UNIVERSITY OF TORONTO, DEPARTMENT OF ECE

ECE1512 - Project Report

Hopfield Network for Handwritten Digits Recognition

Yang Wang
Student Number: 1001319227

December 2, 2015

1 INTRODUCTION

Handwritten digits recognition is an old and important problem of machine learning. The objective is to recognize images of isolated handwritten digits (0-9). More specifically, the problem is equivalent to find a model, which takes image of handwritten digit as input, and output the predicted class label of the image. Furthermore, the ideas and methods to solve this problem would be very useful in various fields of pattern recognition problems where large volumes of real-world data is used[1].

Artificial neural networks are mathematical constructs that model certain aspects of biological neurons. Neural networks consist of one or more neurons which can exhibit complex behavior due to the nature of the connections between neurons. Neural networks can serve to store memories of data. These memories mimic biological systems in several ways[2].

1. They are associative. With associative memories, one can recall a memory through its association with other memories.

2. They are robust in handling error. In a neural network, memories can correct for a certain amount of error and still recall the correct memory.

3. Memories are distributed. In a neural network, each neuron plays a part in memory storage. This is opposed to traditional computing, where a memory is associated with a specific location on the storage device.

Hopfield network is a type of neural network named for John Joseph Hopfield (an American Scientist). It is a fully connected network which contains multiple neurons, each of which is connected to the others via weights. A key feature of Hopfield network is that it converges to a local minimum. This feature creates applications in two main areas: associative memories and optimization. With associative learning, the network can be trained to remember states which can be recalled if it is given only part of the state. Within optimization problems, as the network converges to its local minimum, the neural network's objective function can be optimized for a desired solution.

2 Background

2.1 Hopfield Network Elements

As shown in Figure 2.1, Hopfield network consists of a set of interconnected neurons which update their activation values asynchronously. The activation values are binary, usually {-1,1}. The update of a unit depends on the other units of the network and on itself. A unit i will be influence by another unit j with a certain weight w_{ij}, and have a threshold value[2].

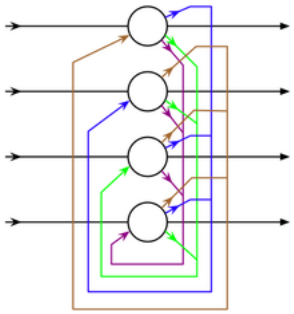


Figure 2.1: An example of Hopfield Network

So there is a constraint due to the other neurons and the specific threshold of the unit.

The new activation value (state) of a neuron is compute, in discrete time, by the function 2.1:

$$x_i(t+1) = sign(\sum_{j=1}^{n} x_j(t) w_{ij} - \theta_i) \tag{2.1}$$

or function 2.2

$$X = sign(XW - T) \tag{2.2}$$

Where X, W, T and the $sign$ function are:

- X is the activation value of the n units/neurons: $X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$
- W is the weight matrix: $W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \vdots & & \ddots & \vdots \\ w_{n1} & w_{n2} & \dots & w_{nn} \end{bmatrix}$ where w_{ij} can be interpreted as the influence of neuron i over neuron j (and reciprocally).
- T is the threshold of each unit: $T = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{pmatrix}$
- the $sign$ function is define as function 2.3:

$$sign(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{2.3}$$

Usually, an Hopfield Network has a weight matrix symmetric, zero-diagonal(no loop, a unit does not influence on itself). We will use the structure above in this project.

2.2 Hebbian Learning

A simple model due to Donald Hebb captures the idea of associative memory. Imagine that the weights between neurons whose activities are positively correlated are increased:

$$\frac{dw_{ij}}{dt} \sim \text{Correlation}(x_i, x_j) \tag{2.4}$$

Now imagine that when stimulus m is present (for example, the smell of a banana), the activity of neuron m increases; and that neuron n is associated with another stimulus, n (for example, the sight of a yellow object). If these two stimuli: a yellow sight and a banana smell. co-occur in the environment, then the Hebbian learning rule (2.4) will increase the weights w_{nm} and w_{mn}. This means that when, on a later occasion, stimulus n occurs in isolation, making the activity x_n large, the positive weight from n to m will cause neuron m also to be activated. Thus the response to the sight of a yellow object is an automatic association with the smell of a banana. We could call this pattern completion. No teacher is required for this associative memory to work. No signal is needed to indicate that a correlation has been detected or that an association should be made. The unsupervised, local learning algorithm and the unsupervised, local activity rule spontaneously produce associative memory[2, 3].

