Pattern Recall Analysis of the Hopfield Neural Network with a Genetic Algorithm

Susmita Mohapatra

Department of Computer Science, Utkal University, India

Abstract: ***This paper is focused on the implementation of Hybrid techniques to explore the optimal solution for real world pattern association problems for which already some solutions exist but they are not efficient to obtain the desired solution for the problem as we can get by using the neural network techniques with genetic algorithm. In present work we have made some comparisons between the existing conventional and proposed evolutionary techniques and carried out the analysis. The results recommend that, in all cases, recalling of any approximate pattern through genetic algorithm outperform the recalling of the same pattern through conventional Hebbian rule.***

Keywords: Pattern Recall Analysis, Hopfield Neural Network, Genetic algorithm, Hebbian Learning Rule

1. Introduction

Pattern recognition is the study of how machines can observe the environment, learn to distinguish patterns of interest from their background, and make sound and reasonable decisions about the categories of patterns. In spite of more than 50 years of research, design of a general-purpose machine pattern recognizer remains an elusive goal.

The inherent differences in information handling by human beings and machines in the form of patterns and data, and their functions in the form of understanding and recognition have led us to identify and discuss several pattern recognition tasks which human beings are able to perform very naturally and effortlessly whereas we have no simple algorithms to implement these tasks on a machine.

Neuro-computing concerns with processing of information with its adaptation. Unlike its programmed computing counterpart, a neuro-computing approach to information processing first involves a learning process within an artificial neural network (neuro-computers) or neural network architecture that adaptively responds to inputs according to a learning rule. Then, the trained neural network can be used to perform certain tasks depending on the particular application.

2. Motivation and Problem Definition

As per the Hopfield analysis for pattern storage networks on presentation of any prototype input pattern or the noisy form of any stored pattern, the network is expected to recall the corresponding stored pattern. But this cannot happen most of the time due to problem of false minima if the size of the network is large. It becomes more difficult when the test patterns are overlapped version of the patterns used in the training process. The ultimate goal is to impart a machine with pattern recognition capabilities comparable to those of human beings. This goal is difficult to achieve using most of the conventional methods. It is for these reasons that the new models of computation inspired by the structure and function of the biological neural network are continuously evolved. Such models for computing are based on Artificial Neural Networks and Genetic Algorithms. In present thesis, our objective is to enhance the performance of Hopfield neural network, especially the capacity and the quality of the storing, by making use of genetic algorithm.

Therefore, to accomplish the above objectives, firstly the patterns of training set have been encoded in the neural network using conventional Hebbian learning rule. It is expected that all the patterns of training set has been successfully stored as the associative memory feature of Hopfield neural network. As a result of this learning process, the expected optimized or sub-optimized weight matrix has been obtained and then the genetic algorithm has been employed to further optimize this weight matrix. In case of genetic algorithm, the population of this approximate optimal weight matrix has been evolved using the population generation technique, crossover operator and fitness evaluation functions, until the selection of the last weight matrix or matrices has been performed. The performance of this network is further improved for the efficient recalling by evolving the obtained weight matrix with the genetic algorithm. The results of various methods for recalling has been compared and analyzed.

3. Pattern Recall Analysis using Hopfield Neural Network with Genetic Algorithm

Artificial neural network (ANN) is a technique for creating artificial intelligence in the machine. This is an attempt of the modeling of the human brain in a serial machine for various pattern recognition tasks. Pattern storage is one of the techniques for the pattern recognition task that one would like to realize using an ANN. The Hopfield neural network is a simple feedback neural network 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. Furthermore there is a degree of stability in the system – if just a few of the connections between nodes are severed, the recalled pattern is not too badly corrupted and the network can respond with a *best guess*. Of course, a similar phenomenon is observed with the brain. Here the network is expected to store the pattern information (not data) for later recall. Pattern storage is generally accomplished by a feedback network consisting of processing units with non-linear bipolar output functions. The stable states of the network represent the stored patterns.

Neural networks are often used for pattern recognition and classification. Hopfield 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, 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, an evolutionary searching (genetic algorithm) 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 offspring of the fit individuals. The process of evaluation, selection, and recombination is iterated until the population converges to an acceptable solution.

