

## ***Associative memory and Hopfield Neural Network***

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-1

## **Content of this lecture**

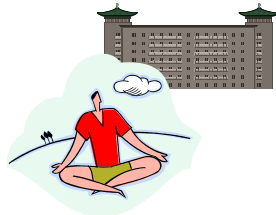
- What is an associative memory?
- What is a Hopfield neural network (HNN)?
- Energy function and property of HNN.
- Method for storing and re-calling patterns using HNN.
- A brief introduction to the Boltzmann machine (BM).

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-2

## **What is associative memory ?**

- Conventional memory in computers
  - Address based memory
- Associative memory
  - Content based memory
  - Keyword based memory
  - Pattern based memory



Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-3

## **Auto-associative memory**

- key  $\rightarrow$  key
- De-noising
- Reconstruction of patterns based on similarity.

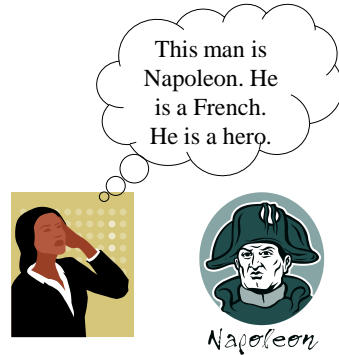


Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-4

## Hetero-associative memory

- key  $\rightarrow$  content
- Recognition of patterns based on hints or features.

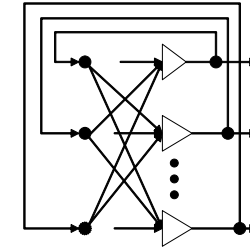


Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-5

## The Hopfield neural network

- Hopfield neural network (HNN) is a model of auto-associative memory.
- The structure is shown in the right figure.
- It is a single layer neural network with feedbacks.

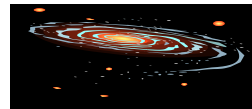


Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-6

## Some terminologies

- The output of each neuron is a binary number in  $\{-1, 1\}$ .
- The output vector is the **state vector**.
- Starting from an initial state (given as the input vector), the state of the network changes from one to another like an automaton.
- If the state converges, the point to which it converges is called the **attractor**.



Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-7

## The state-transition mechanism

Suppose that the current state of the network is

$$\mathbf{v}^k = [v_1^k, v_2^k, \dots, v_n^k]$$

then, the next state can be calculated by

$$v_i^{k+1} = \text{sgn}(\text{net}_i) = \text{sgn}\left(\sum_{\substack{j=1 \\ j \neq i}}^n w_{ij} v_j^k + \theta_i\right)$$

where  $\theta_i$  is the threshold of the  $i$ th neuron

- Note that the update is **asynchronous**. That is, one neuron is updated each time, and the update order is random.
- Note also that  $w_{ij} = w_{ji}$  and  $w_{ii} = 0$  for all  $i$ .

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-8

## The energy function

- We can define an energy function for each HNN.
- For the network shown before, the energy function  $E$  is defined by

$$\begin{aligned} E &= -\frac{1}{2} \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n w_{ij} v_i v_j + \sum_{i=1}^n \theta_i v_i \\ &= -\frac{1}{2} \mathbf{v}^T \mathbf{W} \mathbf{v} + \Theta^T \mathbf{v} \end{aligned}$$

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-9

## Feature of the energy function

- The energy function  $E$  is a function of the state vector  $\mathbf{v}$ .
- For an HNN, its energy function never increases during state transition.
- That is, the value of  $E$  at the initial state is always larger than that at the attractor.

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-10

## Increment of $E$

By updating the state, we get an increment

$$\Delta \mathbf{v} = \mathbf{v}^{k+1} - \mathbf{v}^k$$

Using Taylor extension, we can find the increment of  $E$  by

$$\Delta E = E^{k+1} - E^k = (\nabla E)^T \Delta \mathbf{v} + \varepsilon$$

where  $\varepsilon$  is the 1st order approximation error, and can be omitted.