Similar with the example above, we could imagine each digits image as a complete graph which is shown in Figure 2.2. Each pixel in the image will be the neuron in Hopfield Network. We will use the idea of Hebbian learning to train the network which will change weights between each pair of neuron in the network.

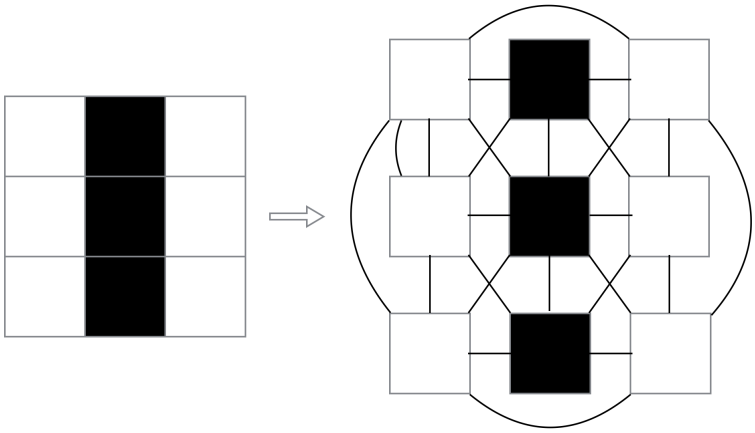


Figure 2.2: Example of digit 1 being denoted as Hopfield Network

In the Hebbian learning, the weight of the matrix results from a unsupervised learning, this learning type is a reinforcement learning. We consider that we can impress patterns to the network in order to recognize one of them even if they are given input with a lot of noise. The learning phase consists in learning different pattern in order to set the weight of the edges. For each pattern, we can resume the learning phase as:

- If two unit has the same activation state ({1, 1} or {-1, -1}), the strength of the connection is reinforce(w_{ij} increase).
- If two unit has opposite activation state ({-1, 1}), the strength of the connection decline.

With those kind of rule, W will store patterns and then allow the network to converge to them. Unfortunately, in that case we can not predict the convergence to the right pattern (the network will still converge but we cannot predict to which stable state).

The weight matrix is modified according to the m pattern we want to store. Those pattern represent the stable state we hope to reach. The weights are modified, for example:

$$w'_{ij} = w_{ij} + \epsilon x_i x_j \tag{2.5}$$

Equation 2.5 is iterated various time for each pattern. x_i and x_j represent the state of the units i and j of the pattern being learned. ϵ is a constant to prevent the connection to became too big or too small(For example: $\epsilon = \frac{1}{m}$)

When $m = 1$, we always converge to the impress pattern. When $m > 1$, we can only hope to converge to a known pattern if the different patterns are as orthogonal as possible (the different patterns have to be really different). If they are quite related it is possible to have a mix of different pattern.

We can increase the capacity of our network by modifying our objective function to measure how well it stores memories and minimizing that function. Consider the error between a single neuron of the network, with inputs $\mathbf{x}^{(n)}$ and binary labels $t^{(n)}$. We can define this error function $G(W)$ as follows:

$$G(W) = -\sum_i \sum_n t_i^{(n)} \ln y_i^{(n)} + (1 - t_i^n) \ln(1 - y_i^{(n)}) \tag{2.6}$$

where

$$t_i^{(n)} = \begin{cases} 1 & x_i^{(n)} = 1 \\ 0 & x_i^{(n)} = -1 \end{cases} \tag{2.7}$$

and

$$y_i^{(n)} = \frac{1}{1 + e^{-a_i^{(n)}}} \tag{2.8}$$

Now we can minimize $G(W)$ to determine our weights. However, we run the risk of over fitting our network. We will reduce this risk by making use of regularization. In this case we will minimize

$$M(W) = G(W) + \alpha E_w(W) \tag{2.9}$$

where α is a regularization parameter and $E_w(W)$ is a penalty function. it will act as a bias against W which will avoid the values becoming too large and over-fitting the data[5].

We will minimize the function by utilizing gradient descent, a first order optimization algorithm. Gradient descent works via a back-propagation algorithm whereby we modify w to move opposite the gradient of G. In this way we find the w which minimizes the error function.

2.3 Convergence of Hopfield Network

Hopfield networks converges to a local state. The energy function of a Hopfield network in a certain state is:

$$E_1 = \frac{1}{2} X^t W X + T X^t = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_i x_j + \sum_{i=1}^{n} \theta_i x_i \tag{2.10}$$

$$E_2 = \frac{1}{2} X^t W X = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_i x_j \tag{2.11}$$

Equation 2.10 can be useful as a general energy function. More often, we use equation 2.11. Equation 2.11 is equivalent to equation 2.10.

