the 100% correct matrix at the 10795-*th* generation. As for the basin of attraction of that weight matrix, on the other hand, we found its basin size is quite small (not shown in this paper). Nevertheless, the new version of the genetic algorithm in this paper can not enlarge the size. And it also fails even to produce the matrix which stores the same set of 16 patterns perfectly. (The number of stored patterns as fixed points is 11.)

For 16 patterns, we show an example of fitness vs generation with one noisy bit genetic algorithm

in Figure 4. Also the "limit of convergence" for 16 patterns are shown in Table 3.

We observed a similar phenomena for 13 to 16 patterns.

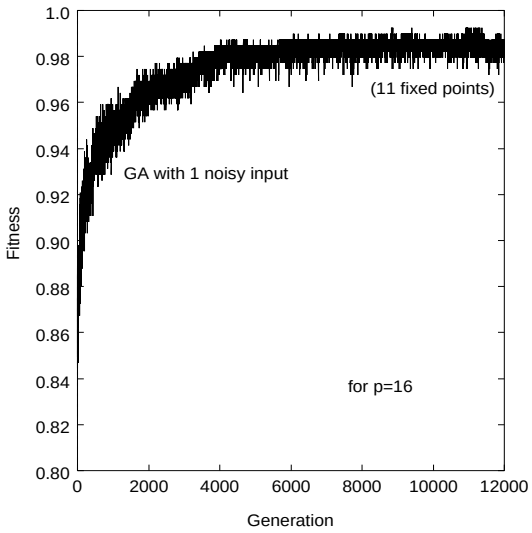

Figure 4: Fitness vs Generation. With one bit noisy inputs (p=11).

Table 3: Limit of Convergence (p=16).

number of noise	0	1
generation	10795	12000
fitness	1.000	0.985

5 Conclusions

One of the goals of this paper is to enlarge the size of basin of attraction of the network which store some number of patterns as fixed points. The simulations were carried out with 49 neurons of fully connected neural networks. And we succeeded in enlarging the basin size for the networks which store 11 and 12 patterns.

The results and analysis reported are only the beginning of our continuing research project. There might be another implementation to control the size of basin of attraction more effectively. And we are also interested in searching solutions starting from randomly determined weight matrix instead of Hebbian learned one.

Acknowledgments

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

Reference

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

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

Hebb, D. O. (1949). *The Organization of Behavior.* Wiley.

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

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

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

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

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

Personnaz, L. Guyon, I., and Dreyfus, G. (1986). Collective Computational Properties of Neural Networks: New Learning Mechanisms. *Physical Review* A**34**(5), 4217-4227.

Schaffer, J. D., Whitley, D., and Eshelman, L. J. (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.

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

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