Applied Soft Computing 12 (2012) 2593–2600

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

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

Study of Hopfield neural network with sub-optimal and random GA for pattern recalling of English characters

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

[a] Noida Institute of Engineering and Technology, Greater Noida, Uttar Pradesh, India
[b] Department of Computer Science, Institute of Computer and Information Science, Dr. B.R. Ambedkar University, Khandari Campus, Agra, Uttar Pradesh, India

ARTICLE INFO

Article history:
Received 31 October 2008
Received in revised form
10 November 2011
Accepted 18 March 2012
Available online 11 April 2012

Keywords:
Hopfield neural network
Hebbian learning rule
Genetic algorithm
Pattern recognition

ABSTRACT

In this paper we are studying the performance of Hopfield neural network for recalling of memorized patterns from the Hebbian rule and genetic algorithm for English characters. In this process the genetic algorithm is employed in random form and sub-optimal form for recalling of memorized patterns corresponding to the presented noisy prototype input patterns. The objective of this study is to determine the optimal weight matrix for correct recalling corresponding to noisy form of the English characters. In this study the performance of neural network is evaluated in terms of the rate of success for recalling of noisy input patterns of the English characters with GA in two aspects. The first aspect reflects the random nature of the GA and the second one exhibits the suboptimal nature of the GA for its exploration. The simulated results demonstrate the better performance of network for recalling of the stored letters of English alphabets using genetic algorithm on the suboptimal weight matrix.

1. Introduction

Pattern storage is one of the techniques for the pattern recognition task that one would like to realize using an artificial neural network (ANN). Pattern storage is generally accomplished by a feedback network consisting of processing units with non-linear bipolar output functions. The Hopfield neural network is a simple feedback neural network (NN) which is able to store patterns in a manner rather similar to the brain – the full pattern can be recovered if the network is presented with only partial information. The stable states of the network represent the stored patterns. Since the Hopfield neural network with associative memory [1,2] was introduced, various modifications [3–10] are developed for the purpose of storing and retrieving memory patterns as fixed-point attractors. The dynamics of these networks have been studied extensively because of their potential applications [11–14]. The dynamics determines the retrieval quality of the associative memories corresponding to already stored patterns. A major drawback of this type of neural networks is that the memory attractors are constantly accompanied with a huge number of spurious memory attractors so that the network dynamics is very likely to be trapped in these attractors [6], and thereby prevents the retrieval of the memory attractors.

Hopfield [1] 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 the dynamically relaxed neuron states. Hopfield used the Hebbian learning rule [15] to prescribe the weight matrix. Hopfield type networks will most likely be trapped in non-optimal local minima close to the starting point, which is not desired. The presence of false minima will increase the probability of error in recall of the stored pattern. The problem of false minima can be reduced by adopting the evolutionary algorithm to accomplish the search for global minima.

Developed by Holland [16], an evolutionary searching is a biologically inspired search technique. In simple terms, the technique involves generating a random initial population of individuals, each of which represents a potential solution to a problem. Each member of that population's fitness as a solution to the problem is evaluated against some known criteria. Members of the population are then selected for reproduction based upon that fitness, and a new generation of potential solutions is generated from the off-spring of the fit individuals. The process of evaluation, selection, and recombination are iterated until the population converges to an acceptable solution. Genetic algorithms (GAs) require only fitness information, not gradient information or other internal knowledge of a problem as in case of neural networks. Genetic algorithms have traditionally been used in optimization but, with a few enhancements, can perform classification, prediction and pattern association as well [17–19].

The neural network applications address problems in pattern classification, prediction, financial analysis, control, and

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

1568-4946/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2012.03.049

optimization [20]. In most current applications, neural networks are best used as aids to human decision makers instead of substitutes for them. Genetic algorithms have helped market researchers performing market segmentation analysis [21]. Genetic algorithms and neural networks can be integrated into a single application to take advantage of the best features of these technologies [22].

Much work has been done on the evolution of neural networks with GA [23–27]. There have been a lot of researches which apply evolutionary techniques to layered neural networks. However, their applications to fully connected neural networks remain few so far. The first attempt to conjugate evolutionary algorithms with Hopfield neural networks dealt with training of connection weights. Evolution has been introduced in neural networks at three levels: architectures, connection weights and learning rules [28]. The evolution of connection weights proceeds at the lowest level on the fastest time scale in an environment determined by architecture, a learning rule, and learning tasks. The evolution of connection weights introduces an adaptive and global approach to training, especially in the reinforcement learning and recurrent network learning paradigm.

In this paper, the recalling is performed under the consideration of reducing the affect of false minima with evolutionary searching method like genetic algorithm. We consider the GA with the Hopfield neural network for recalling of the noisy form of memorized patterns. In this approach the GA is applied in two different aspects. In the first aspect the GA starts from the random weight matrix or solution to determine the optimal weight matrix which is required for efficient recalling of the presented noisy English alphabets. The second aspect of the GA starts from the suboptimal weight matrix. The suboptimal weight matrix reflects the encoded pattern's information of the training set. Hence, the GA starts from the correlation matrix of the training set which we call as parent weight matrix and determine the optimal weight matrix for the presented noisy input patterns of the English alphabets. The performance of the network is evaluated for the pattern recalling corresponding to the noisy English alphabets with random GA and suboptimal GA. The simulated results indicate the better performance of the suboptimal GA as compared to the random GA and the Hebbian rule in terms of the success rate for recalling the noisy English alphabets.