In order to take into account the threshold of the units, we add a artificial unit which is in the activation state 1. This unit is linked to the other with the value of their negative threshold. In that case, the new weight matrix will be

$$W_G = \begin{bmatrix} W & T \\ T^t & 0 \end{bmatrix} \tag{2.12}$$

Those energy functions are helpful to determinate the actual state of the network.

As the dynamic is asynchronous, only one unit is evaluated at each time. If the k^{th} unit is selected, there is two options:

1. k does not change state
2. k changes state

In the first case, the energy function remains the same, $E(t+1) = E(t)$. Otherwise, the energy function changes, according to the new excitation value (x'_k) of the k^{th} unit. The difference of energy of the system is given by:

$$E_1(x) - E_1(x') = (-\sum_{j=1}^{n} w_{kj} x_j x_k + \theta_k x_k) - (-\sum_{j=1}^{n} w_{kj} x_j x'_k + \theta_k x'_k) \tag{2.13}$$

Then we have:

$$E_1(x) - E_1(x') = -(x_k - x'_k) \cdot (\sum_{j=1}^{n} w_{kj} x_j - \theta_k) \tag{2.14}$$

The second term of equation 2.14: $(\sum_{j=1}^{n} w_{kj} x_j - \theta_k)$ is the excitation e_k of the k^{th} unit. The unit changed its state, so the excitation has a different sign than x_k and $-x'_k$ (according to equation 2.1). Therefore, $E_1(x) - E_1(x')$ is positive, which implies that $E(x) > E(x')$. So the energy will decrease for every change, and since there is only a finite number of possible states, the network should reach a state where the energy cannot decrease more, it will be a stable state.

Of course the stable state is not unique but it is certain that the system will converge. The final state will depend on the input, the initial state of the system. This conclusion is very useful since we could first train the Hopfield network to get multiple stable states that represent sample handwritten digits from dataset. Then the testing images which is also from dataset will be expected to converge to a stable state that we trained previously.

3 METHODOLOGY

3.1 DATA SET

The data set will be used in this project is the Semeion Handwritten Digit Data Set which was obtained from the UCI Machine Learning Repository[6]. The data set consists of 1593 writing samples taken from 80 people. Each participant was told to write two copies of each of

the digits zero through nine . One of the copies was supposed to be written as carefully or correctly as possible as shown in Figure 3.1, while the other copy was to be written as fast as possible, or with errors as shown in Figure 3.2. Each written digit was then fit into a 16×16 box, and was represented as a 256 elements array where each element is represented by binary data.

Figure 3.1: Sample digits which written carefully

Figure 3.2: Sample digits which written fast

3.2 Experiment Setup

As mentioned in section 3.1, each handwritten digits was fit into a 16×16 box, so the hopfield network in this project has 256 neurons. The size of corresponding weight matrix is 256×256 and all of weights on the diagonal is set to be zero since $w_{ii} = 0$. Also, we will use Hebbian learning rule to train the weights of Hopfield network.

Those carefully written digits will be implemented as the training set of Hopfield network. And those digits to be written fast will be used as test cases. Also, Hopfield network requires the data to be ± 1, however the Semeion handwritten data is a 0/1 binary data, so we need to convert the 0/1 binary data to ± 1.

In the following experiments, the project will start with storing two memories(digits 0 and 1) to test how well four different sample digits could convergent to the correct stable state. And then will look at the convergence of the same four sample digits with storing three memories (digits 0, 1 and 2), four memories (digits 0, 1, 2 and 3), and finally ten memories (digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9). The experiments will show the relationship between the number of memories stored in the network and the capacity of Hopfield network.

Finally, this project will try to improve the capacity of Hopfield network by defining an objective function that measures how well the network stores all the memories. I will run through the data set and record the percentage of correct convergences when storing two, three, five, nine and ten digits. This experiment is expected to show the how could we improve the capacity of Hopfield network by modifying its learning rule.

4 Experiment Results and discussion

4.1 Experiments based on Hebbian learning rule

4.1.1 Experiments results

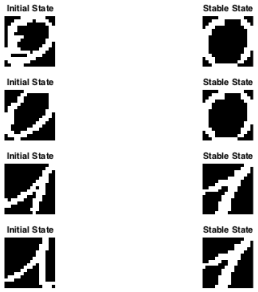


Figure 4.1: Experiment result of storing two memories(digits 0 and 1) in network

We can start with storing two memories(digits 0 and 1) in the Hopfield network first. As shown in the Figure 4.1, all of test cases are successfully convergence into the correct stable state.

