Hopfield Neural Network as Associated Memory with Monte Carlo-(MC-)Adaptation Rule and Genetic Algorithm for pattern storage

Manisha Uprety\textsuperscript{1}, Somesh Kumar\textsuperscript{2}

\textsuperscript{1}M.Tech (CSE) Student
\textsuperscript{2}Noida Institute of Engineering & Technology, Greater Noida
uprety.manisha@gmail.com, someshkumarrajput@rediff.com

ABSTRACT

This paper describes the performance analysis of Hopfield neural networks by using genetic algorithm and Monte Carlo-(MC-) adaptation learning rule. A set of five objects has been considered as the pattern set. In the Hopfield type of neural networks of associative memory, the weighted code of input patterns provides an auto-associative function in the network. The storing of the objects has been performed using Hebbian rule and recalling of these stored patterns on presentation of prototype input patterns has been made using both - conventional hebbian rule and genetic algorithm. In most cases, the recalling of patterns using genetic algorithm with MC-adaptation rule seems to give better results than the conventional hebbian rule, MC-adaptation rule and simple genetic algorithm recalling techniques.

Keywords: Hopfield neural network, Genetic algorithm, Monte Carlo adaptation rule, Pattern association.

INTRODUCTION

Since the Hopfield neural network with associative memory \cite{1-2} was introduced, various modifications \cite{3-9} are developed for the purpose of storing and retrieving memory patterns as fixed-point attractors.

The Artificial neural networks (ANN) attempt the modeling of the human brain in a serial machine for various pattern recognition tasks and have become nowadays well-established computational structures which are usually seen as a method for implementing complex nonlinear functions using simple elementary units connected together with appropriate weighted connections \cite{11-14}.

The Hebbian learning rule \cite{15} is a local learning rule. Monte Carlo-(MC-) adaptation rule \cite{16-18} used for training single-layer recurrent typeneural networks.

The basic idea of MC-adaptation rule is to make adaptation to arandomly chosen synaptic weight and accept the adaptation if it improves thenetwork performance globally. Recently, two heuristic search techniques havegenerated interest in the artificial intelligence community: Neural Networks (NNs) and Genetic Algorithms (GAs) \cite{19}. Both NNs and GAs are based on models fromnature.

The objective is to obtain the optimize weight matrices for efficient recalling of an input prototype pattern/noisy input prototype pattern. To accomplish this, first we obtain the coupling matrix randomly with four digit decimal numbers. After this, we apply the MC-adaptation rule on this coupling matrix. At the end, the genetic algorithm is applied on the weight matrix obtained by the MC-adaptation rule. The genetic algorithm, explore the population generation technique, the crossover operator and the fitness evaluation function in order to generate the optimal weight matrix. In most cases, the recalling of patterns using genetic algorithm with MC-adaptation rule seems to give better results than conventional Hebbian rule. The result suggest that the genetic algorithm with MC-adaptation rule is the better searching technique for recalling the noisy prototype input patterns.

Hopfield Neural Network

AHNN is a simple artificial network which is able to store certain memories or patterns in a manner rather similar to the brain - the full pattern can be recovered if the network is presented with only partial information \cite{20}\cite{21}.

The smart thing about the Hopfield network is that there exists a rather simple way of setting up the connections between nodes in such a way that any desired set of patterns can be made "stable firing patterns". Thus any set of memories can be burned into the network at the beginning. Then if we kick the network off with any old
set of node activity we are guaranteed that a "memory" will be recalled. Not too surprisingly, the memory that is recalled is the one which is "closest" to the starting pattern. In other words, we can give the network a corrupted image or memory and the network will "all by itself" try to reconstruct the perfect image.

In HNN, the output of each unit is fed to all the other units with weights $w_{ij}$ except $w_{ii}$, let the output function of each of the units be bipolar so that

$$s_i = f(x) = \text{sgn}(x_i)$$

$$x_i = \sum w_{ij} s_j - \theta_i$$

where $\theta_i$ is the threshold for the unit $i$.