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-11

## Increment of $E$ (cont.)

The gradient of  $E$  can be calculated by

$$\nabla E = -\frac{1}{2}(\mathbf{W}^T + \mathbf{W})\mathbf{v} + \Theta$$

The weight matrix  $\mathbf{W}$  is supposed to be symmetric, and thus we have

$$\nabla E = -\mathbf{W}\mathbf{v} + \Theta$$

Thus, the increment of  $E$  becomes

$$\Delta E = -(\mathbf{W}\mathbf{v} - \Theta)^T \cdot \Delta \mathbf{v}$$

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-12

## Increment of E (cont.)

Since the state transition is asynchronous, only the  $i$ -th ( $i$  is random) output is changed each time.

The state increment is actually given by

$$\Delta \mathbf{v} = (0, \dots, \Delta v_i, \dots, 0)^T$$

Therefore,

$$\Delta E = -\left(\sum_{j=1}^n w_{ij} v_j - \theta_i\right) \Delta v_i = -net_i \cdot \Delta v_i$$

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-13

## Increment of E (cont.)

We have the following relations :

$$\text{when } net_i < 0, \Delta v_i \leq 0$$

$$\text{when } net_i > 0, \Delta v_i \geq 0$$

Thus,  $\Delta E = -net_i \Delta v_i$  is always negative, and thus  $E$  will never be increased by state transition.

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-14

## Why Hopfield neural network is an associative memory ?

- Starting from any initial state, the HNN will change its state until the energy function approaches to the minimum.
- The minimum point is called the **attractor**.
- Patterns can be stored in the network in the form of attractors.
- The initial state is given as the input, and the state after convergence is the output.

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-15

## How to store the patterns ?

Suppose that we have  $p$  patterns to be stored.

We can calculate the weight matrix as follows :

$$\mathbf{W} = \sum_{m=1}^p \mathbf{s}^m (\mathbf{s}^m)^T - p\mathbf{I}$$

where  $\mathbf{s}^m$  is the  $m$ -th pattern (a column vector), and  $\mathbf{I}$  is the unit matrix. The thresholds of all neurons are set to zeros.

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-16

## How to store the patterns (cont.)?

If the pattern takes value from  $\{-1, 1\}$ , the weights are given by

$$w_{ij} = (1 - \delta_{ij}) \sum_{m=1}^p s_i^m s_j^m$$

where  $\delta_{ij}$  is the Kronecker function defined by

$$\delta_{ij} = \begin{cases} 1 & i=j \\ 0 & \text{otherwise} \end{cases}$$

If the pattern takes value from  $\{0, 1\}$ , we have

$$w_{ij} = (1 - \delta_{ij}) \sum_{m=1}^p (2s_i^m - 1)(2s_j^m - 1)$$

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-17

## How to use a HNN ?

- Phase 1: Store all patterns into the network by finding the weight matrix as above.
- Phase 2: Recall a pattern when an input is given as the initial state.

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-18

## Phase 1: Storage

- Step 1 Initialization:  $W=0$
- Step 2 Store the  $m$ -th pattern  $s^{(m)}$  by  $W = W + s^{(m)}(s^{(m)})^t$
- Step 2 is repeated for all patterns.
- After all patterns are stored, set  $w_{ii}=0$  for  $i=0, 1, \dots, n$ .

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-19

## Program for Storage

```
Store_Pattern(){
    int i,j,k;

    for(i=0; i<n_neuron; i++){
        for(j=0; j<n_neuron; j++){
            w[i][j]=0;
            if(i==j) continue;

            for(k=0; k<n_pattern; k++){
                w[i][j] += s[k][i]*s[k][j];
            }
            w[i][j]=w[i][j]/n_pattern;
        }
    }
}
```

There is NO feedback from one neuron to itself

Store the patterns into the network using previous equation

Normalize the weights

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-20

## Phase 2: Recall

- Step 1: The keyword is given as the initial state.
- Step 2: Select one of the neurons at random, find the output using the current outputs.
- Step 3: If for many neurons, the new outputs are the same as the old ones, the network converges, and the current output is the pattern to be recalled.
- Step 4: If the new output is different from the old one, return to Step 2.