Much work has been done on the evolution of neural networks with GA. 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.

The population generation techniques (mutation and elitism) are used in the parent weight matrix of feedback neural network for evolving the population of weights after storing all English alphabets using Hebbian learning rule. The generated population (by using mutation and elitism) of the weights is evaluated from the first fitness evaluation function. The fixed-point stability is used as first fitness evaluation function for the evaluation of individual weight population. A Crossover operator takes first fittest weights as parents and produces the next generation population of weights as children. The stable state function is used as second fitness evaluation function on crossover generated population of weights for evaluation of individual weight population. This process is continued until the optimal weight matrices are found for recalling of already stored patterns. The evolution of a network's connection weights is an area of curiosity and the centre of attention of this work.

his thesis is organized as follows: First, we present the simulation design, and implementation details of the problem. Secondly, a numerical example provides the details of the example situation for model validation and verification. Then, the experimental results and descriptions are presented. In the end, we conclude the work and suggest new direction for future research.

3.1. Simulation Design and Implementation Details

The experiments described in this segment have been designed to evaluate the performance of a Hopfield neural network with a genetic algorithm for the recalling of already stored English alphabets.

3.1.1. Set of patterns used for training

The patterns used for the simulation 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.

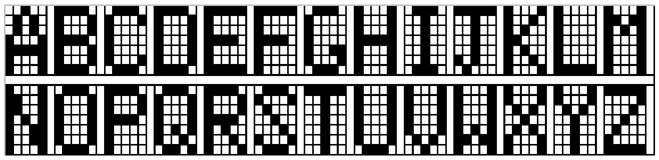
Figure 1: Set of Patterns Used for Training

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 l th pattern vector as

$$x^l = [\alpha_1^l, \alpha_2^l, \text{--------------------------}, \alpha_{35}^l] \qquad (1)$$

where l =1 to 26 and $\alpha_1, \alpha_2, \ldots \ldots \ldots \ldots, \alpha_{35}$ are neurons.

3.1.2. Experiments

Four runs of the experiments were taken on same Hopfield network architecture, i.e., 35 neuron's network. Each run is based on one of the two experiments - recalling English alphabets with Hebbian rule and recalling the same alphabet with genetic algorithm. The inputs for four different runs are zero-, one-, two-, and three-bit errors induced randomly in the pattern already stored in the network. In each experiment, the Hebbian learning rule is used to store patterns in the Hebbian neural network. The genetic operators used in each experiment are summarized in Table 1.

Table 1: Genetic operators used in experiments

Training Algorithms	*Genetic Operator Used*
Hebbian rule	None
Genetic algorithm	Population generation technique (mutation + elitism), crossover and fitness evaluation technique

The parameters used in different runs of the experiments are described in Table 2 and Table 3.

Table 2: Parameters used for Hebbian learning rule

Parameter	*Value*
Initial state of neurons	Randomly generated values either -1 and +1
Threshold values of neurons	0.00

Table 3: Parameters used for Genetic Algorithm

Parameters	*Value*
Initial state of neurons	Randomly generated values either -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 English alphabets as input patterns with the appropriate recalling of the same patterns with induced noise.

3.2 The Hopfield Neural Network

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

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

$$w_{ij}=\frac{1}{N}\sum_{l=1}^{L} x_i^l\ x_j^l\ (i \neq j) \text{ and } w_{ii}=0 \qquad (2)$$

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.

$$f\left(\sum_j w_{ij}\ s_j\right) = s_i \text{ ; where } i = 1, 2, 3 \ldots \ldots \ldots \ldots \ldots \ldots, N. \quad (2)$$

Let the pattern set be $X^L = \{ x^1, x^2, \ldots \ldots \ldots, x^L \}$ (3)

Where $[x^1 = (\alpha_1^1, \alpha_2^1, \ldots \ldots \ldots \ldots \ldots \ldots \ldots, \alpha_N^1)$,

$x^2 = (\alpha_1^2, \alpha_2^2, \ldots \ldots \ldots \ldots \ldots \ldots \ldots, \alpha_N^2)$,

.

