Computers and Mathematics with Applications 60 (2010) 1049–1057

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Pattern recall analysis of the Hopfield neural network with a genetic algorithm

Somesh Kumar [a,*], Manu Pratap Singh [b]

[a] Apeejay Institute of Technology, School of Computer Science, Greater Noida, Uttar Pradesh, India

[b] Department of Computer Science, Institute of Computer & Information Science, Dr. B. R. Ambedkar University, Agra, Uttar Pradesh, India

ARTICLE INFO

Keywords:
Hopfield neural network model
Hebbian learning rule
Genetic algorithm
Pattern recalling
Population generation technique

ABSTRACT

This paper describes the implementation of a genetic algorithm to evolve the population of weight matrices for storing and recalling the patterns in a Hopfield type neural network model. In the Hopfield type neural network of associative memory, the appropriate arrangement of synaptic weights provides an associative function in the network. The energy function associated with the stable state of this model represents the appropriate storage of the input patterns. The aim is to obtain the optimal weight matrix for efficient recall of any prototype input pattern. For this, we explore the population generation technique (mutation and elitism), crossover and the fitness evaluation function for generating the new population of the weight matrices. This process continues until the selection of the last weight matrix or matrices has been performed. The experiments incorporate a neural network trained with multiple numbers of patterns using the Hebbian learning rule. In most cases, the recalling of patterns using a genetic algorithm seems to give better results than the conventional recalling with the Hebbian rule. The simulated results suggest that the genetic algorithm is the better searching technique for recalling noisy prototype input patterns.

1. Introduction

Recently, two heuristic search techniques have generated interest in the artificial intelligence community: Neural Networks (NNs) [1] and Genetic Algorithms (GAs) [2]. Both NNs and GAs are based on models from nature. A GA is a model for genetics and Darwinian evolution, whereas a NN is based on models of human cognition. One common application of the GA is as a function optimizer and another common application of the GA is for evolving organisms that perform well in a given environment. In either application, the GA is based on the survival-of-the-fittest (natural selection) tenet of Darwinian evolution. The NN, on the other hand, appears to be useful as a mechanism of control for the organisms themselves (e.g. an organism should avoid danger and seek food). These two methods naturally reflect a difference in scale, where a NN can be used to control a particular organism and a GA can be used to evolve a population of organisms (e.g. NNs) that perform well in a given environment. The NN is used to encapsulate a particular behavior and the GA can be used to evolve that behavior by evolving a population of NNs.

One particular approach to the evolution of behavior has been described by Garis [3]. In this approach, a GA has been used to evolve a population of NNs. Each NN had a set of adjustable weights and was used to encapsulate some desired behavior (e.g. walking). In other words, once good weights have been found, the NN can adjust itself to perform the desired behavior.

* Corresponding author.
E-mail addresses: someshkumarrajput@rediff.com (S. Kumar), manu_p_singh@hotmail.com (M.P. Singh).

0898-1221/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2010.03.061

Table 1
Genetic operators used in the experiments.

Training algorithms	Genetic operators used
Hebbian rule	None
Genetic algorithm	Population generation technique (mutation + elitism), crossover and fitness evaluation technique

However, since a set of good weights are not known in advance, they must be learnt. There are various learning techniques available by which the network can be trained to adopt a desired behavior [4].

Associative memory is a dynamical system, which has a number of stable states with a domain of attraction around them [5]. If the system starts at any state in the domain, it will converge to the stable state. In 1982, Hopfield [6] proposed a fully connected neural network model of associative memory in which we can store information by distributing it among neurons and recall it from neuron states that are dynamically relaxed.