In the following sections we will present the description of patterns used for training, the Hopfield neural network used for storing the patterns, the GA used for recalling the already stored patterns, experiments detail, discussion of our results obtained through simulation, and the conclusion of our investigations.

2. The set of patterns used for training

The patterns used for the simulations are shown in Fig. 1. Each pattern consists of a 7×5 pixel matrix representing a letter of the alphabet. White and black pixels are respectively assigned corresponding values of −1 and +1.

Now, the input pattern vector for the storage corresponding to English letters is constituted with the series of bipolar values +1 and −1. For example, the pattern vector for letter A can be written as:

[−1−11−1−1−11−11−1−11−11−11−1−1−11111111−1−1−111
−1−1−11]

In the general form we can represent the lth pattern vector as:

$$a^l = \left[a_1^l, a_2^l, \cdots, a_i^l, \cdots, a_j^l, \cdots, a_{35}^l\right] \tag{2.1}$$

where $l = 1{:}26$ and $i, j = 1{:}35$.

3. The Hopfield neural network

The proposed Hopfield model consists of $N(35 = 7 \times 5)$ neurons and $N \times N$ connection strengths. Each neuron can be in one of the two states, i.e. ±1, and L bipolar patterns have to be memorized in associative memory.

For storing L patterns, we could choose a Hebbian rule given by the summation of the Hebbian terms for each pattern, i.e.

$$w_{ij} = \frac{1}{N}\sum_{l=1}^{L} a_i^l a_j^l (i \neq j),\ w_{ii} = 0; \quad \text{and} \quad (i, j = 1, 2, \cdots, N). \tag{3.1}$$

Hence, in order to store 26 letters of English alphabet (all capitals) in a 35-unit bipolar Hopfield neural network, there should be one stable state corresponding to each stored pattern. Thus, the following activation dynamics equation must be satisfied to accomplish the storage:

$$f\left(\sum_{j \neq i}^{N} w_{ij} a_j\right) = a_i; \quad \text{where} \quad i, j = 1, 2, \cdots, N. \tag{3.2}$$

Let the pattern set be $p = \begin{bmatrix} a_1^1 & a_2^1 & a_3^1 & \cdots & a_N^1 \\ a_1^2 & a_2^2 & a_3^2 & \cdots & a_N^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_1^L & a_2^L & a_3^L & \cdots & a_N^L \end{bmatrix}_{L \times N}$

where

$$\begin{aligned} &\left[\mathbf{a}^1 = \left(a_1^1, a_2^1, \cdots, a_N^1\right)^T,\right. \\ &\mathbf{a}^2 = \left(a_1^2, a_2^2, \cdots, a_N^2\right)^T, \\ &\qquad\vdots \\ &\mathbf{a}^L = \left(a_1^L, a_2^L, \cdots, a_N^L\right)^T, \\ &\left.\text{with } N = 35 \quad \text{and} \quad L = 26\right]. \end{aligned} \tag{3.3}$$

From the synaptic dynamics as vectors we have the following equations for encoding the patterns information with initialization of weight vector by zero:

$$\mathbf{W} = \mathbf{P}^T\mathbf{P} \tag{3.4}$$

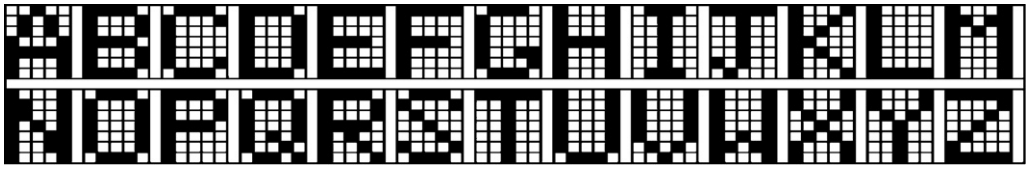

Fig. 1. The set of patterns used for training.

Thus, after the learning for all the patterns, the final parent weight matrix can be represented as:

$$\mathbf{W} = \begin{bmatrix} 0 & \frac{1}{N}\sum_{l=1}^{L} a_1^l a_2^l & \frac{1}{N}\sum_{l=1}^{L} a_1^l a_3^l & \cdots & \frac{1}{N}\sum_{l=1}^{L} a_1^l a_N^l \\ \frac{1}{N}\sum_{l=1}^{L} a_2^l a_1^l & 0 & \frac{1}{N}\sum_{l=1}^{L} a_2^l a_3^l & \cdots & \frac{1}{N}\sum_{l=1}^{L} a_2^l a_N^l \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{N}\sum_{l=1}^{L} a_N^l a_1^l & \frac{1}{N}\sum_{l=1}^{L} a_N^l a_2^l & \frac{1}{N}\sum_{l=1}^{L} a_N^l a_3^l & \cdots & 0 \end{bmatrix}_{N\times N} \tag{3.5}$$