.

$x^L = (\alpha_1^L, \alpha_2^L, \ldots \ldots \ldots \ldots \ldots \ldots \ldots, \alpha_N^L)$.

with $N = 1, 2, 3, \ldots \ldots \ldots \ldots \ldots \ldots, 35$

and $L = 1, 2, \ldots \ldots \ldots \ldots \ldots \ldots \ldots, 26]$. (4)

Now, the initial weights have been considered as $w_{ij} \approx 0$ (near to zero) for all i's and j's from the synaptic dynamics we have:

$$w_{ij}^{new} = w_{ij}^{old} + X_i^1 X_j^1 \qquad (5)$$

$$\text{or } w_{ij}^{new} = w_{ij}^{old} + \sum_{i,j} s_i\ s_j \text{ [initialize } s_i = x_i \text{ for all i]} \quad (6)$$

$$\text{and } w_{ij}^{old} = w_{ij}^{new} \qquad (7)$$

Similarly for the Lth pattern

$$w_{ij}^L = w_{ij}^{L-1} + \sum_{i,j} S_i^L S_j^L \qquad (8)$$

We can generalize this as

$$w_{ij}^L = \frac{1}{N}\sum_{l=1}^{L}\sum_{i,j} S_i^l S_j^l \qquad (9)$$

or in the vector form $W^L = \frac{1}{N}\sum_{l=1}^{L} S^l (S^l)^T$ (10)

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

$$W^L = \begin{pmatrix} 0 & S_1S_2 & S_1S_3 & \cdots\cdots\cdots & S_1S_N \\ S_2S_1 & 0 & S_2S_3 & \cdots\cdots\cdots & S_2S_N \\ | & | & | & | & | \\ S_NS_1 & S_NS_2 & S_NS_3 & \cdots\cdots\cdots & 0 \end{pmatrix} \qquad (11)$$

This square matrix is known as parent weight matrix for storing the given input patterns. Hopfield suggested that the maximum limit for the storage is 0.139 N in network with N neurons, if a small error in recalling is allowed. Later, this was theoretically calculated as p = 0.14 N by using the replica method.

The Hebbian rule, which we are using for recalling letters of English alphabets, can be defined diagrammatically in Figure as

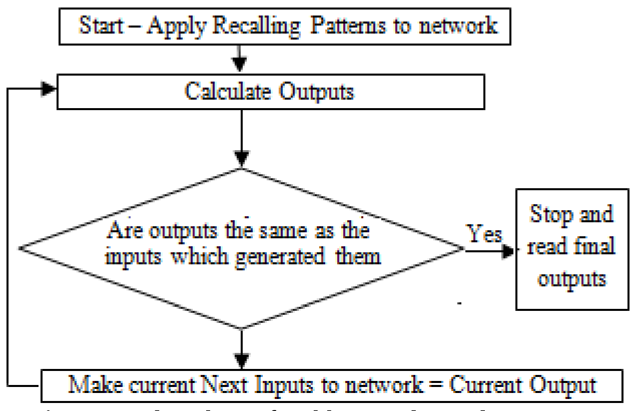
Start – Apply Recalling Patterns to network
Calculate Outputs
Are outputs the same as the inputs which generated them
Yes
Stop and read final outputs
Make current Next Inputs to network = Current Output

Figure 2: Flowchart of Hebbian Rule Implementation

3.3. The Genetic Algorithm Implementation

In this simulation, we are not storing the GAs with random solutions, as it generally starts; instead we start with a sub-optimal solution to achieve the optimal solution. In each iteration, this problem of sub-optimal solution is modified through uniform random mutations and discrete crossovers and their fitness values are evaluated. According to the fitness values, individuals of the next generation are selected using a strategy in ES terminology. The cycle of reconstructing 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 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. 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 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 only represents those weights which exhibit the pattern association during the training of the network and should carry in the next generation of the population, whereas the second fitness evaluation function is used after the crossover operator. The crossover operator has been applied only on the 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 Evolutionary Algorithm, which we are using for recalling the letters of English alphabets, can be defined diagrammatically in Figure 6 as