The state of each unit is either +1 or -1 at any given instant of time. The state at time $(t+1)$ is the same as the state at time $t$ for all the units. That is,

$$s_i(t+1) = s_i(t)$$

for all $i$.

Associated with each state of the network, Hopfield proposed an energy function whose values always either reduces or remains the same as the state of the network changes. Assuming the threshold value of the unit $i$ to be $\theta_i$, the energy function is given by

$$V(s) = V = -\frac{1}{2} \sum_i \sum_j w_{ij} s_i s_j + \sum_i \theta_i s_i$$

The energy $V(s)$ as a function of the state $s$ of the network describes the energy landscape in the state space. The energy landscape is determined only by the network architecture, i.e., the number of units, their output functions, threshold values, connections between units and the strengths of the connections.

The change in energy due to update of the $k$th unit is given by

$$\Delta V = V_{\text{new}} - V_{\text{old}}$$

And therefore,

$$\Delta V = (s_{k}^{\text{old}} - s_{k}^{\text{new}}) \left[ \sum_i w_{ki} s_i^{\text{old}} - \theta_k \right]$$

Therefore the energy decreases or remains the same when a unit, selected at random, is updated provided the weights are symmetric, and the self-feedback is zero.

**Hebbian Rule**

Synaptic dynamics, as discussed earlier, is described in terms of expressions for the first derivative of the weights. They are called learning equations. Hebbian is the simplest learning rule. Each neuron can be in one of the two states, i.e. ±1, and $p$ bipolar patterns $X^\mu = (x_1^\mu, x_2^\mu, \ldots, x_N^\mu)$, $\mu = (1,2,\ldots,p)$, are to be memorized in associative memory. In this type of neural network, the coupling matrix is usually determined by the Hebbian rule as follows:

$$W_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} x_i^\mu x_j^\mu \text{ and } W_{ii} = 0$$

where $\{x_i^\mu, i = 1,2,\ldots,N; \mu = 1,2,\ldots,p\}$ is a set of $p$ patterns to be stored and $N$ is the number of the neurons.

**MC-adaptation Rule**

The MC-adaptation rule is different from the conventional Hebbian rule as well as other learning rules such as perceptron. When apply this rules memory patterns are input into the learning process one by one. i.e. each time the adaptation of the coupling matrix is carried out by pursuing the optimal solution for a specific memory pattern. Hebbian is local learning rule while MC-adaptation rule is a global design rule.

**Step 1**

Coupling matrix can be obtained randomly with four digit decimal numbers between -1 and 1. Then we apply the MC adaptation rule to guarantee all the memory patterns satisfying the fixed point condition by driving the local fields to the region of the following

$$h_i^\mu \leq c \leq 0 \quad \text{where } c = 0.10$$

**Step 2:**

Specify a row in the coupling matrix $(J)$, say the $i^{th}$ row, and calculate

$$h_i^\mu = \sum J_{ij} \xi_j^\mu \text{ for } j = 1,2,3,\ldots,N$$

$$\mu = \mu^1, \ldots, \mu^m \text{ (patterns)}$$

**Step 3:**
Now find the $\min(h_i^\mu)$ and let $\{\mu^1, \ldots, \mu^m\}$ record the indices of patterns satisfying $h_i^\mu = h_i^{\min}$.

**Step 4:**
Initialize two variables as

$$m_j^+ = m_j^+ = 0$$

And proceed as

$$m_j^+ = m_j^+ + 1$$

$$m_j^- = m_j^- + 1 \quad \text{Otherwise}$$

For $\mu = \mu^1, \ldots, \mu^n$ and $j = 1, 2, 3, \ldots, N$

**Step 5:**
Now record column indices $\{j_1^+, j_2^+ \ldots, j_k^+\}$ with satisfying condition $m_j^+ = \max( m_j^+)$. and $\{j_1^-, j_2^- \ldots, j_k^-\}$ with satisfying condition $m_j^- = \max( m_j^-)$ respectively.