Now, to represent **W** in the convenient form, let us assume the notation s_i to express the state of ith unit at stability. So that, states of the units at fixed point stability is expressed as:

$$\begin{aligned} s_1s_2 &= \sum_{l=1}^{L} a_1^l a_2^l, s_1s_3 = \sum_{l=1}^{L} a_1^l a_3^l, ---, s_1s_N = \sum_{l=1}^{N} a_1^l a_N^l, s_2s_1 \\ &= \sum_{l=1}^{L} a_2^l a_1^l, s_2s_3 = \sum_{l=1}^{L} a_2^l a_3^l, ---, s_2s_N = \sum_{l=1}^{L} a_2^l a_N^l, s_Ns_1 \\ &= \sum_{l=1}^{L} a_N^l a_1^l, s_Ns_2 = \sum_{l=1}^{L} a_N^l a_2^l, s_Ns_3 = \sum_{l=1}^{L} a_N^l a_3^l, ---. \end{aligned} \tag{3.6}$$

Therefore, from Eqs. (3.5) and (3.6), we get

$$\mathbf{W} = \frac{1}{N}\begin{bmatrix} 0 & s_1s_2 & s_1s_3 & --- & s_1s_N \\ s_2s_1 & 0 & s_2s_3 & --- & s_2s_N \\ | & | & | & | & | \\ | & | & | & | & | \\ s_Ns_1 & s_Ns_2 & s_Ns_3 & --- & 0 \end{bmatrix} \tag{3.7}$$

This square matrix is known as parent or suboptimal weight matrix for storing the given input patterns because it represents the storing of input patterns. Hopfield 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. Later, this was theoretically calculated as $0.14N$ by using replica method [3]. Wasserman [29] showed that the maximum number of memories 'm' that can be stored in a network of 'n' neurons and recalled exactly is less that cn^2 where 'c'is a positive constant greater than one. It has been observed that the possibility of false minima may occur during the recalling of memorized patterns. Kumar and Singh investigated [14] that the GA of the evolutionary algorithm is much suitable choice to reduce the affect of false minima from the Hopfield neural network during the recalling of memorized patterns.

4. The genetic algorithm implementation

In this simulation, we are using the GA in two forms. The first form is representing the normal procedure for applying the GA as with random population or weight matrix. The second form starts with a sub-optimal solution to achieve the optimal solution. In order to implement the sub optimal GA we consider a population of sub-optimal solution which is modified through uniform random mutation, discrete crossover and the fitness functions.

In GAs implementations we consider the cycle of generating the new population with better individuals and restarting the search is repeated until an optimum solution is found. In this process the two fitness evaluation functions have been used. The first fitness function is evaluating the best matrices of the weights population on the basis of the settlement of network in the stable state corresponding to the stored pattern on the presentation of the already stored pattern as the input pattern. The second fitness evaluation function is selecting the weight matrices on the basis of settlement of the network in the stable state corresponding to the correct or exact stored pattern on the presentation of prototype input pattern as the already stored pattern. It indicates that the stable states of the network will be used for the evaluation of the weights population. Thus in the recalling process, stable state of the network corresponding to the stored pattern should be retained for the selected weight vector on the presentation of prototype input pattern.

In regard of two fitness functions, we wish to say that the first fitness evaluation function determines the suitable weight matrices which are responsible to generate the correct recalling of the stored pattern for the input pattern that has been used in the training set. It means that, at the first level of filtering only those weight matrices will be selected which provide the correct pattern association for the training pattern set. Thus, at this level we will not use any test pattern, which involves the noise in the original pattern. It represents only those weight matrices which exhibit the pattern association during the training of the network and should carry in the next generation of population, whereas the second fitness evaluation function is used after the crossover operator. The crossover operator has been applied only those chromosomes which have been passed from the first fitness evaluation function. The second fitness evaluation function has been applied to determine the population of these weight matrices, which are responsible for recalling of the approximate stored pattern on the presentation of test pattern. The test patterns are considered here as noisy prototype patterns of the training set patterns. Thus the second fitness evaluation function is actually selecting the final population of the chromosomes which are required for generating the optimal solution. The parameters used for genetic algorithm are summarized in Table 1.

4.1. The mutation operator

The mutation operator plays a secondary role in the genetic algorithm. Mutation performs the modification of the value of each gene of a solution with some probability p_m. Nevertheless the choice of p_m is critical to GA performance and has been studied by DeJong [30]. The typical value of mutation probability is in the range 0.005–0.05. The idea of adapting mutation and crossover to

Table 1
Parameter used for genetic algorithm.

Parameter/operation/procedure	Suboptimal genetic algorithm		Random genetic algorithm
	$p_m = 0.001$	$p_m = 0.500$	
Chromosome length	$N \times N$	$N \times N$	$N \times N$
Mutation probability	0.001	0.500	1.000
Mutation population size	N	N	N
Crossover population size	$N \times N$	$N \times N$	$N \times N$
Crossover type	Uniform	Uniform	Uniform
Number of fitness functions	2	2	2
Initial population	Suboptimal weight matrix	Suboptimal weight matrix	Random weight matrix

improve the performance of GAs has been used by researchers using the different criterion [31,32]. Whitley and Starkweather [33] idea for adaptation is based on the Hamming distance between solutions; while in Srinivas and Patnaik [34] approach p_m is based on fitness values of the solutions.