The dynamical behavior of the neuron states strongly depends on the strength of connection between the neurons. The strength of connection of neuron j to neuron i is called the weight and denoted by w_{ij}. When all the weights w_{ij} have been optimized approximately, the network can store some number of patterns as associative memory. Then an input, one of the stored patterns, including a few errors, will result in the relaxation of neuron states to the initial input state. If the appropriate optimized weight matrix is used then the network will produce the correct stored information. In this case, a noisy version of the input should also converge in the attractor. Hopfield used the Hebbian learning rule [7] to prescribe the weight matrix. Various learning techniques have been proposed besides the Hebbian rule. The pseudo-inverse matrix method given by Kohonen et al. [8] and the spectral algorithm given by Pancha et al. [9] are examples. It has been realized by many researchers [10,11] that as the number of input patterns increases, a network can be trapped into false minima during the recalling process. It can be understood that the problem of false minima can be minimized with the selection of the correct weight matrix corresponding to the present noisy input pattern for recalling the correct stored pattern. In the process of determining the correct weight matrix, we use the genetic algorithm technique.

Researchers [12,13] have combined neural networks and genetic algorithms in a number of different ways. Schaffer et al. [12] have noted that this combination can be classified in two different ways—as supportive combinations in which the neural network and genetic algorithm are applied simultaneously, and collaborative approaches, where the genetic algorithm and neural networks are integrated into a single system in which a population of neural networks evolves. Thus, the goal of a system was to find the optimal neural network solution [14]. The genetic algorithm is applied to optimize a neural network with the evolution of weights. In this approach, the genetic algorithm is used as the learning rule for the neural network.

In this paper, we apply a genetic algorithm to Hopfield's neural network model of associative memory. Our goal is to obtain the suitable weight matrices for efficient recalling of an input prototype pattern/noisy input prototype pattern. For this purpose, we explore the population generation technique, the crossover operator and the fitness evaluation function in order to generate the optimal weight matrix. The experiments consist of neural network training with multiple numbers of patterns using the Hebbian learning rule. In most cases, the recalling of patterns using a genetic algorithm seems to give better results than the conventional recalling with the Hebbian rule. The results suggest that the genetic algorithm is the better searching technique for recalling noisy prototype input patterns.

The next section presents the simulation design of the problem. The algorithmic steps are presented in Section 3. The experimental results are described in Section 4. Sections 5 and 6 conclude this paper with a summary, the conclusion of this study and a discussion of future research directions.

2. Simulation design and implementation details

The experiments described in this segment have been designed to evaluate the performance of a neural network with a genetic algorithm for the pattern recalling.

2.1. Experiments

Two experiments have been run based on different network architectures, i.e. five- and ten-neuron networks. Each experiment runs with three different numbers of storage of patterns, i.e. N, $2N$ and $3N$ patterns, which have been generated randomly, where N is the number of neurons in the Hopfield neural network. Because of the random generation of the patterns, the same pattern may be generated multiple numbers of times and will be stored in the Hopfield neural network. It may also occur that all possible combinations of $(1, -1)$ for a fixed number of neurons which design a pattern cannot be generated because of the random generation of patterns at storing time; e.g. if the number of neurons in the network is 5, then the possible number of patterns will be 2^5. In each experiment, the Hebbian learning rule is used to store the patterns in the Hopfield neural network and two different algorithms, i.e. the Hebbian rule and the genetic algorithm, are used for recalling the patterns. The genetic operators used in each experiment are summarized in Table 1.

Descriptions of these genetic operators can be found in Section 2.3.

The parameters used in all three experiments are described in Tables 2 and 3.

Table 2
Parameters used for the Hebbian learning rule.

Parameter	Value
Initial state of neurons	Randomly generated values between −1 and 1
Threshold values of neurons	0.00

Table 3
Parameters used for the genetic algorithm.

Parameter	Value
Initial state of neurons	Randomly generated values between −1 and 1
Threshold values of neurons	0.00
Mutation population size	$N + 1$
Mutation probability	0.5
Crossover population size	$N * N$

The task associated with the Hopfield neural networks in all experiments is storing the number of input patterns with the appropriate recalling of the noisy prototype input pattern.