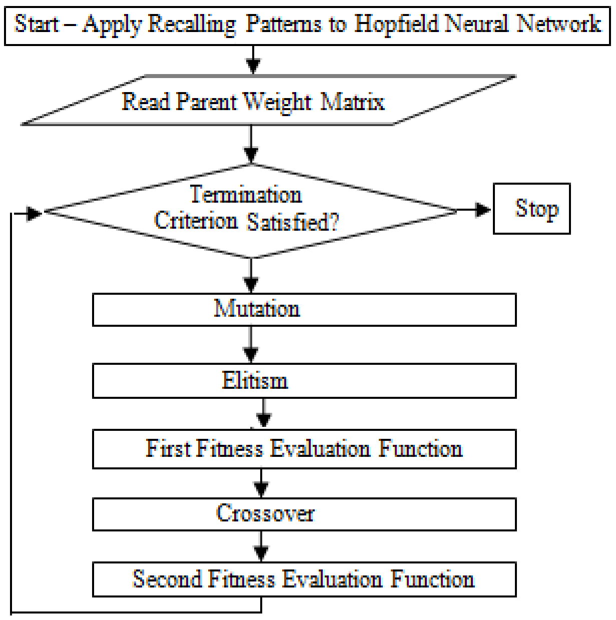
Start – Apply Recalling Patterns to Hopfield Neural Network
Read Parent Weight Matrix
Termination Criterion Satisfied?
Stop
Mutation
Elitism
First Fitness Evaluation Function
Crossover
Second Fitness Evaluation Function

Figure 3: Flowchart of Genetic Algorithm Implementation

3.3.1. The Population Generation Technique

The population generation technique produces the population of N weight matrices of 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 is having a fixed length of N × 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^L and produces N weight matrices slightly different from W^L. The modification can be represented as

$$w_{ij}^n = S_iS_j\, A^n\,(N.i + j)\ (i, j = 1, 2,\ldots,N),\ (n=1, 2, \ldots, M) \quad (12)$$

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

3.3.2. The Pseudo-Code of the Population Generation Technique

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

0	S_1S_2	….	S_1S_N	S_2S_1	….	S_NS_1	S_NS_2	S_NS_3	….	0

Figure 4: Chromosome Representation

3.3.3. The First Fitness Evaluation

The first fitness evaluation function (f) is used for selecting the good or efficient next generation of weight matrices. Evaluation of *f* for each individual weight matrix is made with a set of randomly pre-determined patterns X^L. when one of the stored patterns X^L is given to the network as an initial state, the state of neurons varies over time until X^L 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

$$z^L = \frac{1}{N} \sum_{i=1}^{N} x_i^L S_i^L\,(t) \quad (13)$$

Here $S_i^L\,(t)$ is the state of the ith neuron at time t. In evaluating the fitness value, the temporal average overlap (z^L) 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.

$$f = \frac{1}{t_0 L} \sum_{t=1}^{t_0} \sum_{l=1}^{L} z^L\,(t) \quad (14)$$

Here t_0 has been set to N (the number of processing units). We must note that the 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. These new population will be used for generating the next better population of weight matrices with the recombination or crossover operator.

3.3.4. The Crossover Operator

Crossover is an operation which may be used to combine multiple parents and make offsprings. 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 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 number 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:

$$\sum_r^{N\times N} w_{ij}^n = \sum_r^{N\times N} S_i S_j \; A^k \,(N.i + j)$$
$$\text{and} \quad \sum_r^{N\times N} w_{ij}^n = \sum_r^{N\times N} S_i S_j \; A^n \,(N.i + j) \qquad (15)$$
$$(i, j = 1,\ 2,\ \dots\ \dots\ \dots\ \dots,\ N;\ k, n = 1,\ 2,\ \dots\ \dots\ \dots,\ T;\ n \neq k)$$

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

$$\{w_{1\,N\times N}^{new},\ w_{2\,N\times N}^{new},\ \text{-----------------------},\ w_{k\,N\times N}^{new}\} \qquad (16)$$

