

Hopfield Networks: A Simple OCR Application

Introduction

In this paper I discuss the results of training and testing a Hopfield network (see next section: What is a Hopfield Network) and its optimized counterpart to recognize the digits 0-9. I also compare the performance of the optimized and non-optimized brands. This paper covers my own experience of following the process laid out in Sigillito[1]. My results turn out to be very similar to those found by Sigillito.

What is a Hopfield Network?

A Hopfield network is a type of neural network that allows stored patterns to be retrieved by context. This means that you can feed the network partial versions of patterns it has learned and it will return the corresponding correct pattern (usually).

A Hopfield network is made up of nodes that are connected via weights in a fully connected and symmetric fashion with no self connections. If you need to see that in a formulaic fashion, here it is:

1. $w_{ii}=0$ for all i
2. $w_{ij}=w_{ji}$ for all i,j

The number of nodes to use will equal the number of bits in your input patterns (exemplars). For example, for the bit patterns used in this experiment we have each exemplar being 120 bits in length, so we will have 120 nodes.

The **learning process** of Hopfield network is to update its weights for each exemplar that is presented to it. Sigillito[1] explains this most succinctly and clearly in his paper:

The network connectivity is given by the weight matrix $\mathbf{W} = (w_{ij})$ of connection strengths. The weight matrix \mathbf{W} is determined by an outer-product learning rule: Let $\mathbf{e}^1, \mathbf{e}^2, \dots, \mathbf{e}^p$ be p exemplars (or patterns) that the network is to learn. Each exemplar has n components, so the j th exemplar is $\mathbf{e}^j = (e_j^1, e_j^2, \dots, e_j^n)$. Exemplar components can only take on the values ± 1 . Then the weight matrix is given by the outer-product sum

$$\mathbf{W} = \sum_{s=1}^p (\mathbf{e}^s)^T \mathbf{e}^s - \rho \mathbf{I}_n,$$

where T denotes vector transpose and \mathbf{I}_n is the $n \times n$ identity matrix. The effect of the term $\rho \mathbf{I}_n$ is to zero all self-connections.

The **recall process** of a Hopfield net can be synchronous or asynchronous. I must again quote from Sigillito[1] because I am not able to improve on his explanations¹.

¹ And it would take me hours to type in all of those subscripts and mathematical symbols. But really, he does offer such clear and concise descriptions that can't be beaten.

Synchronous dynamics:

Let $\tilde{\mathbf{e}} = (\tilde{e}_1, \tilde{e}_2, \dots, \tilde{e}_n)$ denote a noisy exemplar.

Then the initial output (at iteration 0) of each neuron is

$$x_i(0) = \tilde{e}_i, i = 1, 2, \dots, n.$$

Then iterate until convergence:

$$a_i(t+1) = \sum_j w_{ij} x_j(t) \quad i = 1, 2, \dots, n,$$

$$x_i(t+1) = F_h[a_i(t+1)]$$

where t is the iteration number.

Asynchronous dynamics:

The $x_i(0)$ are defined as for synchronous dynamics. Then iterate until convergence. Let m be an integer chosen at random from the set $\{1, 2, \dots, n\}$. Then

$$a_m(t+1) = \sum_j w_{mj} x_j(t),$$

$$x_m(t+1) = F_h[a_m(t)],$$

and

$$a_j(t+1) = a_j(t) \quad j = 1, 2, \dots, n; j \neq m.$$

$$x_j(t+1) = F_h[a_j(t+1)]$$

For both recall processes:

$$F_h(a) = \begin{cases} +1 & a > 0 \\ -1 & a \leq 0 \end{cases}.$$

What is an Optimized Hopfield Network?

An optimized Hopfield network works just like the previously described Hopfield network except that the learning rule of:

$$\mathbf{W} = \sum_{s=1}^p (\mathbf{e}^s)^T \mathbf{e}^s - \rho \mathbf{I}_n,$$

is replaced by a learning rule that minimizes the squared error E :

$$E = \frac{1}{2} \sum_{s=1}^p \sum_{l=1}^n (e_l^s - \sum_{k=1}^n w_{lk} e_k^s)^2.$$

For more discussion on how these weights are updated see appendix A.