2.2. The Hopfield neural network

The proposed Hopfield model consists of N neurons and N^2 connection strengths. Each neuron can be in one of two states, i.e. ± 1, and p bipolar patterns $X^\mu = \left(x_1^\mu, x_2^\mu, \ldots, x_N^\mu\right), \mu = (1, 2, \ldots, p)$, are to be memorized in associative memory. Hopfield employed a discrete-time, asynchronous update scheme. Namely, each neuron updates its states, one at a time, as

$$S_i\,(t+1) = f\left(\sum_{i\neq j}^{N} w_{ij} S_j\,(t)\right) \tag{2.2.1}$$

where $S_i\,(t)$ is the state of the ith neuron at time t, w_{ij} is a synaptic weight, for the connection between neuron j and neuron i, and $f\,(z) = 1$ if $z \geq 0$ and -1 otherwise, where $z = \sum_{i\neq j}^{N} w_{ij} S_j\,(t)$.

The behaviors of the collective states of individual neurons are characterized by the synaptic weights. When these synaptic weights are determined appropriately, the networks store some number of patterns as fixed points. Hopfield specified the w_{ij}'s by using the Hebbian rule [7], i.e.

$$w_{ij} = \sum_{\mu=1}^{p} x_i^\mu x_j^\mu \quad (i \neq j)\,; \qquad w_{ii} = 0. \tag{2.2.2}$$

Hopfield associated an energy function [6] with the state of the network at equilibrium in order to represent the stored pattern as minima of the energy landscape. The energy function has been considered as

$$E^\mu\,(S) = -\frac{1}{2}\sum_{i}\sum_{j\neq i} w_{ij}\,(t)\, S_i^\mu\,(t)\, S_j^\mu\,(t)\,. \tag{2.2.3}$$

So for storing a pattern, the energy function of Eq. (2.2.3) should be minimized. Thus, Eq. (2.2.3) will be minimized only for

$$w_{ij}\,(t) = S_i^\mu\,(t)\, S_j^\mu\,(t)\,. \tag{2.2.4}$$

Therefore, from Eq. (2.2.3) we have

$$E^\mu\,(S) = -\frac{1}{2}\sum_{i}\sum_{j\neq i} S_i^{2\mu}\,(t)\, S_j^{2\mu}\,(t)\,. \tag{2.2.5}$$

Here, $S_i\,(t)$ and $S_j(t)$ are the output states of the units ith and jth in the bipolar pattern, i.e. $\{-1, 1\}$. Thus, to store the p input patterns the optimized weight matrix W is constructed using Eq. (2.2.4):

$$W^p = \begin{bmatrix} 0 & S_1S_2 & S_1S_3 & \ldots & S_1S_N \\ S_2S_1 & 0 & S_2S_3 & \ldots & S_2S_N \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ S_NS_1 & S_NS_2 & S_NS_3 & \ldots & 0 \end{bmatrix}_{N\times N}. \tag{2.2.6}$$

This square matrix is known as the parent weight matrix for storing the given input patterns. Thus, on introducing one of the memorized patterns, including a few errors, into the network, an initial state will result as the stable state after a certain

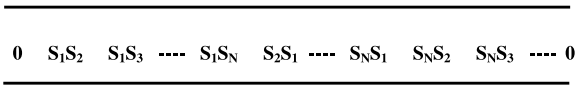

Fig. 1. Chromosome representation.

number of iterations. Hopfield [6] suggested that the maximum limit for the storage is $0.15N$ in a network with N neurons if a small error in recalling is allowed. It has also been observed that the possibility of false minima may occur during the recalling of memorized patterns.

Efficient recalling depends strongly on how synaptic strengths are specified. The specification of the synaptic strengths is conventionally known as learning.

2.3. The GA implementation

In this simulation, a population of weight matrices is produced randomly when the GA starts. In each generation, the matrices of this population have been modified through discrete crossovers and uniformly random mutations, and their fitness values have been evaluated. According to the fitness values, individuals of the next generation are selected, using a $(\mu + \lambda)$-strategy in ES terminology. The cycle of reconstructing the new population with better individuals and restarting the search is repeated until a perfect solution is found. The Hopfield energy function is used for selecting the most suitable weight matrices, as the second fitness evaluation function.