3.3.5. The Steps for 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 step2 again.*

3.3.6. The Second Fitness Evaluation

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

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

$$N(S^l) = \{S_1^l, S_2^l, \dots\dots\dots\dots\dots\dots\dots\dots.\,.\,S_n^l\} \qquad (17)$$

This represents one of the stable states of the network which is occupying the stored pattern L during the learning.
Let the prototype of presented input pattern be $x^{l+€}$. This pattern represents the noisy or distorted form of the already stored pattern x^l. We have the population of K weight matrices after the crossover operation. Now, we start selecting the weight matrices from this population to evaluate it with this fitness function. Let $W^{POP.k}$ be the kth weight matrix from the generated population of weight matrices. Now, we assign this selected matrix to the network and use the activation dynamics to determine the output state of the network as

$$S_i^{l+} = \sum_{j=1}^{N} W_{ij}^{POP.k}\; S_j^{l+}\,(t+1) \qquad (18)$$
$$\text{If } S_j^{l+}\,(t+1) = S_i^{l}\,(t) \quad \text{for all } i = 1 \text{ to } N. \qquad (19)$$

It implies that the network settles in the same stable state which corresponds to the already stored pattern so that the $W^{POP.k}$ has been selected from the fitness function if it is able to settle the network in the stable state of the already stored pattern.

$$\text{i.e.} \qquad N\,(S^{l+€}) = N\,(S^l) \qquad (20)$$

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

4. A Numerical Example

Here, we are analyzing the performance of the proposed method for pattern recalling with the help of an example. For this, we consider the 26 capital letters of English alphabet (A - Z), as represented in Fig 1, have been encoded in a 35 nodes Hopfield neural network using the Hebbian learning rule shown in section 3.1.1. The encoded patterns construct a sub-optimal weight matrix of order 35, which is as represented by equation. Now, the prototype of these letters without noise and with noise – are presented to the network and attempt to recall is made by both hebbian rule and genetic algorithm.

The letters are represented in the form of bipolar values either +1 or -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]

Here, the noise means reverting a particular value from +1 to -1 and vice-versa. For example, if a noise of two-bit is introduced in the above pattern of letter A at position 3 and 5, it becomes:

[-1-1-1-11-11-11-1-11-11-11-1-1-11111111-1-1-111-1-1-11]

This prototype pattern has been attempted to recall by Hebbian rule, the flow graph for which is shown in Fig 2. The result shows that the success of correct recall of letter A with two-bit induced error using Hebbian rule is 0%.

The same prototype pattern has been attempted to recall by the genetic algorithm, the flow graph for which is shown in Fig 3. The GA implementation starts with the sub-optimal weight matrix (parent weight matrix W^L) obtained after storing all the patterns using hebbian rule. The first operator applied is mutation. During this operation, we obtain 35 weight matrices of order 35 slightly different from parent weight matrix W^L using equation 12. By virtue of next operator elitism, the parent weight matrix is also included in the generated population of matrices. It makes the total population of weight matrices 36 i.e. 35+1. Now for the survival of fittest, a first fitness function is applied to the above generated population of 36 weight matrices. This fitness function passes through only those weight matrices through which all stored patterns can correctly be recalled using equation 13 and 14. Only 30 such weight matrices are qualified to be included in the next generation. The next operator, crossover, after randomly selecting two weight matrices from 30 weight matrices, applies the crossover operator to generate next population weight matrices using the steps described for crossover operator in section 3.3.5. The limit of crossover population is 1225.

The second fitness function filters in those weight matrices (out of this 1225 matrices) through which the prototype presented is correctly recalled using hebbian rule. Four weight matrices were qualified through this fitness function which suggests that out of 1225 matrices only 4 weight matrices, the network is converging for correct recall of a two-bit noise induced prototype pattern of letter A. During the GA implementation, it is not necessary that we get some weight matrices, through which correct recalling is made, in the first iteration only, instead it may take more than one iterations. If it happens within 20 iterations, the success is count; otherwise, we consider it as failure. In the particular example of letter A with 2-bit induced noise, correct recalling could be made in the 5th iteration. The result shows that the success of correct recall of letter A with two-bit induced error using GA is 100%.

