Neural Networks

Lecture 8

Hopfield Networks (1)

Md. Mijanur Rahman, Prof. Dr.

Dept. of Computer Science and Engineering, Jatiya Kabi Kazi Nazrul Islam University, Bangladesh.

Web: www.mijanrahman.com | Email: mijanjkkniu@gmail.com; mijan@jkkniu.edu.bd

Chapter: Hopfield Networks

- **This chapter covers the following topics:**
  - Introduction to Hopfield Networks
  - Basic Hopfield Net and How It works
  - Updating Rule in Hopfield Network
  - Discrete Hopfield Network
  - Continuous Hopfield Network

Introduction…

- A Hopfield network is a special kind of neural network whose response is different from other neural networks. It is calculated by a converging iterative process.
- It has just one layer of neurons relating to the size of the input and output, which must be the same.

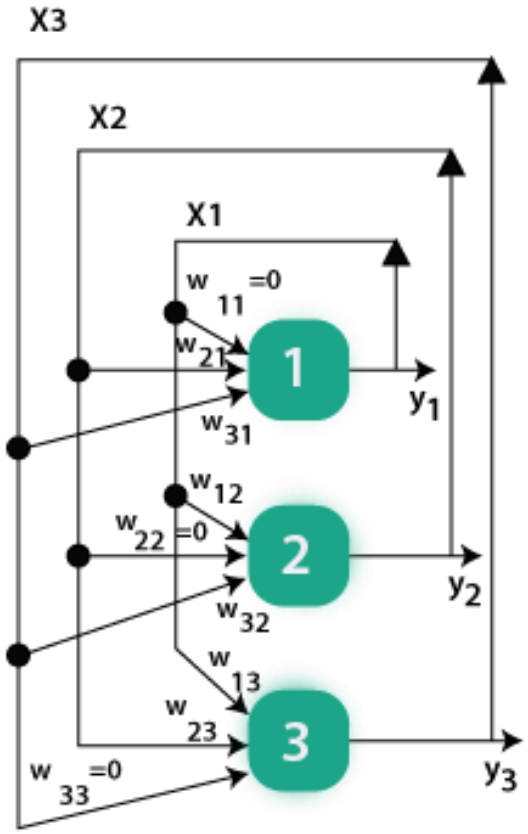
- When such a network recognizes, for example, digits, we present a list of correctly rendered digits to the network.
- Subsequently, the network can transform a noisy input into the related perfect output.

Introduction

- In 1982, **John Hopfield** introduced an artificial neural network to collect and retrieve memory like the human brain.
- Here, a neuron is either on or off in the situation. The state of a neuron (on +1 or off 0) will be restored, relying on the input it receives from the other neuron.
  - A Hopfield network is at first prepared to store various patterns or memories.
  - Afterward, it is ready to recognize any of the learned patterns by uncovering partial or even some corrupted data about that pattern, i.e., it eventually settles down and restores the closest pattern.
- Thus, similar to the human brain, the **Hopfield Model** has stability in pattern recognition.

Hopfield Networks…

- A Hopfield network is a single-layered and recurrent network in which the neurons are entirely connected, i.e., each neuron is associated with other neurons.
- If there are two neurons i and j, then there is a connectivity weight $\mathbf{w_{ij}}$ that lies between them, which is symmetric, $\mathbf{w_{ij} = w_{ji}}$.
- With zero self-connectivity, $\mathbf{w_{ii} = 0}$. In the figure, the three neurons with values i = 1, 2, 3 and values $\mathbf{x_i}$ = ±1 have connectivity weights $\mathbf{w_{ij}}$.



Hopfield Networks…

- In a Hopfield Network, the updating rule determines how neuron states are updated over time until the system reaches a stable state (an energy minimum).
- There are two main types of updating rules:
  - **Synchronously:**
    - In this approach, the update of all the nodes takes place simultaneously at each time.
  - **Asynchronously:**
    - In this approach, at each point in time, update one node chosen randomly or according to some rule. Asynchronous updating is more biologically realistic.

Hopfield Networks…

- **Asynchronous (Sequential) Updating Rule:**
- Neurons are updated one at a time. At each step, a neuron is selected and updated based on the current states of all other neurons. This is the most common and stable approach.

