Pre-course Lab — The Hopfield Model

CS6482: Neural Networks

February 2, 2003

1 INTRODUCTION

In this lab, you will run simulations of a Hopfield model. You will gain experience with the dynamics of this system, train it to store patterns and act as an auto-associative memory, and use it for very rudimentary image processes.

2 TASKS

In this section, I describe the questions which I want you to investigate in the lab. In the next section, titled "resources", I describe how you are to go about it using matlab.

2.1 Dynamics

The Hopfield model consists of a network of nodes which take the values -1 or $+1$. Let N denote the number of nodes and x_i denote the value (± 1) of node i. The nodes are updated one at a time in the following way. For each timestep: pick a node i at random. Update it according to the following rule:

$$x_i \to \begin{cases} 1; & \text{if } \sum_j^N w_{ij}x_j + b_i > 0 \\ -1; & \text{if } \sum_j^N w_{ij}x_j + b_i \leq 0 \end{cases} . \tag{1}$$

Here w_{ij} is the weight from j to i, and b_i is the bias at i. Hopfield noted that if $w_{ii} = 0$ and $w_{ij} = w_{ji}$, this dynamics would always converge to fixed points.

Problems

1. Set up a small network with weights that satisfy Hopfield's conditions. Find a set of weights that do not satisfy Hopfield's conditions and with which the network does not converge. (This is done as a example in section 1.3.3.)

2. Suppose you want to make a system which converges to the state with all nodes 1 if the majority of the nodes initially were 1's, and to the state with all nodes -1 of the majority of nodes in the initial states was -1. That is, you want a network which computes whether the density of 1's is greater to one half. What weights would you use? Does it always work? Simulate it to see how it behaves.

3. Suppose you have the same problem as above. However, the nodes are connected in a line, and each node is only connected to its two nearest neighbors (one on each side)? What weights would you use? Run this system and see how its behaviour differs from that above.

2.2 Auto-Associative Memory

A Hopfield model can function as an auto-associative memory. The weights can be determined by a Hebbian rule. To add the pattern F_i^p to the current memory, change the weights as follows,

$$w_{ij} = w_{ij} + F_i^p F_j^p. \tag{2}$$

A set of patterns are present in /opt/info/courses/NeuralNets/Hopfield directory. They are called dat0, dat1, dat2, dat3, dat4, dat6, dat9, and datdot (and have been stolen from Haykin's book). These are 120 component vectors which represent 12 by 10 images of characters.

Problems

1. Store some of these patterns in the Hopfield model. See if it is true that if the network is initialized to a noisy version of one of these patterns, the dynamics will evolve to the nearest original stored pattern. (An example showing how to do this in matlab is in section 1.3.4.)

2. How does the performance depend upon the number of stored patterns?

3. How does the performance depend upon the amount of noise in the initial pattern?

2.3 Image Restoration

A Hopfield model can be used for cleaning up images without storing them explicitly, if the appropriate assumptions are made about the nature of the images. The network should have weights and biases which have the property that images "like" the images to be restored are stable in the network, while images "like" noisy versions of the images will evolve under the network dynamics to the stable ones. Perhaps you will need to think about what properties an image will have to be like the images to be restored.

Problem

1. In the /opt/info/courses/NeuralNets/Hopfield directory, there is a set of images, called stripes1, stripes2, ..., etc. These consist of a few straight edges. Devise a set of weights and thresholds which restores noisy versions of these patterns. Test these.

3 RESOURCES — THE HOPFIELD MODEL IN MATLAB

3.1 Representation

Here I describe how to represent weights and node values for the Hopfield model in MATLAB. It is much like what you did in the pre-course work, except there will be many nodes instead of one, and there will be no batching of patterns.

Network values: This is an $N \times 1$ matrix of ± 1. For example, the outputs of a three node network could be (as MATLAB would them),

```
x =
    -1
     1
     1
```

Weights: This is an $N \times N$ matrix. According to Hopfield's conditions, it must be symmetric ($w_{ij} = w_{ji}$) and zero along the diagonals ($w_{ii} = 0$). For example, the weights for a three node networks could be,

```
w =
     0     1     1
     1     0     1
     1     1     0
```

Biases: This is an $N \times 1$ matrix containing b_i. Often, the biases will all be 0. For a three node example, this would be,

```
b =
    0
    0
    0
```

3.2 Hopfield Simulation Commands

In the matlab neural network toolbox, there is a set of alleged Hopfield simulation commands which are described in chapter 9 of the Neural Network Toolbox Users Guide. However, these implement parallel updating which means that all nodes are updated in each timestep. Thus, convergence is not guaranteed. In addition, these act on continuous varying neurons. In other words, they do not really simulate the Hopfield model at all, but a related model. I have written a new Hopfield simulation command, which is described here.

simuhoptrue: This simulates a Hopfield network using the correct dynamics. It can be used in two ways:

```
af = simuhoptrue(a,w,b,ts);
```

where a is the initial values of the network, w is the weight matrix, b is the bias vector, and ts is the number of timesteps that the simulation is to run. The output, af, is a vector of network outputs after the simulation is run. If simuhoptrue is invoked with only three arguments, ts takes the default value of 1.

```
[af,aaf]=simuhoptrue(a,w,b,ts);
```

This is as above. In addition, aaf contains the network outputs for all timesteps.