The population generation technique

The weight matrix W^p and associated energy functions, from Eq. (2.2.5), $E^1, E^2, \ldots, E^p$, have been determined by using the Hebbian rule for storing the p bipolar patterns. Each component of the weight matrix w_{ij} has been determined from Eq. (2.2.2).

The population generation technique produces the population of N weight matrices of the same order as the original parent weight matrix. The original weight matrix remains unchanged during the evaluation. The total number M (i.e. $N+1$) of chromosomes are produced after using the mutation and elitism. Each chromosome has a fixed length of $N \times N$ alleles.

Each component of the original weight matrix, w_{ij}, i.e. S_iS_j, is multiplied by one of these alleles. We denote the ith allele of the nth chromosome as A_i^n. Each chromosome modifies the original weight matrix W^P and produces N weight matrices slightly different from W^P. The modifications can be represented as

$$w_{ij}^n = S_iS_jA^n\,(N.i + j) \quad (i, j = 1, 2, \ldots, N)\,, (n = 1, 2, \ldots, M)\,, \tag{2.3.1}$$

where w_{ij}^n denotes the i–j component of the nth weight matrix in the population.

The pseudo-code for the population generation technique

Step 1: Generate the mutation positions in the chromosome randomly.
Step 2: Modify the parent chromosome shown in Fig. 1 at the positions generated in step 1, using Eq. (2.3.1) and $\{1, -1\}$.
Step 3: Repeat steps 1 and 2 until a number N of mutated chromosome populations have been created.
Step 4: Apply elitism to include the parent chromosome in the mutated populations, which makes the population count M (i.e. $N + 1$).

The first fitness evaluation

Now for selecting a good or efficient next generation of weight matrices, the first fitness evaluation function (f) is used. Evaluation of f for each individual weight matrix is carried out with a set of randomly pre-determined patterns X^μ. When one of stored patterns X^μ is given to the network as an initial state, the state of neurons varies over time until X^μ is a fixed point. In order to store the pattern in the network, these two states must be similar. The similarity as a function of time is defined by

$$z^\mu\,(t) = \frac{1}{N}\sum_{i=1}^{N} x_i^\mu S_i^\mu\,(t)\,. \tag{2.3.2}$$

Here $S_i^\mu\,(t)$ is the state of the ith neuron at time t. In evaluating the fitness value, the temporal average overlap $\langle z^\mu \rangle$ is calculated for each stored pattern, as follows. First the total of the inner products of the initial states and states is calculated at each time of update not greater than a certain time t_0. After that, these are summed up over whole set of initial patterns, i.e.,

$$f = \frac{1}{t_0 p}\sum_{t=1}^{t_0}\sum_{\mu=1}^{p} z^\mu\,(t)\,. \tag{2.3.3}$$

Here t_0 has been set to N (the number of processing units). We must note that fitness 1 implies that all the initial patterns have been stored as fixed points. Thus, we consider only those generated weight matrices that have the fitness evaluation value 1. Hence, all the selected weight matrices will be considered as the new generation of the population.

This new population will be used for generating the next better population of weight matrices with the recombination or crossover operator.

The crossover operator

Crossover is an operation which may be used to combine multiple parents and make offspring [15]. This operator is responsible for the recombination of the selected population of weight matrices. This operator forms a new solution by taking some parameters from one parent and exchanging them with ones from another at the very same point. Here, we are applying the recombination with the uniform crossover. In this process, the network selects randomly a string of non-zero chromosomes from a selected weight matrix and exchanges it with string of non-zero chromosomes from another selected weight matrix. Thus, a large population of the weight matrices will be generated. Hence, on applying this crossover operator with the constraint that the numbers of chromosomes or components selected for exchange should be equal for the two weight matrices, the modification has been made in the selected weight matrices as follows:

$$\sum_{r}^{N\times N} w_{ij}^{n} = \sum_{r}^{N\times N} S_i S_j A^k \,(N.i + j)$$

and

$$\sum_{r}^{N\times N} w_{ij}^{k} = \sum_{r}^{N\times N} S_i S_j A^n \,(N.i + j) \quad (i, j = 1, 2, \ldots, N; k, n = 1, 2, \ldots, T; n \neq k)\,. \tag{2.3.4}$$