Experimental Setup, Results, and Discussion

Part I (Implementing a Hopfield Network)

Using Exemplars 0-4

In this part of the experiment I set up a Hopfield network with 120 nodes and gave it 5 exemplars to learn (figure 1). Each exemplar is made up of 12 rows of 10 bits. However the network represents this internally as 120 bit vectors. Additionally the network represents the bits as either -1 or +1.

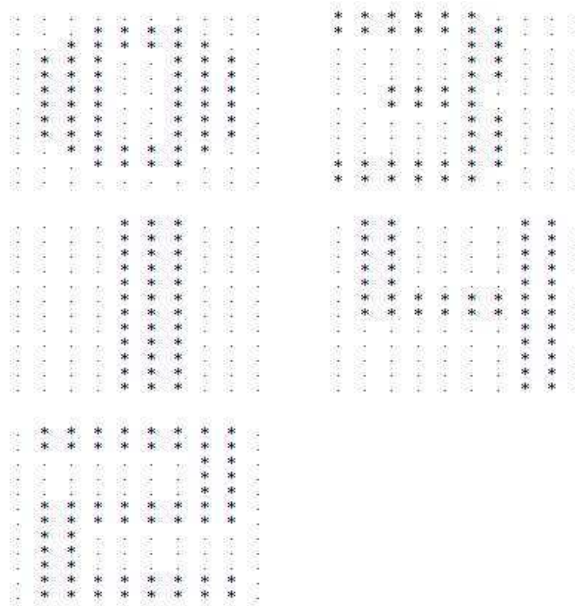


Figure 1: The exemplars used in this experiment. Note that *'s are interpreted as +1 and -'s as -1.

Once the network has learned these patterns I did some testing. Specifically I tested recall performance with 5 different noise levels (0, .1, .2, .3, and .4). These noise level values refer to the probability that any given bit gets flipped in the pattern presented to the network. (See figure 2.) I limited the number of iterations to 25 during recall.

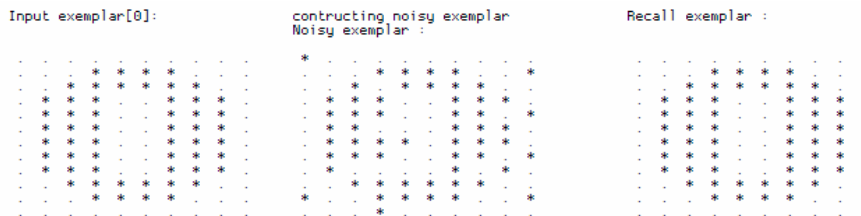


Figure 2: An instance of the testing network recall. The learned exemplar 0 (left) gets its bits flipped with a probability of .1 (middle), and the learned exemplar 0 is correctly recalled (right)

The network performed quite well. As you can see from figures 3 and 4, we get perfect recall for noise levels 0 and .1. At .2 noise level, performance is almost perfect, and after that it gets worse. **Error method 1** means that the recalled pattern is considered in correct if it differs from the correct pattern by more than 5 bits. **Error method 2** considers the recalled pattern correct if it is closed in Hamming distance to the correct pattern compared to all of the other learned patterns. As you can guess, error method 2 tends to be bit a bit more forgiving.