5. Results and Discussion

The results presented in this section have demonstrated that, within the simulation framework presented above, large significant difference exist between the performance of the genetic algorithm and the conventional Hebbian rule for recalling the letters of English alphabets which have been stored in the Hopfield neural network using the Hebbian learning rule. These results recommend that, in all cases, recalling of any approximate pattern through genetic algorithm outperform the recalling of the same pattern through conventional Hebbian rule.

Table 4: Results for recalling letters of English alphabet with no error

Letters	*Recalling success in (%)*		*Letters*	*Recalling success in (%)*	
	Hebbian rule	*GA*		*Hebbian rule*	*GA*
A	69.2	100	N	85.5	100
B	92.3	100	O	94.4	100
C	89.9	100	P	90.8	100
D	96.4	100	Q	80.5	100
E	87.4	100	R	92.5	100
F	86.6	100	S	86.5	100
G	89.4	100	T	93.6	100
H	91.3	100	U	86.3	100
I	100	100	V	84.3	100
J	91.8	100	W	95.4	100
K	76.6	100	X	87.4	100
L	76.1	100	Y	89.3	100
M	88.6	100	Z	84.2	100

Table 4 has the results for recalling the stored letters of English alphabets using both the Hebbian rule and the genetic algorithm, while there is no noise present in the input pattern. In total 5000 times the recalling was made through both the algorithms separately for each letter. During GA implementation for single recall, the success is considered only if the recalling of letter I made within 20 iterations, i.e. mutation, elitism, first fitness evaluation function, crossover and second fitness evaluation function.

Table 5-7 represent the results for recalling the corresponding stored patterns while these are presented with induced noise. In these cases, noise was created by reverting one-, two-, and three-bits in the presented prototype input patterns of the already stored patterns. These positions of the bit(s) to be reverted to create noise are taken randomly.

Table 5: Results for recalling letters of English alphabet with one-bit error

Letters	*Reverted*	*Recalling success in (%)*		*Letter*	*Reverted*	*Recalling success in (%)*	
		Hebbian rule	*GA*			*Hebbian rule*	*GA*
A	1	4.1	100	N	35	1.2	100
B	1	4.8	100	O	2	1.6	100
C	4	2.6	100	P	5	3.1	100
D	1	3.8	100	Q	7	2.0	100
E	1	3.3	100	R	10	1.9	100
F	3	2.8	100	S	20	1.6	100
G	2	3.0	100	T	6	2.1	100
H	5	2.7	100	U	1	4.5	100
I	35	2.2	100	V	2	1.9	100
J	1	4.8	100	W	35	1.1	100
K	14	1.8	100	X	10	2.8	100
L	17	2.2	100	Y	33	2.6	100
M	31	1.7	100	Z	2	2.5	100

Table 6: Results for recalling letters of English alphabet with two-bit error

Letters	*Reverted*	*Recalling success in (%)*		*Letter*	*Reverted*	*Recalling success in (%)*	
		Hebbian rule	*GA*			*Hebbian rule*	*GA*
A	(1,2)	0.02	100	N	(1,34)	0.02	100
B	(1,4)	0.04	100	O	(28,31)	0.02	100
C	(2,32)	0.02	100	P	(13,20)	0.02	100
D	(1,20)	0.05	100	Q	(8,13)	0.04	100
E	(3,34)	0.02	100	R	(3,33)	0.04	100
F	(3,31)	0.01	100	S	(2,35)	0.02	100
G	(2,4)	0.06	100	T	(1,3)	0.02	100
H	(10,15)	0.04	100	U	(8,14)	0.01	100
I	(2,34)	0.05	100	V	(1,4)	0.02	100
J	(8,10)	0.02	100	W	(2,4)	0.04	100
K	(1,35)	0.02	100	X	(1,5)	0.05	100
L	(1,6)	0.04	100	Y	(5,8)	0.02	100
M	(3,6)	0.03	100	Z	(1,3)	0.02	100