The mutation operator produces the population of N weight matrices or chromosomes of same order as the original parent matrix on applying it N times. Thus, each chromosome is having a fixed length of $N \times N$ genes or alleles. In this process of mutation we select randomly any genes, i.e. $s_i^r s_j^r$ from parent chromosome, where r is the position of the gene in the parent chromosome or weight matrix. We consider another randomly generated chromosome of the same order as $N \times N$ in which on the same randomly selected position, i.e. r, the allele values are non zero in the interval −1 and +1. Now, we add this randomly generated chromosome with the parent chromosome and generate the new chromosome or weight matrix as

$$w_{ij}^{new} = s_i(g(r))s_j(h(r)) + A_{ij} \quad \text{and} \quad i = j \quad \text{then} \quad A_{ij} = 0. \tag{4.1}$$

4.2. Elitism

Elitism is used when creating each generation so that the genetic operators do not lose good solutions. This involves copying the Hebbian-encoded weight matrix, i.e. the suboptimal solution unchanged in the new population, which includes $\mathbf{W}^L$ for creating the total number M (i.e. N+1) of chromosomes.

4.3. 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. When one of the stored pattern a^l is given to the network as an initial state, the state of neurons varies over time until a^l becomes 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 [10]

$$z^l = \frac{1}{N}\sum_{i=1}^{N} a_i^l s_i^l(t) \tag{4.2}$$

Here $s_i^l(t)$ is the state of the ith neuron at time t for the lth pattern. In evaluating the fitness value, the temporal average overlap $\langle z^l \rangle$ 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 values are summed up over whole set of initial patterns, i.e. [10]

$$f = \frac{1}{t_0 L}\sum_{t=1}^{t_0}\sum_{l=1}^{L} z^l(t) \tag{4.3}$$

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 crossover operator.

4.4. The crossover operator

The power of GAs arises from crossover. Crossover is a structured and randomized exchange of genetic material between solutions, with the probability that 'good' solutions can generate 'better' ones. This operator is responsible for the recombination of the selected population of weight matrices. Notable crossover techniques include the single point, the two-point, and the uniform types. 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 genes or alleles selected for exchange should be equal for the two weight matrices, the modification has been made in the selected weight matrices as follows:

$$\begin{aligned} & w_{1,N\times N}^{new} = w_{1,old} \cup \left[w_{1,t\times u}^{sel} \leftarrow w_{2,t\times u}^{sel} \right] \\ \text{and} \quad & w_{2,N\times N}^{new} = w_{2,old} \cup \left[w_{2,t\times u}^{sel} \leftarrow w_{1,t\times u}^{sel} \right] \\ & \text{where} \quad t, u \le N \end{aligned} \tag{4.4}$$

Here $w_{1,N\times N}^{new}$ and $w_{2,N\times N}^{new}$ are newly generated weight matrices after crossover operator, $w_{1,old}$ and $w_{2,old}$ are two selected weight matrices randomly from mutated population to perform the crossover operator, $w_{1,t\times u}^{sel}$ and $w_{2,t\times u}^{sel}$ are the selected sub matrices randomly from $w_{1,old}$ and $w_{2,old}$ respectively for exchange to perform crossover operator, and ∪ is the operator for construction of new weight matrix at crossover operator.

In this way, we can generate the population of K weight matrices each of order $N \times N$ by applying the crossover operator on the population of M chromosomes.

Thus, we have the new large population of K weight matrices from the crossover operator as,

$$\left\{ w_{1,N\times N}^{new}, w_{2,N\times N}^{new}, ---, w_{K,N\times N}^{new} \right\} \tag{4.5}$$

Steps for Crossover Operation

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

4.5. The second fitness evaluation

In the process of recalling the stored pattern, corresponding to a noisy letter of the English alphabet, the best suitable weight matrix is selected from the generated population of K weight matrices.

Let the state of the network corresponding to the already stored lth pattern be

$$N(s^l) = \left\{ s_1^l, s_2^l, ---, s_N^l \right\} \tag{4.6}$$

This represents one of the stable states of the network for the memorized lth pattern.

Let the prototype of presented input pattern be $a^l + \varepsilon$. This pattern represents the noisy or distorted form of the already stored pattern a^l in the network, where ε shows the induced error in terms the number of bits, i.e. one-, two-, three, four-, and five-bit. We have the population K of weight matrices after the crossover operation. Now, we select the weight matrices from this population to evaluate its fitness. Let w^k be the kth weight matrix from the generated population K of weight matrices. Now, we assign this selected

Table 2
Results for recalling English alphabets which involve zero-bit error from the stored English alphabets.