Here T is the selected weight matrix for M generated matrices, $r = 1$ to $N \times N$, only for non-zero elements, and w_{ij}^{n} and w_{ij}^{k} denote the i–j components of the nth and kth weight matrices in the population. Thus, we have a new large population of K weight matrices from the crossover operator as follows:

$$\left\{ w_{1_{N\times N}}^{new}, w_{2_{N\times N}}^{new}, \ldots, w_{K_{N\times N}}^{new} \right\}. \tag{2.3.5}$$

The pseudo-code for the crossover operation

Step 1: Initialize the crossover population size limit with value $N * N$.
Step 2: Extract two chromosomes from among the M (i.e. $N + 1$) chromosomes randomly.
Step 3: Obtain a random position in each extracted chromosome for exchanging the values.
Step 4: Exchange the values between the chromosomes.
Step 5: Include both chromosomes in the crossover population.
Step 6: Check whether the population size is equal to $N * N$. If not, go to step 2 again.

The second fitness evaluation

In the process of recalling the stored pattern, corresponding to a noisy prototype input pattern, the most suitable weight matrix or matrices will be selected from the generated population K of weight matrices. Our genetic algorithm implementation will use the Hopfield energy function analysis as the second fitness evaluation. Hence, in this process, the network uses the energy function for representing each stored input pattern from Eq. (2.2.5) and the energy function estimation from each of the generated weight matrices with the state of the network on presenting a prototype input pattern. Let the energy functions from Eq. (2.2.5) be represented as $E^{\mu}(S)$, where $\mu = 1, 2, \ldots, p$ (the number of patterns), and the energy function corresponding to the generated weight matrices for the input pattern X^{μ} be represented as $E_N^{\mu}(S)$, where $N = 1, 2, \ldots, K$. Now, for a noisy prototype input pattern, say $(X^{\mu} + \in)$, the energy function for the generated weight matrices is given as

$$E_N^{\mu}(S) = -\frac{1}{2} \sum_{i} \sum_{j\neq i} w_i^{new}(t)\, S_i^{\mu+\in}(t)\, S_j^{\mu+\in}(t)\,, \quad \text{where } N = 1, 2, \ldots, K. \tag{2.3.6}$$

Hence, the fitness evaluation of these weight matrices for the presented input pattern $(X^{\mu} + \in)$ is defined as

$$F^i = \left[E_i^{\mu}(S) - E^{\mu}(S) \right] = 0; \quad i = 1, 2, \ldots, K. \tag{2.3.7}$$

Hence, we select only those weight matrices for which F^i is zero. Thus, these energy functions are the same and represent the same minima of the energy landscape. Hence, the pattern that is stored at that minimum will be recalled as the network settles into these minima. The selected weight matrices will be used as the final weight matrices for recalling an input pattern efficiently. In this manner, the proposed method finds a useful technique for efficient recalling of a given input pattern corresponding to the noisy prototype input pattern. The following are the algorithmic steps of the above described technique.