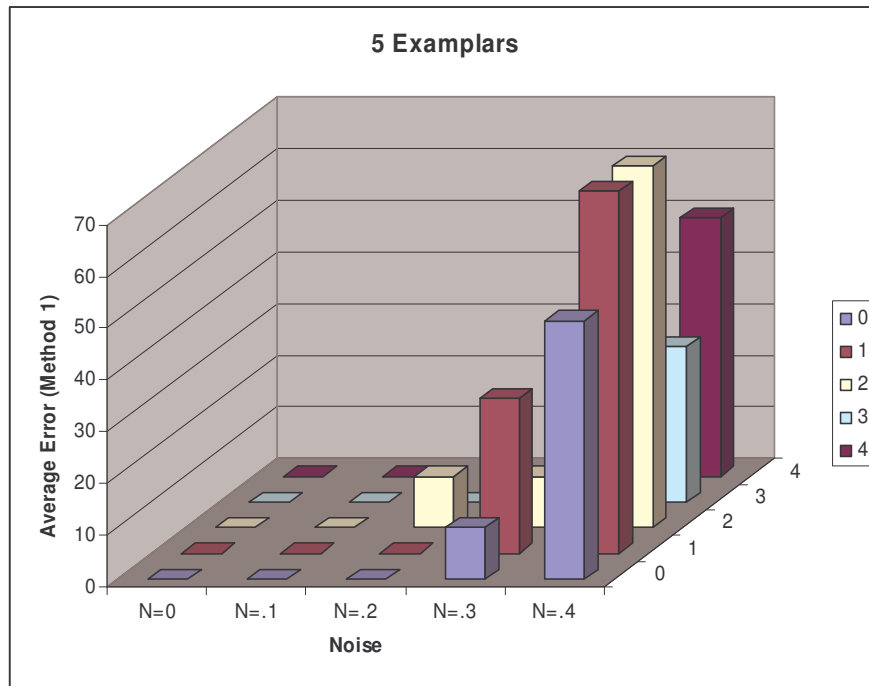


Figure 3: Recall performance by exemplar (0-4) at various noise levels using error method 1

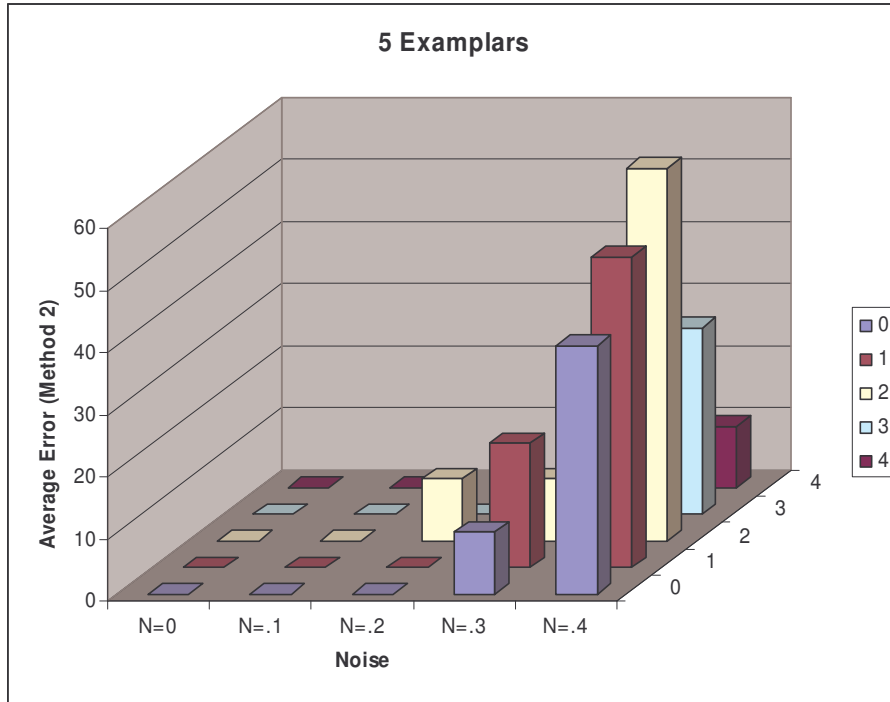


Figure 4: Recall performance by exemplar (0-4) at various noise levels using error method 2

Using all 10 Exemplars (0-9)

This part of the experiment was set up with all of the same parameters as the last part except that I trained the network with all 10 exemplars (0-9). As you will see this had the effect of essentially overloading the network beyond its capacity and our performance suffers. As mentioned in Sigillito[1], in the best case scenarios when all of the exemplars are stochastically independent **the capacity of a Hopfield net is about $.14 * (\text{the number of neurons})$ exemplars** which in our case is $.14 * 120 = 16.8$. However in our case the exemplars are not stochastically independent because some of the digits look close to one another. In fact in this experiment, patterns 2, 4, 7, 8 and 9 are never correctly recalled even at a noise level of 0 (See figure 5.). Thus I reached the same conclusion as Sigillito[1] here that **the capacity of this network is effectively 5 exemplars.**