Letters	Recalling success (in %)				Letters	Recalling success (in %)			
	Hebbian rule	Sub-optimal GA		Random GA		Hebbian rule	Sub-optimal GA		Random GA
		$p_m = 0.001$	$p_m = 0.50$				$p_m = 0.001$	$p_m = 0.50$	
A	69.20	100	100	100	N	85.50	100	100	100
B	92.30	100	100	100	O	94.40	100	100	100
C	89.90	100	100	100	P	90.80	100	100	100
D	96.40	100	100	100	Q	80.50	100	100	100
E	87.40	100	100	100	R	92.50	100	100	100
F	86.60	100	100	100	S	86.50	100	100	100
G	89.40	100	100	100	T	93.60	100	100	100
H	91.30	100	100	100	U	86.30	100	100	100
I	100.00	100	100	100	V	84.30	100	100	100
J	91.80	100	100	100	W	95.40	100	100	100
K	76.60	100	100	100	X	87.40	100	100	100
L	76.10	100	100	100	Y	89.30	100	100	100
M	88.60	100	100	100	Z	84.20	100	100	100

weight matrix to the network and use the activation dynamics to determine the output state of the network as

$$\left(s_i^l + \varepsilon\right) = \sum_{j=1}^{N} w_{ij}^k \left(s_j^l + \varepsilon\right)(t+1) \tag{4.7}$$

$$\text{if } \left(s_i^l + \varepsilon\right)(t+1) = s_i^l(t); \quad \forall i = 1 : N \tag{4.8}$$

It implies that the network settles in the same stable state which corresponds to the already stored pattern, so that w^k, is selected from the fitness function if it is able to settle the network in the stable state corresponding to the already stored lth pattern.

$$\text{i.e.} \quad N(s^l + \varepsilon) = N(s^l) \tag{4.9}$$

This process will continue for all the weight matrices from the K population. It is possible to obtain more than one optimal weight matrices for the recalling of prototype input pattern.

5. Experiments detail

To do the simulation we are considering the Hopfield neural network of 35 neurons. This neural network is trained for pattern storage with the Hebbian learning rule for given training set of English alphabets. In the training set every pattern consists with 35 bipolar features. The pattern information of the training set is encoded in the form of connection strengths of interconnections between the neurons of the network. In this way the parent weight matrix is constructed which represents the encoded pattern information of memorized patterns.

Now we start the process of recalling for any presented noisy prototype input pattern of already memorized pattern. The process of recalling is accomplished with three different methods namely the Hebbian rule, the sub optimal GA and the random GA. The implementation of above stated methods is performed with experiments to study the performance behavior of these methods.

The recalling through Hebbian rule starts by applying the presented prototype input pattern to the Hopfield network and then we continue to iterate the network till it reaches to stability. The stable state of the network reflects any one of the memorized pattern. In this method the possibility of false recalling or false minima is likely to occur in most of the cases if the presented input pattern is a noisy pattern.

The recalling process through the suboptimal GA can be described in the algorithmic form as

```
Sub Optimal Genetic Algorithm ()
{
read suboptimal parent weight matrix;
initialize input pattern to the network;
      do
      {
          perform mutation and elitism;
          select solutions for next population by first fitness function;
          perform crossover;
          evaluate population by second fitness function;
      }
      while (convergence not achieved OR 20 times)
}
```

Similarly, we can describe the recalling process of the random GA. In this process, we start the recalling through random weight matrix and the pseudo code can be represented as

Table 3
Results for recalling English alphabets which involve one-bit error from the stored English alphabets.

Letters	Recalling success (in %)				Letters	Recalling success (in %)			
	Hebbian rule	Sub-optimal GA		Random GA		Hebbian rule	Sub-optimal GA		Random GA
		$p_m = 0.001$	$p_m = 0.50$				$p_m = 0.001$	$p_m = 0.50$	
A	1.70	100	100	100	N	1.00	100	100	100
B	2.60	100	100	100	O	2.50	100	100	100
C	3.20	100	100	100	P	3.30	100	100	100
D	3.30	100	100	100	Q	2.30	100	100	100
E	3.20	100	100	100	R	2.30	100	100	100
F	2.80	100	100	100	S	1.60	100	100	100
G	2.30	100	100	100	T	2.80	100	100	100
H	2.10	100	100	100	U	3.00	100	100	100
I	2.80	100	100	100	V	2.70	100	100	100
J	3.10	100	100	100	W	2.60	100	100	100
K	1.40	100	100	100	X	2.80	100	100	100
L	1.00	100	100	100	Y	1.20	100	100	100
M	2.50	100	100	100	Z	2.10	100	100	100

Table 4
Results for recalling English alphabets which involve two-bit error from the stored English alphabets.

Letters	Recalling success (in %)				Letters	Recalling success (in %)			
	Hebbian rule	Sub-optimal GA		Random GA		Hebbian rule	Sub-optimal GA		Random GA
		$p_m = 0.001$	$p_m = 0.50$				$p_m = 0.001$	$p_m = 0.50$	
A	0.00	81.25	97.50	92.24	N	0.20	72.24	99.25	95.89
B	0.50	87.19	100.00	96.49	O	0.30	99.50	100.00	98.88
C	0.20	90.97	100.00	97.68	P	0.20	86.86	99.50	94.80
D	0.30	98.77	100.00	99.38	Q	0.00	63.43	98.36	88.83
E	0.20	75.73	95.20	90.09	R	0.20	84.54	97.58	93.38
F	0.20	85.87	99.20	89.62	S	0.20	83.93	88.88	86.41
G	0.30	89.44	100.00	89.62	T	0.20	92.27	98.00	97.30
H	0.10	87.30	99.50	92.31	U	0.10	87.30	99.80	95.44
I	0.40	99.50	100.00	99.50	V	0.00	74.39	90.72	81.92
J	0.10	86.50	95.50	90.24	W	0.00	89.80	93.77	94.58
K	0.00	70.00	90.58	77.34	X	0.30	93.90	96.64	92.36
L	0.20	70.53	93.76	84.74	Y	0.20	86.43	82.50	81.17
M	0.10	82.60	94.40	90.73	Z	0.00	76.12	87.40	80.71

```
Random Genetic Algorithm ()
{
Generate the random weight matrix;
initialize input pattern to the network;
      do
      {
          perform mutation and elitism;
          select solutions for next population by first fitness function;
          perform crossover;
          evaluate population by second fitness function;
      }
      while (convergence not achieved OR 20 times)
}
```