**Step 6:**
Adaptation as follows

If $\max( m_j^+) > \max( m_j^-)$ randomly pick $j$ from the list $\{j_1^+, j_2^+ \ldots, j_k^+\}$ and make adaptation $J_{ij} = J_{ij} + 1/N$.

If $\max( m_j^+) < \max( m_j^-)$ randomly pick $j$ from the list $\{j_1^-, j_2^- \ldots, j_k^-\}$ and make adaptation $J_{ij} = J_{ij} - 1/N$.

Otherwise randomly pick an index from the list $j_1^+, j_2^+ \ldots, j_k^+$ or $j_1^-, j_2^- \ldots, j_k^-$ with equal probability, and if $j \in \{j_1^+, j_2^+ \ldots, j_k^+\}$ make an adaptation

$$J_{ij} = J_{ij} + 1/N \quad \text{otherwise}$$

$$J_{ij} = J_{ij} - 1/N .$$

Repeat step 2 to step 6 until $h_i^{\min} > c$

Apply the above procedure until $h_i^{\min} > c$ for all rows.

When $h_i^{\min} > c$ for all rows $i$, we obtain a desirable weight matrix $J$ with the patterns being memory as fixed point.

**The Genetic Algorithm**

In this simulation, recalling is done by Genetic Algorithm. When GA [22-25] starts, a population of weight matrices is produced by crossover from the parent weight matrices which are generated by MC adaptation rule in the storing stage. In each generation, this population is modified through uniform random mutations and their fitness values are evaluated.

The cycle of generating the new population with better individuals and restarting the search is repeated until an optimum solution was found. The fitness function is evaluating the best matrices of the weights population on the basics of the hundred percent successful recalling with zero bit error of the stored patterns on the presentation of the same as the input pattern. It indicates that the stable states of the network will be used for the evaluation of the weight’s population.

**Crossover** generates new population of size $N \times N + \text{Initial Population}$

Where $N$ is number of neuron

**Fitness function** evaluates and collect all those weight matrices which can successfully recall the respective stored patterns by providing the same as input pattern.
(with no error) at a time will be considered as fitted weight matrix.
After fitness evaluation mutation operator executes to increase the population and generates population of size-

\[ N \times N + \text{Fitted Population} \]

Now this generated population will be used for recalling purpose.

**EXPERIMENTS**

The patterns used for the simulations are shown in Figure 1. Each pattern consisted of a 5 X 4 pixel matrix representing an alphabet of the set. White and black pixels are respectively assigned corresponding values of -1 and +1.

![Figure 1: Set of patterns used for training](image)

Using these bipolar values, the set of above alphabets is represented in the form a series. For example, object a is written as:

\[-1 \ 1 \ -1 \ 1 \ -1 \ -1 \ 1 \ 1 \ 1 \ -1 \ -1 \ -1 \ -1 \]

The results presented in this section demonstrate that, within the simulation framework presented above, large significant difference exists between the performance of genetic algorithm conventional Hebbian rule for recalling alphabets those have been stored in Hopfield neural network using MC adaptation rule and Hebbian learning rule respectively.

In total 1000 times the recalling was made through both the algorithms separately for each pattern. In these cases, noise was created by reverting 0-bit, 1-bit, 2-bits and 3-bits in the presented prototype input patterns in the already stored patterns. These positions of the bit(s) to be reverted to create noise are taken randomly.

![Figure 2: The results of recalling of taken set of patterns with no error in the input patterns](image)

**Table 1**: The results of recalling of taken set of patterns when there is no error in the presented input prototype patterns.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Hebbian</th>
<th>GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pat1</td>
<td>90.8</td>
<td>100</td>
</tr>
<tr>
<td>Pat2</td>
<td>90</td>
<td>100</td>
</tr>
<tr>
<td>Pat3</td>
<td>99.6</td>
<td>100</td>
</tr>
<tr>
<td>Pat4</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Pat5</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

