UNIVERSITY OF TORONTO, DEPARTMENT OF ECE

ECE1512 - Project Report
Hopfield Network for Character Recognition

Yang Wang
Student Number: 1001319227

November 27, 2015

1 INTRODUCTION

Neural networks have received increasing usage in the areas of pattern recognition. One of the important applications is in character recognition. Character recognition has many practical interests, such as zip code recognition, document ananlysis. Many questions are usually involved, such as the choice of data representation, the design of classification algorithm and the selection of training data. In using neural net classifier, many resarchers were choosing the multi-layer feedforward model with backpropagation algorithms. In this report, we choose to investigate the handritten character(isolated digits) recognition project by using the discrete Hopfield network model as a pattern classifier. The data representation is based on image pixels. This type of representation is simple and straightforward.

Neural networks process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements working in parallel to solve a specific problem. Neural networks learn by example. A neuron has many inputs and one output. The neuron has two modes of operation:

(a) Training mode. In the training mode, the neuron can be trained for particular input patterns.

(b) Using mode. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output.

If the input pattern does not belong in the taught list of input patterns, the training rule is used. Neural network has many applications. The most likely applications for the neural networks are classification, association and reasoning. An important application of neural networks is pattern recognition. Pattern recognition can be implemented by using a feed-forward neural network that has been trained accordingly. During training, the network is trained to associate outputs with input patterns. When the network is used, it identifies the input pattern and tries to output the associated output pattern.

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 an other unit j with a certain weight w_{ij}, and have a threshold value.

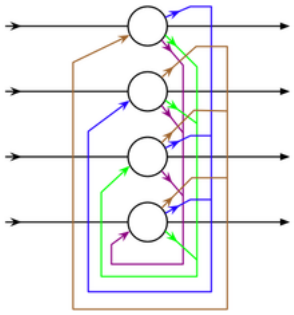

Figure 2.1: An example of Hopfield Network

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

The new activation value (state) of a neuron is compute, in discret 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 symmetrix, zero-diagonal(no loop, a unit does not influence on itself). We will only consider that case in our 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 as- sociation should be made. The unsupervised, local learning algorithm and the unsupervised, local activity rule spontaneously produce associative memory.

Similar with the example above, we could imagine each standard character 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.

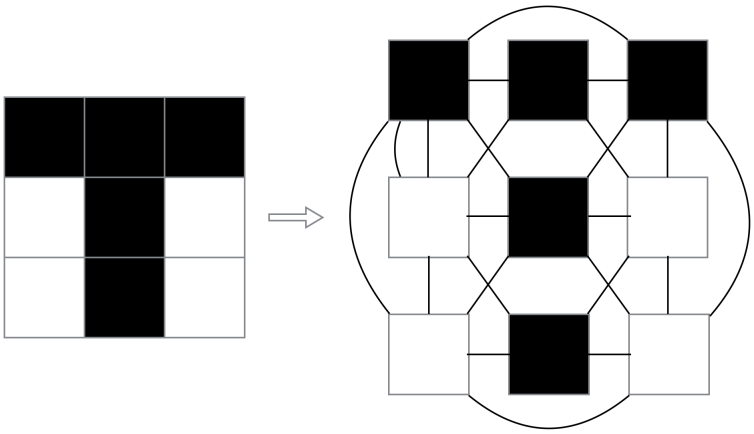

Figure 2.2: Character T being denoted as Hopfield Network

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 âĂİimpressâĂİ patterns to the network in order to recognize one of them even if they are given in 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 to big or to 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 overfitting 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. This regularizer will act as a bias against W which will keep the values from becoming too large and overfitting the data.

We will be minimizing the function by utilizing gradient descent, a first order optimization algorithm. Gradient descent works via a backpropagation 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

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 state that stand for specific character or digits. Then the input image will be handwritten character or digits with noise and they will be expected to converge to a stable state that we trained previousely.

3 METHODOLOGY

3.1 DATA SET

The data set used in this project is the Semeion Handwritten Digit Data Set which was obtained from the UCI Machine Learning Repository. 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 âĂİcarefullyâĂİ or correctly as possible as shown in Figure 3.1, while the other copy was to be written as âĂİfastâĂİ 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 element 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 written 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 diagnal 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 require the data to be ± 1, however, The Semeion handwritting 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 looked 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.

Finaly, we will try to improve the capacity of hopfield network by defining an objective function that measures how well the network stores all the memories.

4 Experiment Results and discussion

5 Conclustion