6. The results and discussion

The results presented in this section are demonstrating that, within the simulation framework presented above, large significant difference exists between the performance of genetic algorithm and the Hebbian rule for recalling English alphabets which have been memorized in Hopfield neural network using the Hebbian learning rule.

The results is also indicating the difference in the recalling successes of sub optimal GA and random GA. In the experiments, the prototype input patterns used for recalling purpose consists the error which has been generated randomly with respect to memorized patterns. Tables 2–7 show the results of recalling the patterns which containing zero-, one-, two-, three-, four-, and five-bit errors from stored patterns in the Hopfield neural network. The performance of sub optimal GA and random GA is same corresponding to zero-, and one-bit error in prototype input patterns. The results clearly indicate that the Hebbian rule works well for a noiseless pattern, for most of the cases, but its performance degrades substantially and recalling success goes down to a maximum of 3.30% in the case of one-bit error, 0.50% in the case of two-bit error, 0.01% in the case of three-bit error, and 0.000% in the cases of four- and five-bit errors. On the other hand, both GAs recall the pattern successfully even when high noise is presented in the input test pattern, i.e. four-bit and five-bit errors. It is observed that in most of the cases the performance of the suboptimal GA outperform the random GA. It is also observed that variance in the mutation probability in sub optimal GA does not have any substantial impact in the performance of this algorithm.

Amit [3] claimed that the capacity of deterministic Hopfield model with the Hebbian rule is about $0.15N$ for the noisy prototype input patterns, where N is the number of nodes in the network. If such a network is overloaded with a number of patterns exceeding its capacity, its performance rapidly deteriorates toward zero. Here, we are storing the 26 alphabets in a network of 35 nodes and the performance of the GA suggests that on inducing 5-bit error in presented prototype input pattern the network is able to recall the stored patterns for both the GAs. It implies that the network capacity has increased up to $0.75N$. Thus, the numbers of attractors are existing here and successfully explored during the recalling process. It is quit obvious to understand that the GA has searched the suitable optimal weight matrices which are responsible to generate sufficiently large number of attractions. Hence, the Hebbian rule which has been used to encode the pattern information is not the optimal weight matrix for finding the global minima of the problem due to the limited capacity of the Hopfield model. Thus capacity has

Table 5
Results for recalling English alphabets which involve three-bit error from the stored English alphabets.

Letters	Recalling success (in %)				Letters	Recalling success (in %)			
	Hebbian rule	Sub-optimal GA		Random GA		Hebbian rule	Sub-optimal GA		Random GA
		$p_m = 0.001$	$p_m = 0.50$				$p_m = 0.001$	$p_m = 0.50$	
A	0.00	40.95	38.40	21.63	N	0.00	62.20	61.68	39.03
B	0.00	76.50	83.73	36.90	O	0.00	90.29	88.12	55.91
C	0.00	77.95	84.94	45.21	P	0.00	71.33	72.05	40.37
D	0.00	90.09.	90.21	44.54	Q	0.00	48.33	56.75	24.50
E	0.00	59.40	71.04	29.01	R	0.00	70.57	62.75	34.81
F	0.00	66.67	66.40	28.11	S	0.00	69.37	59.83	37.93
G	0.00	77.53	80.05	46.01	T	0.00	80.74	59.50	51.23
H	0.00	72.50	74.20	34.56	U	0.00	69.30	72.44	35.30
I	0.00	95.30	100.00	61.56	V	0.00	52.94	54.29	35.61
J	0.00	70.39	54.23	46.83	W	0.00	78.30	72.56	36.77
K	0.00	45.02	54.88	28.71	X	0.01	72.95	49.44	42.46
L	0.00	44.04	59.57	39.38	Y	0.00	56.41	40.91	41.31
M	0.00	60.75	62.11	27.23	Z	0.00	52.03	45.93	29.01

Table 6
Results for recalling English alphabets which involve four-bit error from the stored English alphabets.