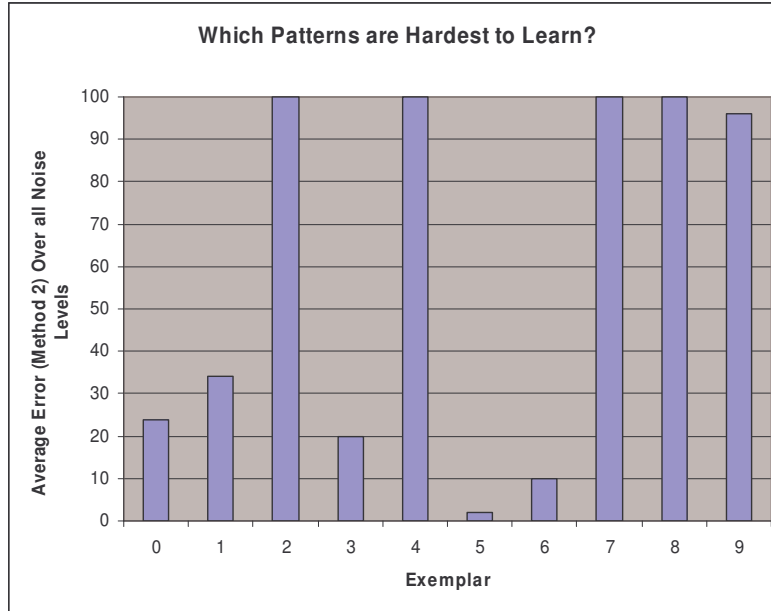


Figure 5: Average method 2 recall error across all 5 noise levels. Interestingly while at a noise level of 0, pattern 9 is never correctly recalled, at higher noise levels we get it correct a few times. I am fairly sure it is just blind luck so to speak and you shouldn't attach any significance to it.

To conclude part 1 of this paper, using 5 exemplars works quite well, but once you get over 5 exemplars, recall suffers drastically. Figure 6 compares the performance of these two experiments.

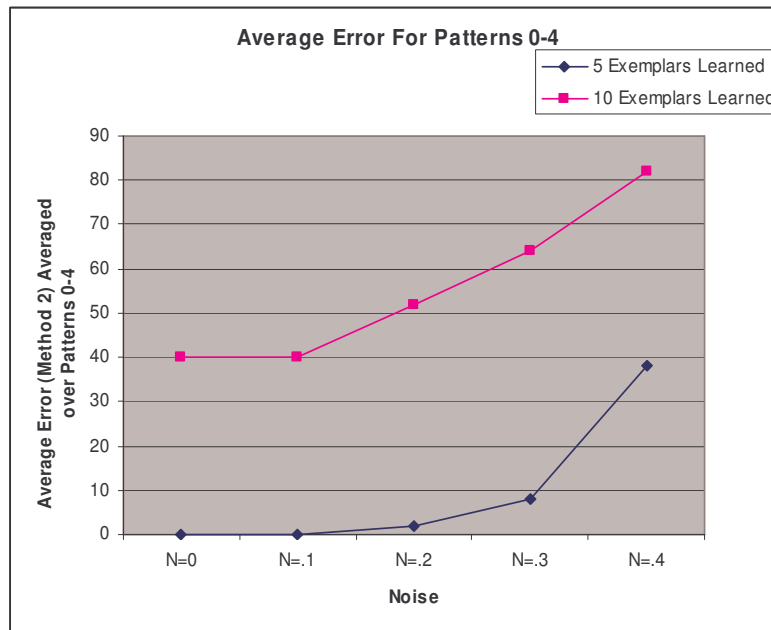


Figure 6: Comparing the performance of two Hopfield nets, one trained at what we have deemed full capacity (5 exemplars) and one trained to double that. Notice how the recall performance suffers. Note that this is only comparing performance on digits 0-4.

Part II (Implementing and Testing an Optimized Hopfield Net)

Next I implemented an Optimized Hopfield net. See the section: What is a Hopfield Net for the **specific learning and recall processes of an optimized Hopfield net**. I trained it with all 10 exemplars and tested² its recall using the same five noise levels of 0, .1, .2, .3, and .4 as in part 1. I also had to initialize all of the weights to 0, and I used an eta value of .1 (eta is the learning rate. See appendix A for more discussion of the learning rule.)

As you can in figures 7 and 8, recall is perfect for noise level $\leq .2$ and this it gets slightly worse. However even at noise level .4 we are still below 50% error on average.