Table 7: Results for recalling letters of English alphabet with three-bit error

Letter	*Reverted*	*Recalling success in (%)*		*Letters*	*Reverted*	*Recalling success in (%)*	
		Hebbian rule	*GA*			*Hebbian rule*	*GA*
A	(16,19,22)	0.00	72.5	N	(1,5,35)	0.00	90.7
B	(1,3,7)	0.01	90.2	O	(4,7,10)	0.01	81.5
C	(3,7,10)	0.00	74.2	P	(6,10, 12)	0.00	78.8
D	(3,10,4)	0.00	78.3	Q	(9,12, 16)	0.00	75.5
E	(2,4,6)	0.00	79.9	R	(31,32,35)	0.00	9.30
F	(6,18, 21)	0.00	69.9	S	(2,6,8)	0.00	71.9
G	(4,6,10)	0.00	77.6	T	(28,31,33)	0.00	79.0
H	(5,6,10)	0.00	78.3	U	(31,32,34)	0.00	17.5
I	(2,8,13)	0.00	73.5	V	(10,11,15)	0.00	77.9
J	(3,8,13)	0.00	28.4	W	(10,11,15)	0.00	80.5
K	(1,5,35)	0.00	58.4	X	(7,9,12)	0.01	78.6
L	(1,5,8)	0.00	11.1	Y	(19,21,23)	0.00	57.1
M	(31,33,35)	0.00	68.2	Z	(26,31,33)	0.00	21.2

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.

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 be also reduce. Thus, the GA has also improved in its implementation because it is less random and consuming less time for searching the optimal solution.

These results clearly indicate that 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 4.8% in the case of one-bit error, 0.06% in the case of two-bit error and 0.001% in the case of three-bit error induced in the test pattern randomly. On the other hand, the GA recalls the pattern successfully even when high noise is present in the input pattern i.e. up to four- or five-bits. Although due to limited resources, a very few experimental runs were conducted for the input test patterns with four- or five-bits induced noise and therefore these results are not included in this section.

It has been claimed by that the capacity of deterministic Hopfield model with hebbian rule is about 0.15N for the noisy prototype input patterns, where N is the number of modes in the network. If such a network is overloaded with a number of patterns exceeding its capacity, its performance rapidly deteriorates towards zero. Here, we are storing the 26 alphabets in a network of 35 nodes and the performance of the GA suggests that on inducing five-bit error in presented prototype input pattern the network is able to recall the stored patterns. It implies that the network capacity has increased up to 0.75N. Thus, the numbers of attractions exist here and successfully explored during the recalling process. It is quite 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 the capacity has been increased with the GA by exploring the optimal weight matrices for the encoded patterns.

Figure 5-8 are presenting comparison chart of performance of two algorithms, i.e. the Hebbian rule and the genetic algorithm graphically based on results provided in Tables 4-7.

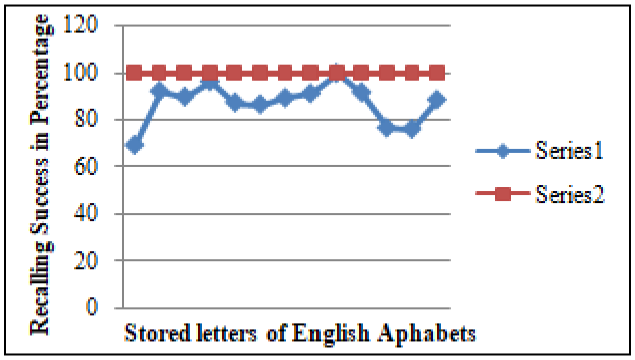

Figure 5: Comparison chart for table 4

Series 1: Percentage of Recalled Patterns using Hebbian Rule
Series 2: Percentage of Recalled Patterns using Genetic Algorithm