3. The algorithm

1. Initialize the network states, weights, and threshold randomly to keep the network at a stable state.
2. Present the input pattern $X^{\mu} = \left(x_1^{\mu}, x_2^{\mu}, \ldots, x_N^{\mu}\right)$ to the network and store it as [where $\mu = (1, 2, \ldots, p)$]

$$S_i\,(t+1) = f\left(\sum_{i \neq j}^{N} w_{ij} S_j\,(t)\right)$$

and

$$w_{ij} = \sum_{\mu=1}^{p} x_i^{\mu} x_j^{\mu} \quad (i \neq j)\,; \qquad w_{ii} = 0.$$

The energy function associated with the state of the network at stability is given as

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

3. Determine the parent weight matrix for storing the input patterns using

$$w_{ij}\,(t) = S_i^{\mu}\,(t)\, S_j^{\mu}\,(t)$$

and

$$W^{p} = \begin{bmatrix} 0 & S_1S_2 & S_1S_3 & \ldots & S_1S_N \\ S_2S_1 & 0 & S_2S_3 & \ldots & S_2S_N \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ S_NS_1 & S_NS_2 & S_NS_3 & \ldots & 0 \end{bmatrix}_{N \times N}.$$

4. Generate the population of weight matrices from the parent weight matrix using the population generation technique. Each chromosome modifies the original weight matrix to

$$w_{ij}^{n} = S_iS_jA^{n}\,(N.i + j) \quad (i, j = 1, 2, \ldots, N)\,, (n = 1, 2, \ldots, M)\,.$$

The best population of the weight matrices can be selected as

$$f = \frac{1}{t_0 p} \sum_{t=1}^{t_0} \sum_{\mu=1}^{p} z^{\mu}\,(t)$$

and

$$z^{\mu}\,(t) = \frac{1}{N} \sum_{i=1}^{N} x_i^{\mu} S_i^{\mu}\,(t)\,.$$

Here x^{μ} is the pattern given to the network as an initial state, $S_i^{\mu}\,(t)$ is the state of the ith neuron at time t and t_0 is set to the number of processing units in the network, i.e. N. The weight matrices selected from the population must have the fitness evaluation value 1.

5. The next generation population of the weight matrices will be generated by using the crossover operator among the selected weight matrices of step 4. The numbers of chromosomes or components selected for exchange should be equal for the two weight matrices. The modification or recombination is made as follows:

$$\sum_{r}^{N \times N} w_{ij}^{n} = \sum_{r}^{N \times N} S_iS_jA^{k}\,(N.i + j)$$

and

$$\sum_{r}^{N \times N} w_{ij}^{k} = \sum_{r}^{N \times N} S_iS_jA^{n}\,(N.i + j)\,.$$

Here $i, j = 1, 2, \ldots, N$; $k, n = 1, 2, \ldots, T$; $n \neq k$; $r = 1$ to $N \times N$, only for non-zero elements.

6. w_{ij}^{n} and w_{ij}^{k} denote the i–j components of the nth and kth weight matrices in the selected population of T weight matrices. The new large population of K weight matrices is given as follows:

$$\left\{ w_{1_{N \times N}}^{new}, w_{2_{N \times N}}^{new}, \ldots, w_{K_{N \times N}}^{new} \right\}.$$

Table 4
Results for recalling patterns which involve zero-bit error from the stored patterns.

S. No.	Number of stored patterns in the network	No. of patterns recalled using the Hebbian rule (%)	No. of patterns recalled using the genetic algorithm (%)
1	5	1	59
2	10	1	50
3	15	6	53

Table 5
Results for recalling patterns which involve one-bit error from the stored patterns.

S. No.	Number of stored patterns in the network	No. of patterns recalled using the Hebbian rule (%)	No. of patterns recalled using the genetic algorithm (%)
1	5	2	50
2	10	4	36
3	15	0	54

Table 6
Results for recalling patterns which involve two-bit error from the stored patterns.

S. No.	Number of stored patterns in the network	No. of patterns recalled using the Hebbian rule (%)	No. of patterns recalled using the genetic algorithm (%)
1	5	6	38
2	10	2	49
3	15	0	19

Table 7
Results for recalling patterns which involve zero-bit error from the stored patterns.