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-21

## Program for recall

```
Recall_Pattern(int v[ ]){
    int i,i0,j,k, n_update, net, vnew;

    for(k=1,n_update=0; ;k++){
        for(i=0; i<n_neuron; i++){
            i0=rand()%n_neuron;

            for(j=0,net=0; ; j<n_neuron; j++) net+=w[i0][j]*v[j];
            vnew=(net >= 0)? 1:-1;

            if(vnew != v[i0]) { n_update++; v[i0]=vnew; }
        }
        if(n_update==0) break;
    }
}
```

Select a neuron to update at random or one by one

Find the new output of the i0-th neuron

Stop if there is no change after many iterations

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-22

## Example of associate memory

```
*****
*****
** **
** **
** **
** **
** **
** **
*****
*****

*****
*****
** **
** **
** **
** **
** **
** **
*****
*****

**
**
**
**
**
**
**
**
*****
*****

*****
*****
** **
** **
** **
** **
** **
** **
*****
*****
```

- We have four patterns given on the left.
- The purpose is to save them into a Hopfield neural network, and then recall them when they are corrupted by noises.

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-23

## An example of results

```
*****
*****
*** **
** **
** **
** **
** **
** **
** **
*****
*****

*****
*****
** **
* **
*** **
** **
** **
** **
** **
*****
*****

*****
*****
** **
* **
*** **
** **
** **
** **
** **
*****
*****

*****
*****
** **
** **
** **
** **
** **
** **
** **
*****
*****
```

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-24

## How to see the usefulness of the Hopfield neural network?

- As an associative memory, an HNN can only store  $0.14n$  patterns, where  $n$  is the number of neurons.
- The HNN, however, can be useful for signal or image restoration.
- It may also be useful for solving combinatorial optimization problems.

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-25

## Boltzmann machine

- Boltzmann machine (BM) is another type of neural network, proposed by G. Hinton and T. Sejnowski (1985).
- The structure is the same as HNN with the following restrictions:
  - There is no feedback from a neuron to itself.
  - The connection matrix is symmetric.
- The energy function can also be defined in the same way as in HNN.

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-26

## BM is a stochastic network

- In a BM, a neuron fires according with a probability given by

$$p(v_k = 1 | \mathbf{net}_k) = \frac{1}{1 + \exp(-\frac{\mathbf{net}_k}{\tau})}$$

where  $\tau$  is called the temperature.

- That is, the neuron may not fire even if the effective input is larger than the threshold.
- The property might be useful to get the global optimal solution when we use the network to solve combinatorial problems.

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-27

## Property of the BM

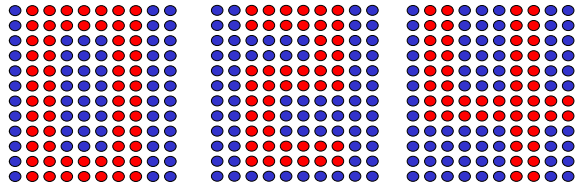
- If we run BM by repeatedly choosing a neuron and finding its next state with the above mentioned formula, the probability of the state will become stationary, and will follow the Boltzmann distribution.
- To guarantee the convergence of the state, we should set a relatively high temperature, and reduce it gradually.
- This process is called **simulated annealing**.

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-28

## Team Project III

- Suppose that a Hopfield neural network contains 12x10 neurons. We want to store some patterns given as follows (not necessarily the same) into the network, and recall any of them with a pattern corrupted by noise.



Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-29

## Team Project III (cont.)

- Write a computer program to implement the algorithm.
- Try to recall the patterns with the noise level being 0%, 10% or 15%.
- We say a pixel is a noise if its value is changed from 1 (or 0) to 0 (or 1).

Produced by Qiangfu Zhao (Since 1997), All rights reserved ©

Lecture 4-30