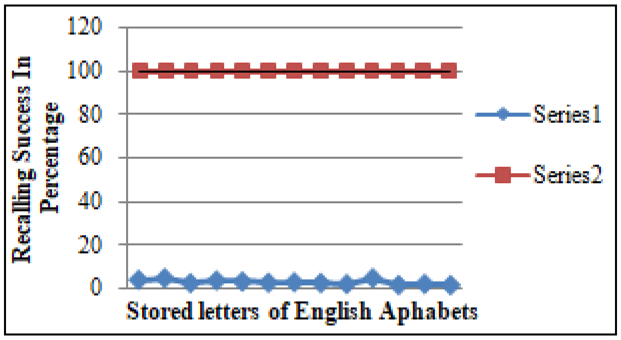

Figure 6: Comparison chart for Table 5

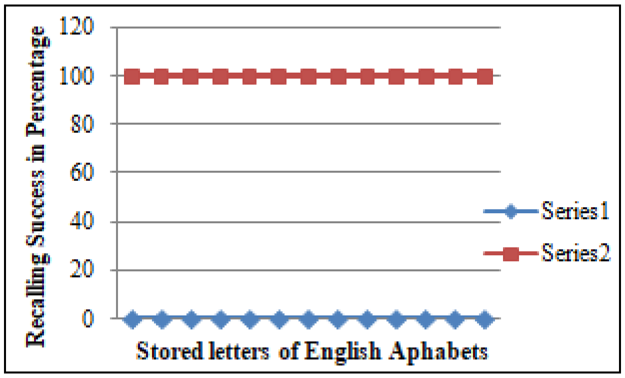

Figure 7: Comparison chart for Table 6

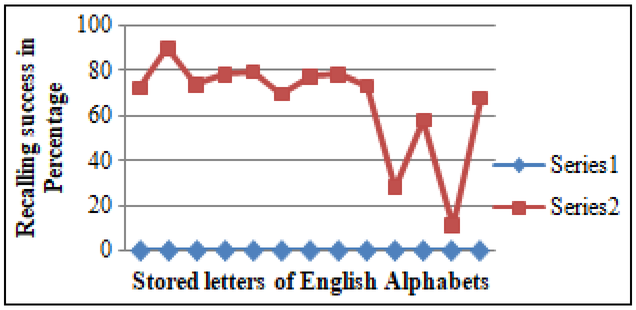

Figure 8: Comparison chart for Table 7

6. Conclusions

The simulation results, i.e. tables 4-7 indicate that the genetic algorithm has more success rate than the Hebbian rule for recalling the letters of English alphabet, which are containing zero-, one-, two-, and three- bit errors from stored patterns in the Hopfield neural network. Sometimes it has also been observed that the performance of the GA was less than what was expected to be. One of the reasons for this deviation may be the position(s) of bits reverted to induce noise in the recalling pattern. Another interesting observation is that there is confusion while recalling some of the letters through both, the Hebbian rule and the GA. For example, on inducing two-bit error in letter D at positions (1, 5), (1, 2) and (31, 35), the recalling was confused with letter O. Similarly, on induction of three bit error in letter U at position (30, 32, 34), the recalling was also made for letter V. The fact that the recalling algorithms are based on the minimum hamming distance of noisy pattern with stored pattern may be the reason of confusion in recalling those pairs of letters that are having similar minimum hamming distances.

It has been found that, the GA can give more than one convergent weight matrices for any prototype input pattern in comparison to the conventional Hebbian rule, if the prototype input pattern is correctly recognized. For conventional Hebbian recalling algorithm, if the prototype input patterns are correctly recognized by the network, then only one convergence matrix will be obtained. This shows the higher accuracy rate in the pattern recognition with GA.

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.

References

[1] S. Kumar, (2004) Neural Networks: A classroom approach, TMH.
[2] J. J. Hopfield, (1992) *Neural Networks and Physical Systems with Emergent Collective Computational Abilities*, in: Proceedings of the National Academy Sciences, USA, Vol79, pp. 2554-2558.
[3] M. Manga, and M. P. Singh, (2006) *Handwritten English Vowels using Hybrid Evolutionary Feed-forward Neural Network*, Malaysian Journal of Computer Science, Vol. 19, No. 2, pp. 169-187.
[4] A. Imada & K. Araki, (1997) *Applications of an Evolutionary Strategy to the Hopfield Model of Associative Memory*, in: Proceedings of the IEEE International Conference on Evolutionary Computation, pp. 679-680.