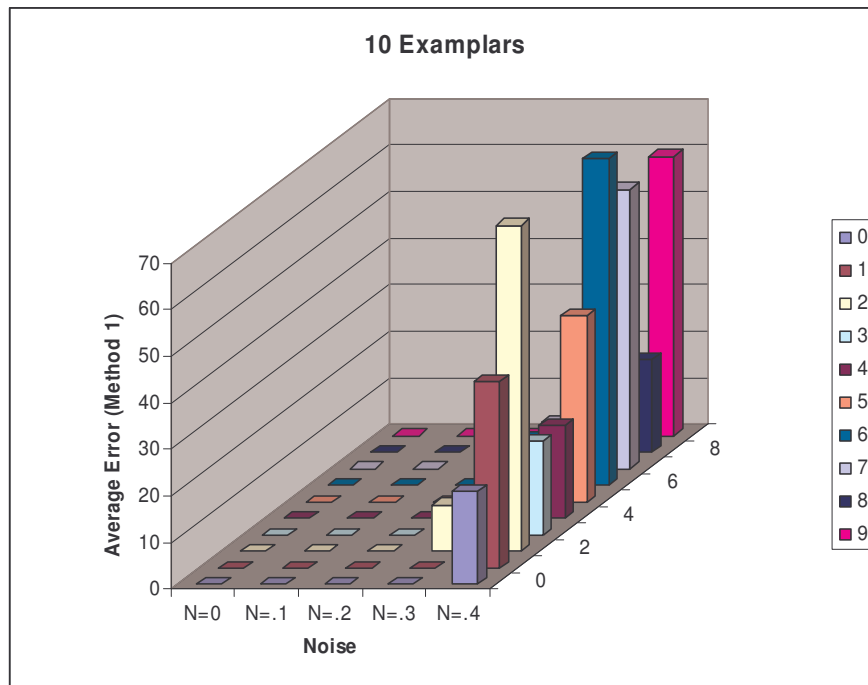


Figure 7: Recall performance by exemplar (0-9) at various noise levels using error method 2. Notice that recall is perfect for low noise levels.

² Note that I only discuss and graph error method 2 here. This is fair because I believe it is a slightly more representative error method. Additionally, the software I worked with only provided error method 2 for using an Optimized Hopfield network.

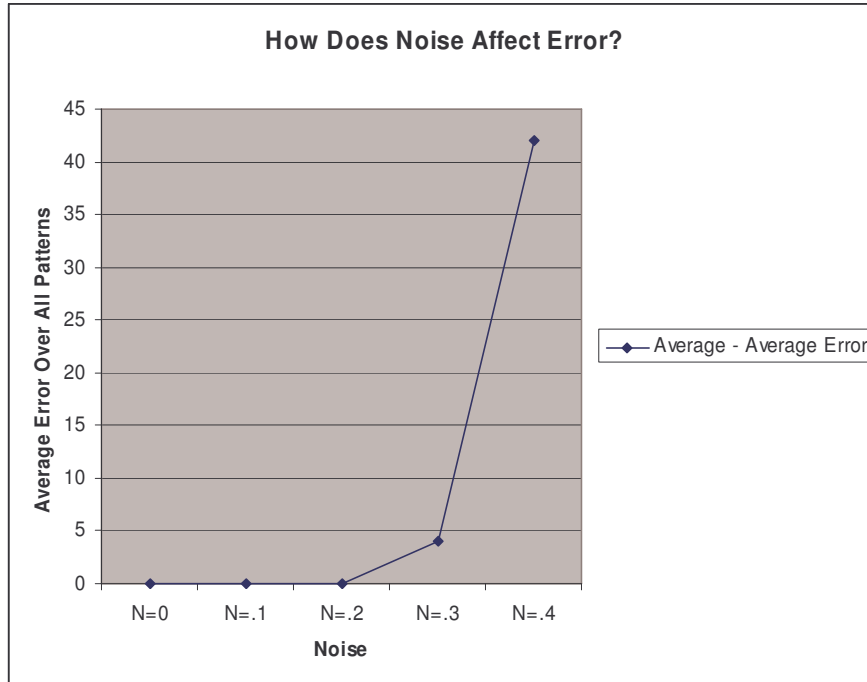


Figure 8: Average recall error (across all 10 patterns) versus r (the noise level)

Optimized Hopfield Network versus Non-optimized Hopfield Network

Finally comparing the average performance of this optimized Hopfield net with the performance of the un-optimized Hopfield net of part 1 we get figure 9. As you can see, **the performance is greatly improved**. One can safely assume that we haven't reached the capacity of the optimized Hopfield net.