In Figure 4.2, we can notice that, when we store 3 memories(digits 0, 1 and 2), all test cases could also convergence into the correct stable state.

As shown in Figure 4.3, When 4 memories(digits 0, 1, 2 and 3) stores in the Hopfield network, the first three test cases could still convergence to the correct stable state. However, the fourth

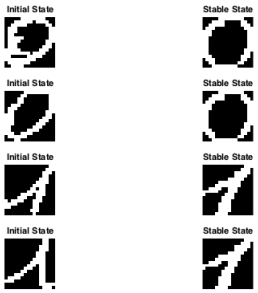


Figure 4.2: Experiment result of storing three memories(digits 0, 1 and 2) in network

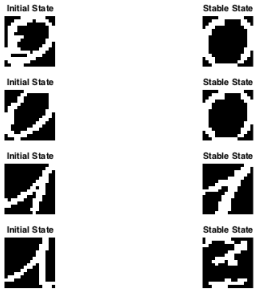


Figure 4.3: Experiment result of storing four memories(digits 0, 1, 2 and 4) in network

test case digit 1 converge into a spurious stable state which likes a mixture state of digit 2 and 3. It is possible that this mixture state could have a lower energy compare with sample digit 1 that we stored in Hopfield network.

We can squash more memories into the network. Figure 4.4 shows a set of five memories(digits 0, 1, 2, 3, 4) storing in the network. We can notice that the first two digit 0 test cases could still converge to the correct stable state. However, two of those digit 1 test case fail to converge to the correct stable state. Instead of converging to the sample digit 1 we stored in the network, they both converge into the same spurious stable state which is similar with sample digit 1 but have few pixels flipped around the top.

In figure 4.5, We continued add memories into the network up to ten, it can be seen that none of those test cases could converge to the corresponding stored memories. The first two digit 0 converge to same mixed stable state and the two digit 1 converge converge to a different mixed stable state which closely resembles a digit 7. We found the fact that test cases are tend to converge to some mixed stable states, and believe that these particular mixed stable states must be dominant when running our algorithm with this data set. They are most likely two of the lowest energy states in the entire network.

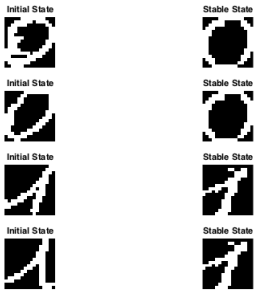


Figure 4.4: Experiment result of storing five memories(digits 0, 1, 2, 4 and 5) in network

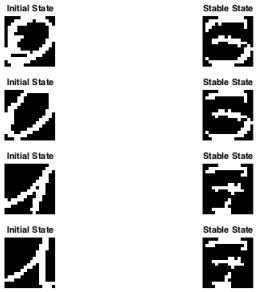


Figure 4.5: Experiment result of storing ten memories in network

4.1.2 Discussion about experiment results

After looking at the experiment results above, we saw an interesting trend. As shown in Figure 4.6, it is obvious that as the number of memories stored increases, the number of test cases that converge to the correct stored memory decreases. This trend was expected due to the variation of our data set, and the nature of Hopfield networks in general.

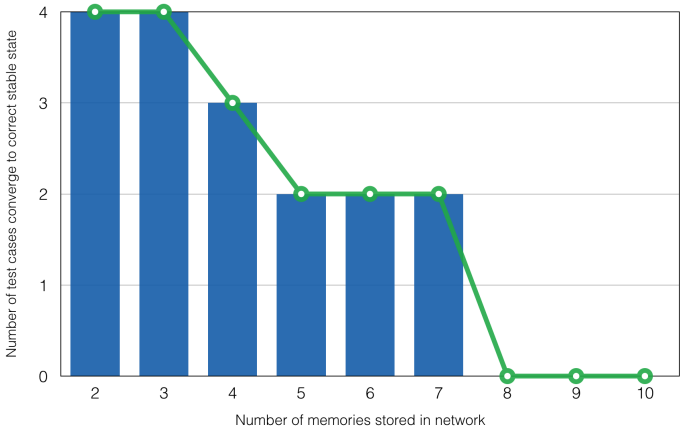


Figure 4.6: the number of test cases that converge to the correct stored memory decreases following the increasing number of memories stored in network