Letters	Recalling success (in %)				Letters	Recalling success (in %)			
	Hebbian rule	Sub-optimal GA		Random GA		Hebbian rule	Sub-optimal GA		Random GA
		$p_m = 0.001$	$p_m = 0.50$				$p_m = 0.001$	$p_m = 0.50$	
A	0.00	6.70	11.66	1.13	N	0.00	14.90	11.63	3.54
B	0.00	20.24	23.56	4.50	O	0.00	27.10	19.92	2.07
C	0.00	22.62	20.54	6.00	P	0.00	18.63	13.33	3.35
D	0.00	29.94	19.94	8.38	Q	0.00	11.16	8.00	2.75
E	0.00	16.04	13.15	5.75	R	0.00	19.04	11.83	3.40
F	0.00	16.71	12.08	4.75	S	0.00	17.74	11.04	5.59
G	0.00	19.68	19.74	5.50	T	0.00	21.66	9.60	8.05
H	0.00	19.77	16.41	4.41	U	0.00	21.28	12.39	2.80
I	0.00	29.23	15.93	10.30	V	0.00	14.14	7.80	3.50
J	0.00	20.41	9.47	5.20	W	0.00	25.50	13.50	4.23
K	0.00	8.81	7.75	2.75	X	0.00	22.95	4.80	4.60
L	0.00	10.61	8.88	5.33	Y	0.00	13.55	6.95	10.01
M	0.00	19.13	14.03	5.10	Z	0.00	12.90	5.90	4.39

Table 7
Results for recalling English alphabets which involve five-bit error from the stored English alphabets.

Letters	Recalling success (in %)				Letters	Recalling success (in %)			
	Hebbian rule	Sub-optimal GA		Random GA		Hebbian rule	Sub-optimal GA		Random GA
		$p_m = 0.001$	$p_m = 0.50$				$p_m = 0.001$	$p_m = 0.50$	
A	0.00	0.00	1.00	0.00	N	0.00	2.59	0.30	0.58
B	0.00	1.00	2.00	0.30	O	0.00	2.64	2.33	1.07
C	0.00	1.70	1.70	0.28	P	0.00	2.90	1.70	0.54
D	0.00	2.00	3.00	0.20	Q	0.00	1.50	0.80	0.14
E	0.00	0.50	2.00	0.10	R	0.00	2.85	1.54	0.59
F	0.00	2.50	0.50	0.17	S	0.00	2.63	1.00	0.53
G	0.00	3.50	1.50	0.34	T	0.00	2.96	2.52	0.56
H	0.00	3.70	0.08	1.31	U	0.00	3.50	1.01	0.30
I	0.00	4.50	1.50	1.02	V	0.00	4.00	1.04	0.40
J	0.00	3.50	0.00	0.80	W	0.00	4.00	2.00	0.70
K	0.00	1.00	0.70	0.00	X	0.00	3.00	0.40	0.56
L	0.00	1.80	2.50	0.25	Y	0.00	1.50	0.50	0.97
M	0.00	2.61	1.90	0.60	Z	0.00	1.70	0.40	0.25

been increased with GA by exploring the optimal weight matrices for the encoded patterns.

The simulation program, which is developed in MATLAB-7, to test the Hebbian rule, the suboptimal GA, and the random GA for the recalling of English alphabets, stores the patterns in the Hopfield neural network of 35 neurons. It is to note that during suboptimal GA and random GA, the success is considered only if the recalling is done within 20-iterations.

7. Conclusion

The simulation results, i.e. Tables 2–7 indicate that the sub optimal genetic algorithm has more success rate than the random GA and the Hebbian rule for the stated problem. It has been found that both genetic algorithms, i.e. suboptimal GA and random GA can give more than one convergent weight matrices for any prototype input patterns in comparison to the Hebbian rule, if the prototype input pattern is correctly recognized. This shows the more chances for GAs to explore better solution than the Hebbian rule. Further the sub optimal GA is starting from sub optimal weight matrix so it has more chances to explore more number of convergent weight matrices with respect to random GA. It might be the reason that the performance of suboptimal GA is found better than the random GA. In the purposed method it can be seen that the two fitness evaluation functions are used. There are two basic advantages of the two fitness evaluation functions:

1. The randomness of the GA has minimized, because the population is filtered twice. Hence, the less number of populations will be generated and the generated population will be more fitted for the solution.
2. As the number of population has minimized, the searching time will also be reduced. Thus, the GA has improved in its implementation because it is less random and consuming less time for searching the optimal solution.

The direct application of GA to the pattern association has been explored in this research. The aim is to introduce an alternative approach to solve the pattern association problem. The results from the experiments conducted on the algorithm are quite encouraging. Nevertheless more work needs to be perform especially on the tests for noisy input patterns. We can also use this concept for pattern recognition in the case of different objects, shapes, numerals and overlapped alphabet, etc. It is also proposed to undertake the following problems in future research program.

1. We would like to train the Hopfield neural network with the genetic algorithm and then the pattern recalling with genetic algorithm.
2. We would like to apply the same approach on the quantum Hopfield neural network.

References

[1] J.J. Hopfield, Neural networks physical systems with emergent collective computational abilities, Proceedings of the National Academy Sciences, USA 79 (1982) 2554–2558.