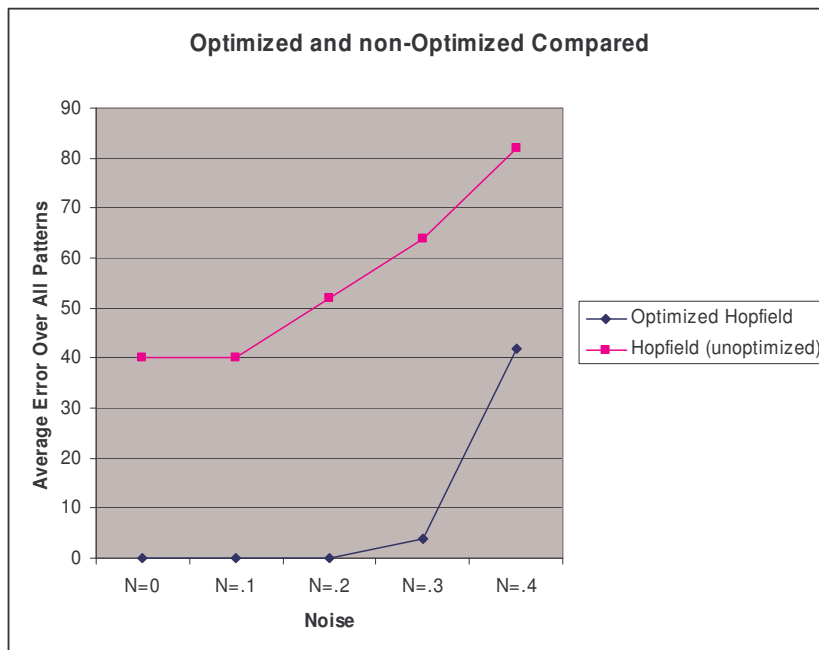


Figure 9: Comparing the average method 2 error over all patterns (0-9) for optimized and un-optimized Hopfield nets.

Conclusion

The primary lesson you should take away from this paper are the even though the experts say that a Hopfield net can reliably store up to $.14 * (\text{the number of neurons})$ exemplars, in real world practice this number may well not even reach half of that. Even in a set of exemplars as innocuous as the digits 0-9 we see that many patterns are not all that independent. In this case we only achieved good performance by keeping the number of stored patterns at 5.

The other lesson is that the performance of an optimized Hopfield net is significantly better. I believe that the performance is better because the learning rule here lets us find not just weight values that represent the learned patterns, but actually find the optimal weight values that will minimize confusion (error) when recalling those patterns.

Personally as far as **future work**, I am very interested in seeing how a multi-layer perceptron performs against an optimized Hopfield net in recognizing these digits with various noise levels. I am also interested in researching methods to increase the capacity of Hopfield nets and/or work around the existing capacity constraints.

References

1. V. Sigillito, "Associative Memories and Feedforward Networks: A Synopsis of Neural-Network Research at the Milton S. Eisenhower Research Center", Johns Hopkins APL Technical Digest, vol. 10, no. 3, pp. 254—261, 1989.
2. K. Knight, "Connectionist ideas and algorithms," Communications of the ACM, vol. 33, no. 11, pp. 59--74, 1990.
3. Wikipedia contributors (2006). Hopfield net. Wikipedia, The Free Encyclopedia. Retrieved 17:50, May 1, 2006 from http://en.wikipedia.org/w/index.php?title=Hopfield_net&oldid=50960238.

Appendix A

This is another excerpt from Sigillito[1] which shows how the weights get updated in an optimized Hopfield net. You can find this and more discussion on page 257, figure 5 of Sigillito's paper.

- The learning rule for the optimal Hopfield network is based on minimizing the squared error E :

$$E = \frac{1}{2} \sum_j \sum_i (e_j^i - \sum_k w_{jk} e_k^i)^2 .$$

- Differentiating E with respect to w_{ij} results in the following iterative learning rule:

Initially, set w_{ij} to zero.

For $s = 1, 2, \dots, p$, iterate until convergence:

$$\Delta^s w_{ij} = \eta (e_j^i - \sum_k w_{jk} e_k^i) e_j^i$$

$i = 1, 2, \dots, n; j = 1, 2, \dots, n.$

$$w_{ij} = w_{ij} + \Delta^s w_{ij}$$

The learning rate parameter, η , is needed for numerical stability. A typical value of η is $1/n$.

Set the self-connection weights to zero:

$$w_{ii} = 0, i = 1, 2, \dots, n.$$

- The recall process can be either synchronous or asynchronous. In either case the equations that describe the process are identical to those given in Figure 1 for traditional Hopfield networks.