According to the experiment results, Hopfield network model can fail in various ways:

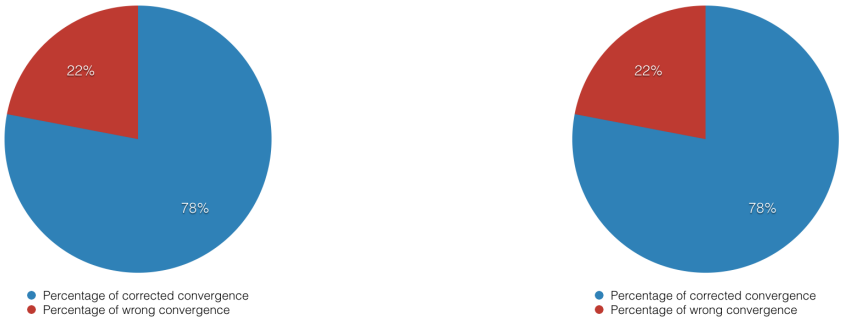
1. Individual bits in some memories might be corrupted, that is, a stable state of the Hopfield network is displaced a little from the desired memory.
2. Entire memories might be absent from the list of attractors of the network; or a stable state might be present but have such a small basin of attraction that it is of no use for pattern completion and error correction.
3. Spurious additional memories unrelated to the desired memories might be present.
4. Spurious additional memories derived from the desired memories by operations such as mixing and inversion may also be present.

Meanwhile, we also found that as the number of memories stored increases, the possibility of test cases converging into spurious stable states also increases. The reason behind it is that spurious states are tend to have lower energy than those memories(sample digits) we stored in the network when we stored too much memories in the network.

4.2 Experiment based on optimized weights

In the previous section, We observed that as the number of memories stored increases, the number of samples that converge to the correct stored memory decreases. To better illustrate this trend, I decided to run through the data set and record the percentage of correct convergences when storing two, three, five, nine and ten digits. Also, I will implement the objective function (2.6) to optimize the weights matrix and compare its performance with standard weights matrix based on Hebbian learning rule, in hopes of seeing any kind of improvement

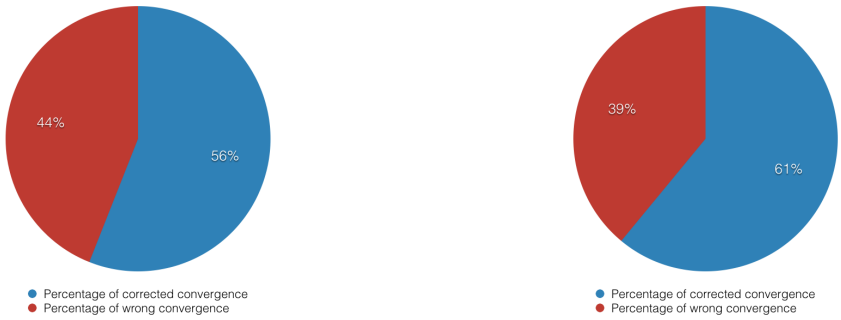
when using the optimized weights.



(a) Standard weights (b) Optimized weights

Figure 4.7: Summary results with two memories

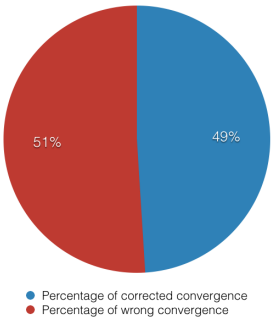
Looking at the first two pie charts in Figure 4.7. With only two memories stored in network, it can be seen that the percentage correct using the standard weights and optimized weights is the same at 78%.



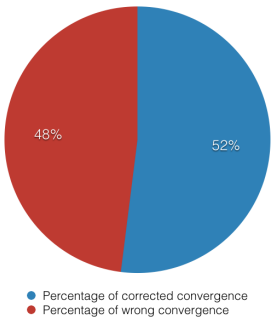
(a) Standard weights (b) Optimized weights

Figure 4.8: Summary results with three memories

In Figure 4.8, when storing only three memories, that percent correct for the standardized weights drops to 56%, and the percent correct for the optimized weights jumps to 61%. Thus both percentages have dropped, but it can be seen that the optimized weights is performing slightly better.