**Table 2**: The results of recalling of taken set of alphabets when there is 1-bit error in the presented input prototype patterns.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Hebbian</th>
<th>GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pat1</td>
<td>4.9</td>
<td>100</td>
</tr>
<tr>
<td>Pat2</td>
<td>3.8</td>
<td>100</td>
</tr>
<tr>
<td>Pat3</td>
<td>5.1</td>
<td>100</td>
</tr>
<tr>
<td>Pat4</td>
<td>4.1</td>
<td>100</td>
</tr>
<tr>
<td>Pat5</td>
<td>8.1</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reverted bit position</th>
<th>3</th>
<th>6</th>
<th>5</th>
<th>9</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reverted bit position</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>14</td>
</tr>
</tbody>
</table>
It is also possible to obtain more than one weight matrices from the generated population of input prototype patterns. One of the reasons for this deviation may be the position(s) of bits reverted to induce noise in the presented input prototype patterns. One of the reasons for this deviation may be the position(s) of bits reverted to induce noise in the presented input prototype patterns. One of the reasons for this deviation may be the position(s) of bits reverted to induce noise in the presented input prototype patterns. One of the reasons for this deviation may be the position(s) of bits reverted to induce noise in the presented input prototype patterns. One of the reasons for this deviation may be the position(s) of bits reverted to induce noise in the presented input prototype patterns.

The results of recalling of taken set of alphabets when there is 3-bits error in the presented input prototype patterns.

![Figure 3](image3.png)

**Figure 3**: The results of recalling of taken set of patterns with 1-bit error in the input patterns

<table>
<thead>
<tr>
<th>Alphabet</th>
<th>Pattern 1</th>
<th>Pattern 2</th>
<th>Pattern 3</th>
<th>Pattern 4</th>
<th>Pattern 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>Reverted bit position</td>
<td>1, 8, 12</td>
<td>2, 7, 15</td>
<td>5, 7, 11</td>
<td>3, 7, 9</td>
<td>5, 10, 19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Recall Success (in %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>15, 28, 32, 10, 47</td>
</tr>
<tr>
<td>Hebbian</td>
<td>.2, .0, .1, .0, .0</td>
</tr>
</tbody>
</table>

**Table 3**: The results of recalling of taken set of alphabets when there is 2-bits error in the presented input prototype patterns

![Figure 4](image4.png)

**Figure 4**: The results of recalling of taken set of patterns with 2-bits error in the input patterns

![Figure 5](image5.png)

**Figure 5**: The results of recalling of taken set of patterns with 3-bits error in the input patterns

**CONCLUSION AND FUTURE SCOPE**

By using MC adaptation rule to slightly modify the Hopfield Neural network designed by the Hebbian rule, one can significantly improve the performance of the network. The improved Neural Network has higher storage capacity than the original ones. Figure 2, 3, 4 & 5 are presenting the comparison chart of performance of two algorithms (i.e. Hebbian rule algorithm and GA algorithm) graphically based on results provided in Tables 1, 2, 3 & 4.

The simulation results are indicating that genetic algorithm with MC adaptation learning rule has more success rate then the Hebbian rule for storing and recalling the taken set of alphabets, which are containing 0, 1, 2, and 3 bit errors from stored patterns in Hopfield neural network. Sometimes it has also been observed that the performance of GA was less than what was expected to be. One of the reason for this deviation may be the position(s) of bits reverted to induce noise in the recalling pattern. Second reason may be high similarity between two stored patterns which can be reduced by taking more pixels and hence more neurons in the Hopfield memory. It is also possible to obtain more than one weight matrices from the generated population of...
weight matrices as the optimal weight matrices for recalling the exact pattern on presentation of any prototype input pattern of already stored pattern.

The direct application of GA with MC adaptation rule to the pattern association has been explored in this paper. The aim is to introduce as 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 performed especially on the tests for noisy input patterns. We can extend this concept for pattern recognition for alphabets of different languages, shapes, numerals. Some real dataset of handwritten characters may also be tested using the presented approach and the comparison with the previous approaches may be analyzed.

References