[2] J.J. Hopfield, Neural Networks, Physical systems with emergent collective computational abilities, Proceedings of the National Academy Sciences, USA 81 (1984) 3088–3092.
[3] D.J. Amit, H. Gutfreund, H. Somopolinsky, Storing infinite number of patterns in a spin-glass model of neural networks, Physical Review Letters 55 (14) (1985) 461–482.
[4] D.J. Amit, Modeling Brain Function: The World of Attractor Neural Networks, Cambridge University Press, New York, NY, USA, 1989, p. 58 (Chapter 2).
[5] S. Haykin, Neural Networks: A Comprehensive Foundation, Prentice Hall, Upper Saddle River, 1998, p. 64 (Chapter 14).
[6] Z. Zhou, H. Zhao, Improvement of the Hopfield neural network by MC-adaptation rule, Chinese Physics Letters 23 (6) (2006) 1402–1405.
[7] H. Zhao, Designing asymmetric neural networks with associative memory, Physical Review 70 (6) (2004) 066137–66144.
[8] M. Kawamura, M. Okada, Transient dynamics for sequence processing neural networks, Journal of Physics A: Mathematical and General 35 (2) (2002) 253.
[9] D.J. Amit, Mean-field Ising model and low rates in neural networks, in: Proceedings of the International Conference on Statistical Physics, 5–7 June, Seoul, Korea, 1997, pp. 1–10.
[10] A. Imada, K. Araki, Genetic algorithm enlarges the capacity of associative memory, in: Proceedings of the Sixth International Conference on Genetic Algorithms, 1995, pp. 413–420.
[11] J.J. Hopfield, D.W. Tank, Neural computation of decisions in optimization problems, Biological Cybernetics 52 (3) (1985) 141–152.
[12] D.W. Tank, J.J. Hopfield, Simple neural optimization networks: an A/D converter, signal decision circuit, and a linear programming circuit, IEEE Transactions on Circuits and Systems 33 (5) (1986) 533–541.
[13] T. Jin, H. Zhao, Pattern recognition using asymmetric attractor neural networks, Physical Review E 72 (6) (2005) 066111–66117.
[14] S. Kumar, M.P. Singh, Pattern recall analysis of the Hopfield neural network with a genetic algorithm, Computers and Mathematics with Applications 60 (4) (2010) 1049–1057.
[15] D. Hebb, The Organization of Behavior: A Neuropsychological Theory, Wiley, New York, 1949.
[16] J.H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, Michigan, 1975.
[17] M. Mangal, M.P. Singh, Analysis of multidimensional XOR classification problem with evolutionary feed-forward neural networks, International Journal on Artificial Intelligence Tools 16 (1) (2007) 111–120.
[18] M. Mangal, M.P. Singh, Analysis of classification for the multidimensional parity-bit-checking problem with hybrid evolutionary feed-forward neural network, Neurocomputing 70 (7–9) (2007) 1511–1524.
[19] M. Mangal, M.P. Singh, Handwritten English vowels using hybrid evolutionary feed-forward neural network, Malaysian Journal of Computer Science 19 (2) (2006) 169–187.
[20] M. Paliwal, U.A. Kumar, Review: neural networks and statistical techniques: a review of applications, Expert Systems with Applications 36 (1) (2009) 2–17.
[21] R.J. Kuo, K. Chang, S.Y. Chien, Integration, Self-organizing feature maps and genetic-algorithm-based clustering method for market segmentation, Journal of Organizational Computing and Electronic Commerce 14 (1) (2004) 43–60.
[22] Q. Cao, M.E. Parry, Neural network earnings per share forecasting models: a comparison of backward propagation and the genetic algorithm, Decision Support Systems 47 (1) (2009) 32–41.
[23] S.K. Pal, S. De, A. Ghosh, Designing Hopfield type networks using genetic algorithms and its comparison with simulated annealing, International Journal of Pattern Recognition and Artificial Intelligence 11 (3) (1997) 447–461.
[24] J. Xu, J. He, X. Yao, Solving equations by hybrid evolutionary computation techniques, IEEE Transactions on Evolutionary Computation 4 (3) (2000) 295–304.
[25] S. Salcedo-Sanz, X. Yao, A hybrid Hopfield network-genetic algorithm approach for the terminal assignment problem, IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics 34 (6) (2004) 2343–2353.
[26] A.H.M. Amin, R.A.R. Mahmood, A.I. Khan, Analysis of pattern recognition algorithms using associative memory approach: a comparative study between the Hopfield network and distributed hierarchal graph neuron (DHGN), in: IEEE 8th International Conference on Computer and Information Technology Workshops (CIT Workshop-2008), 2008, pp. 153–158.
[27] M. Blumenstien, X.Y. Liu, B. Verma, An investigation of the modified direction feature for cursive character recognition, Pattern Recognition 40 (2) (2007) 376–388.
[28] X. Yao, Evolving artificial neural networks, Proceeding of the IEEE 87 (9) (1999) 1423–1447.
[29] P.D. Wasserman, Neural Computing: Theory and Practice, Van Nostrand Reinhold Co, New York, NY, USA, 1989.
[30] K.A. DeJong, An analysis of the behavior of a class of genetic adaptive systems, Ph.D. dissertation, University of Michigan, 1975.
[31] L. Davis, Adapting operator probabilities in genetic algorithms, in: Proceedings of Third International Conference on Genetic Algorithms, 1989, pp. 61–69.
[32] T.C. Fogarty, Varying the probably of mutation in genetic algorithms, in: Proceedings Third International Conference on Genetic Algorithms, 1989, pp. 104–109.
[33] D. Whitley, D. Starkweather, Genitor-I.I. A distributed genetic algorithm, Journal of Experimental & Theoretical Artificial Intelligence 2 (3) (1990) 189–214.
[34] M. Srinivas, L.M. Patnaik, Adaptive probabilities of crossover and mutation in genetic algorithms, IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics 24 (4) (1994) 656–667.