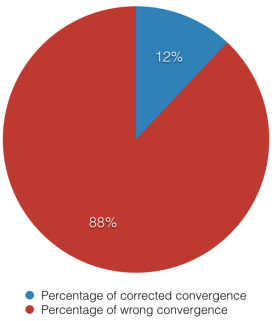
(a) Standard weights



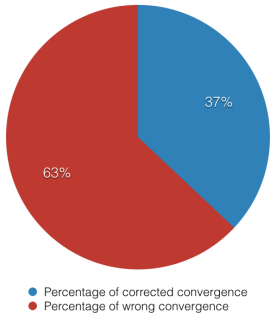
(b) Optimized weights

Figure 4.9: Summary results with five memories

Looking at Figure 4.9, with five memories stored, the percent correct for the standard weights drops to 49% and for the optimized weights drop to 52%. This is a smaller gap between the performance of the standard vs optimized weights, but the optimized weights are still outdoing the standard weights.



(a) Standard weights



(b) Optimized weights

Figure 4.10: Summary results with nine memories

In Figure 4.10, with nine memories stored, the percent correct plummets to 12% for the standard weights and 36% for the optimized weights. This is a much more noticeable gap between the performance of the standard and optimized weights.

(a) Standard weights

(b) Optimized weights

Figure 4.11: Summary results with ten memories

Similarly, in Figure 4.11, with ten memories stored, the percent correct drops to 10% for the standard weights and remains at 36% for the optimized weights. This again shows that the optimized weights perform better than the standard weights.

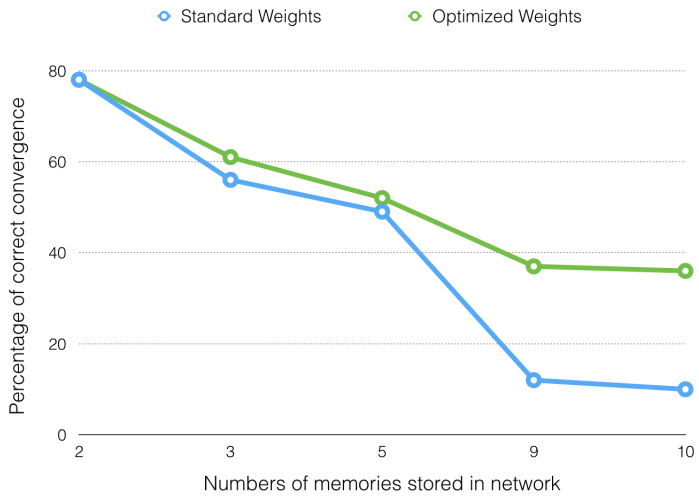


Figure 4.12: Summary of experiemnt results

In sum, as shown in Figure 4.12, as the number of memories stored in network increases, the percentage of test cases that converge into correct stable state decreases. Also, the optimized weights does have better performance than standard weights.

5 CONCLUSTION

Hopfield network is an easily implemented neural network that can effectively demonstrate associative memory. Partial or corrupted memories can be restored to their correct state. In addition, inputs that are sufficiently close to a stored memory will converge to the desired memory. The choice of weights is key in ensuring the highest rates of accuracy. Hebbian learning optimized via gradient descent seems to be the best choice to optimize the weights while preventing over fitting of the neural network.

REFERENCES

[1] P. Solanki, M. Bhatt, *Printed Gujarati Script OCR using Hopfield Neural Network*, International Journal of Computer Applications(0975-8887), 2013.

[2] David J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*. New York, NY: Cambridge University Press, 2003.

[3] P. K. Sarangi, A. K. Sahoo, P. Ahmed, *Recognition of Isolated Handwritten Oriya Numerals using Hopfield Neural Network*, International Journal of Computer Applications(0975-8887), 2012.

[4] V.M. Vieira, M. L. Lyra, C. R. da Silva, *Recognition ability of the fully connected Hopfield neural network under a persistent stimulus field*, Physica A 388 (2009) 1279-1288, 2008.

[5] Alan Julian Izenman, *Modern Multi-variate Statistical Techniques:Regression, Classification, and Manifold Learning.*, New York, NY: Springer, 2008.

[6] Massimo Buscema, *Semeion Handwritten Digit Data Set*, Irvine, CA: UCI Machine Learning Repository [http://www.ics.uci.edu/ mlearn/MLRepository.html], 2003.

Appendix

Source code of this project could be downloaded from following link:

https://github.com/turbofsi/ece1512-HNN

The platform of this project is on Matlab. To run this program properly, *Semeion Handwritten Digit Data Set* should be downloaded beforehand and put it at same directory with all the other source code files. ex1.m is corresponding with experiment 1, and ex2.m is source code of experiment 2.