In addition, some other commands are useful in producing and viewing Hopfield simulations.

displayhop: This is a command which I produced to display the results of simuhoptrue. This displays the output of the network as a vertical grid, with black squares representing -1 and white representing $+1$. Each timestep of the dynamics is displayed. This would be used with simuhoptrue as follows:

```
[af,aaf]=simuhoptrue(a,w,b,20);
displayhop(aaf);
```

ones, zeros, eye: These are built-in matlab commands. ones(n,m) makes an $n \times m$ matrix of all ones; zeros(n,m) makes an $n \times m$ matrix of all zeros, and eye(n) makes a square $n \times n$ matrix with 1's on the diagonal and zeros elsewhere. These will be useful in creating weight matrices and so forth. Suppose you wanted to make a ten node Hopfield network in which each node was connected to each other with a weight of -1. Rather than typing in the 10×10 weight matrix by hand, you could use

```
w=eye(10)-ones(10,10);
```

(eye is used to set the diagonal elements to zero).

3.3 Examples:

As an example of the simulation of a Hopfield model in MATLAB, we shall do task 1 from the dynamics tasks. First, let us set up a very simple network with two nodes and weights which satisfy Hopfield's conditions.

- Define a weight matrix which satisfies Hopfield's conditions.

```
w = [0 1;
     1 0];
```

- Define the bias vector. For this example, we use a 2×1 matrix of zeros.

```
b=zeros(2,1);
```

- Set the initial state of the network. I have chosen this arbitrarily.

  ```
  x=[-1;1];
  ```

- Run the simulation. This can be done by typing

  ```
  x=simuhoptrue(x,w,b)
  ```

  over and over (the control sequence ^p makes the last command the current command). Or, the simulation can be run for 20, say, timesteps and then via,

  ```
  [x,aa]=simuhoptrue(x,w,b,20);
  ```

  The results are then viewed by looking at the numbers in aa, or by

  ```
  displayhop(aa);
  ```

  The latter will show the value of the nodes as a function of time.

If you run this example using simuhop rather than simuhoptrue, you will see how the dynamics is different using parallel updating.

Now we will see what happens when one of the Hopfield conditions are violated. Set up a two node network with weights from node 1 to node 2 as 1, but the weight from node 2 to node 1 is -1.

```
w=[0 -1;1 0];
b=[0;0];
x=[1;1];
[x,aa]=simuhoptrue(x,w,b,40);
displayhop(aa);
```

Does the network converge?

3.4 Auto-associative Memory

I need to tell you more commands.

load: This is used to load in the image files.

```
load dat0;
```

reads in the file called dat0 and makes a pattern vector also called dat0.

learnh: The Hopfield model uses Hebbian learning, so the Hebb rule command can be used. The Hopfield model is auto-associative, which means that the association is between the pattern and itself. To add the memory of pattern dat0 to already existing patterns in the model, the command would be

```
w=w+learnh(dat0,dat0,1);
```

addnoises: This is a command to add noise to a ± 1 pattern. This changes the sign of components of the pattern at random. The usage is,

```
a=addnoises(dat0,0.3);
```

here dat0 is the pattern to be noised up, a is the noisy version of the pattern, and the second argument to addnoises is the noise probability. If it is 0, no noise is added, if it is 1 every component is changed, if it is 0.5, half the components are changed on average, and so forth.

showpattern2d: The patterns represent 2 dimensional images. To see the pattern as a 12×10 image, use the command

```
showpattern2d(dat0,10,12);
```

(you can also look at the noisy images this way). The second and third argument to the command are the height and width respectively. Note: matlab has by default one window to display plots and images. To view more than one image simultaneously, the command **figure** will open a new window for displaying plots and images.

Displaying the pattern overlaps: Since it is difficult to view changes to a 120 component vector, a simpler method of viewing the dynamics is required. One method is to plot the similarity between the pattern being recalled and the current state of the network. A common similarity measure is the overlap,

$$m = \frac{1}{N} \sum_i x_i F_i^p, \tag{3}$$

where x_i is the current state of the network and F^p is the pattern to be recalled. The terms in the sum, $x_i F_i^p$, will be 1 where the patterns are the same and -1 where they differ. Thus, m will be 1 if the network recalls the pattern perfectly, 0 if the recalled pattern is unrelated to the stored one, and -1 if the recalled pattern is the opposite of the stored one. The sum can be computed using the MATLAB matrix transpose operator " ' " and matrix multiply.

```
[a,aa]=simuhoptrue(a,w,b,100);
plot(aa'*dat0/120);
```

anihop2d: As an alternative to the above, I have produced another Hopfield command which shows the state of the network as a two dimensional image at various times during the dynamics. Thus, this shows the Hopfield dynamics as an animation. The usage is

```
aa = anihop2d(a,w,b,ts,height,width,iterations);
```

Here a, w, and b are as for simuhoptrue. The parameter ts is the number timesteps between displays. height and width are the dimensions of the images to be displayed, they will be 10 and 12 for these images. The total number of iterations is determined by the final argument, iterations; if this is missing, it defaults to 100. The final state of the network is returned.

As an example of the use of these commands, I show how to store one pattern in a Hopfield model. Comments on the commands follow %, obviously, you won't type these comments in.

```
load dat0;
showpattern2d(dat0,10,12); % if you want to see what it looks like
w=zeros(120,120);          % initialize the weights to zero
b=zeros(120,1);
w=w+learnh(dat0,dat0,1);   % Hebbian Learning
a=addnoises(dat0,0.3);     % See if Hopfield model can store correct
                           % pattern if input is a noisy one.
[a,aa]=simuhoptrue(a,w,b,200);
plot(aa'*dat0/N);
```

Alternatively, the last two commands could be replaced by anihop2d

```
af=anihop2d(a,w,b,80,10,12,100);
```

3.5 Image Restoration

No new commands are required to do this task. The stripe images are 16×16 images, so showpattern2d and anihop2d may be useful. The trick is to come up with an appropriate set of weights and biases.