S. No.	Number of stored patterns in the network	No. of patterns recalled using the Hebbian rule (%)	No. of patterns recalled using the genetic algorithm (%)
1	10	0	17
2	20	0	22
3	30	6	23

7. Calculate the energy function corresponding to the generated weight matrices on presenting a noisy prototype input pattern $(X^{\mu} + \in)$ as follows:

$$E_N^{\mu}(S) = -\frac{1}{2}\sum_i \sum_{j \neq i} w_i^{new}(t)\, S_i^{\mu+\in}(t)\, S_j^{\mu+\in}(t); \quad N = 1, 2, \ldots, K.$$

The fitness evaluation of the new population of K weight matrices for selecting them for recalling is given as

$$F^i = \left[E_i^{\mu}(S) - E^{\mu}(S)\right] = 0; \quad i = 1, 2, \ldots, K.$$

Then, we select only those weight matrices for which $F^i = 0$.
8. Stop.

4. Results

4.1. *Experiment 1* ($N = 5$)

See Tables 4–6.

4.2. *Experiment 2* ($N = 10$)

See Tables 7–9.

5. Discussion

The results of the previous section demonstrate that, within the simulation framework and selected training data sets, a large significant difference exists between the performance with the conventional Hebbian rule and the GA when applied to the Hopfield neural network model for the recalling of presented prototype noisy input patterns.

We have considered training data sets for two different numbers of neurons, i.e. 5 and 10, in the Hopfield neural network and executed a separate experiment for each, i.e. experiments 1 and 2. We have obtained three different numbers of storage

Table 8
Results for recalling patterns which involve one-bit error from the stored patterns.

S. No.	Number of stored patterns in the network	No. of patterns recalled using the Hebbian rule (%)	No. of patterns recalled using the genetic algorithm (%)
1	10	1	7
2	20	0	27
3	30	0	15

Table 9
Results for recalling patterns which involve two-bit error from the stored patterns.

S. No.	Number of stored patterns in the network	No. of patterns recalled using the Hebbian rule (%)	No. of patterns recalled using the genetic algorithm (%)
1	10	0	5
2	20	0	10
3	30	0	6

of patterns randomly for each experiment, i.e. numbers N, $2N$ and $3N$ of storage patterns, where N is the number of neurons in the Hopfield neural network. This can be understood as us obtaining a maximum of 2^N patterns for the number N of neurons. We are storing the numbers N, $2N$ and $3N$ of patterns in the Hopfield model among the 2^N patterns, where N represents the total number of neurons in the network. Because of the random generation of patterns, a similar pattern may be generated several times.

At the time of recalling also, the pattern has been generated randomly. So, any pattern can be obtained among the maximum of 2^N patterns. This pattern may not exist in the stored pattern list of the network and then does not have zero-, one-, and two-bit errors from them. Therefore, it will not be recalled in both the experiments, because we have only considered the zero-, one-, and two-bit error cases. We obtain a total of 1000 patterns for recalling and distribute them according to the error from stored patterns in the network. After that, we applied both recalling techniques to them and got the simulation results shown in Tables 4–9.

As far as the probability of a mutation operator of a genetic algorithm is concerned, we have set it as 0.5 to avoid randomness in the search process. This probability is set as a constant for every experiment.

Tables 4–6 have been designed for experiment 1 in which the network consists of five neurons. The tables have been separated on the basis of bit errors between the presented prototype patterns and training set patterns. Each table contains three entries based on the number of stored patterns in the network, i.e. numbers N, $2N$ and $3N$ of storage patterns. The tables also contain the experimental results, which are obtained by applying the Hebbian rule and the genetic algorithm for recalling the patterns. Tables 7–9 are constructed by using the same concept but these tables are for ten neurons in the Hopfield network.

The simulation programs have been developed in MATLAB 6.5, for testing the performance of the Hebbian rule and the genetic algorithm in the recalling of patterns stored in a Hopfield neural network. Due to the limitation of resources, we could only run two experiments and could not go beyond that. However, on the basis of the given simulation results we can say that the genetic algorithm is more efficient and consistent for recalling the patterns/noisy patterns in comparison with the Hebbian rule.

6. Conclusion

The simulation results of in this paper indicate that the genetic algorithm is a better searching technique for recalling noisy prototype input patterns in comparison to the Hebbian rule for the Hopfield type neural network model.

The simulation results, i.e. Tables 4–9, indicate that the genetic algorithm has a greater success rate than the Hebbian rule as regards recalling the patterns containing zero-, one- and two-bit errors from stored patterns in the Hopfield neural network. Sometimes it has also been observed that the performance of the GA is less than the expected percentage. This may happen because of the storage of more than one pattern in the same energy minimum or equilibrium. The recalled pattern can be any pattern stored in the minimum, not necessarily the corresponding right pattern. However, it is certain that the patterns are selected from the correct energy minimum. Another reason for the occurrence of this is the large amount of patterns stored from the random generation in the network.

We found that the genetic algorithm is more consistent than the Hebbian rule in recalling the pattern. Say we have obtained the same pattern, which has already been stored in the network multiple numbers of times for recalling, and applied both techniques for recalling the corresponding stored pattern. Then the GA gives the same results every time whereas the Hebbian rule gives different results every time in the recalling process.

The direct application of the GA to the pattern association has been explored in this research. The aim is to introduce the GA as an alternative approach for solving the pattern association problem. The results from the experiments conducted on the algorithm are quite encouraging. Nevertheless more work needs to be performed, especially on tests for large complex patterns. Some future work should also be carried out. For instance, current work shows the performance of the GA exceeding that of the Hebbian rule in recalling processes when up to ten neurons exist in the network, but we can proceed further and

use this idea for more neurons. We can also use this concept for any pattern recognition/association problem to obtain efficient and consistent results in the future.

References

[1] S. Amari, Learning and statistical inference, in: M.A. Arbib (Ed.), The Handbook of Brain Theory and Neural Networks, MIT Press, Cambridge, MA, 1993, pp. 522–526.
[2] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1996.
[3] H. de Garis, Genetic programming: building nanobrains with genetically programmed neural network modules, in: Proceeding of the International Joint Conference on Neural Networks, San Diego, CA, June 1990.
[4] J.M. Zurada, Introduction to Artificial Neural Systems, Info Access and Distribution, Singapore, 1992.
[5] A. Imada, K. Araki, Basin of attraction of associative memory as it is evolves from random weights, in: Proc. of the 1st Asia–Pacific Conf. on Simulated Evolution and Learning, 1996, pp. 271–278.
[6] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proceedings of The National Academy Sciences, USA 79 (1982) 2554–2558.
[7] D. Hebb, The organization of behaviour, in: A Neuropsychological Theory, Wiley, New York, 1949.
[8] T. Kohonen, M. Ruohonen, Representation of associated data by matrix operators, Institute of Electrical and Electronics Engineers. Transactions on Computers C 22 (7) (1973) 701.
[9] G. Pancha, S.S. VenKatesh, Feature and memory-selective error correction in neural associative memory, in: M.H. Hassoun (Ed.), Associative Neural Memories: Theory and Implementation, Oxford University Press, 1993, p. 225.
[10] A. Imada, K. Araki, Genetic algorithm enlarges the capacity of associative memory, in: Proc. of the 6th International Conf. on Genetic Algorithms, 1995, pp. 413–420.
[11] M. Morita, Associative memory with non-monotone dynamics, Neural Networks 6 (1993) 115–226.
[12] J.D. Schaffer, D. Whitley, L.J. Eshelman, Combinations of genetic algorithms and neural networks: a survey of the state of the art, in: Proceedings of the Workshop on Combinations of Genetic Algorithms and Neural Networks, 1992, pp. 1–37.
[13] X. Yao, A review of evolutionary artificial neural networks, International Journal of Intelligent Systems 8 (1993) 539–567.
[14] J. Branke, Evolutionary algorithms for neural network design and training, in: 1st Nordic Workshop on Genetic Algorithms and its Applications, Vaasa, Finland, 1995.
[15] D. Goldberg, Genetic Algorithm in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.