Update Formula:

$$s_i^{new} = \text{sign}\left(\sum_{j=1}^{N} w_{ij} s_j - \theta_i\right)$$

Where:

- s_i = state of neuron i (usually +1 or −1)
- w_{ij} = weight between neuron i and j
- θ_i = threshold (often 0)
- sign(x) = +1 if x ≥ 0, otherwise −1

Hopfield Networks…

- **Synchronous (Parallel) Updating Rule:**
  - All neurons are updated simultaneously.
  - Uses the same formula, but applied to all neurons at once.

  - Limitation:
  - May cause oscillations (network may not converge).
  - Less commonly used in practice.

Hopfield Networks…

- **Energy Function:**
  - Hopfield networks minimize an energy function:

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} s_i s_j + \sum_i \theta_i s_i$$

  - Each update reduces energy (in asynchronous mode).
  - Stable patterns = local minima of energy.

Hopfield Networks…

- **Weight Update Rule (Learning Rule):**

  – Before updating states, weights are usually set using Hebbian learning:

$$w_{ij} = \frac{1}{N} \sum_{p=1}^{P} x_i^p x_j^p$$

Where:

- x^p = stored patterns
- P = number of patterns

Example Setup

- A step-by-step numerical example of the Hopfield Network updating rule using a small system.
  - We use a network with 3 neurons.
  - Stored Pattern (Memory): $x = [1,-1,1]$

Step 1: Compute Weight Matrix (Hebbian Rule)

$$w_{ij} = x_i \cdot x_j \quad (\text{and } w_{ii} = 0)$$

Outer product:

$$W = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

Set diagonal to 0:

$$W = \begin{bmatrix} 0 & -1 & 1 \\ -1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$$

Example Setup

- A step-by-step numerical example of the Hopfield Network updating rule using a small system.

Step 2: Initial (Noisy) Input

Suppose the network receives a distorted pattern:

$$s = [1, 1, -1]$$

Step 3: Asynchronous Updating (One Neuron at a Time)

We update neurons one by one using:

$$s_i^{new} = \text{sign}\left(\sum_j w_{ij} s_j\right)$$

Example Setup

- A step-by-step numerical example of the Hopfield Network updating rule using a small system.

✅ **Update Neuron 1**

$$h_1 = (0)(1) + (-1)(1) + (1)(-1) = -1 - 1 = -2$$

$$s_1 = \text{sign}(-2) = -1$$

New state:

$$s = [-1, 1, -1]$$

Example Setup

- A step-by-step numerical example of the Hopfield Network updating rule using a small system.

✅ **Update Neuron 2**

$$h_2 = (-1)(-1) + (0)(1) + (-1)(-1) = 1 + 1 = 2$$

$$s_2 = \text{sign}(2) = 1$$

State unchanged:

$$s = [-1, 1, -1]$$

Example Setup

- A step-by-step numerical example of the Hopfield Network updating rule using a small system.

✅ Update Neuron 3

$$h_3 = (1)(-1) + (-1)(1) + (0)(-1) = -1 - 1 = -2$$

$$s_3 = \text{sign}(-2) = -1$$

State unchanged:

$$s = [-1, 1, -1]$$

Example Setup

- A step-by-step numerical example of the Hopfield Network updating rule using a small system.

Step 4: Next Iteration

Repeat updates again:

Neuron 1:

$$h_1 = (0)(-1) + (-1)(1) + (1)(-1) = -2 \Rightarrow s_1 = -1$$

Neuron 2:

$$h_2 = (-1)(-1) + (-1)(-1) = 2 \Rightarrow s_2 = 1$$

Neuron 3:

$$h_3 = -1 - 1 = -2 \Rightarrow s_3 = -1$$

Final Stable State

$$s = [-1, 1, -1]$$

Basic Hopfield Net and How It works…

- A Hopfield Network is a model that can reconstruct data after being fed with corrupt versions of the same data.
- We can describe it as a network of nodes, or units or neurons, connected by links. Each unit has one of two states at any point in time, and we are going to assume these states can be +1 or -1. We can list the state of each unit at a given time in a vector V.
- Links represent connections between units and they are symmetric. In other words, the link between node-i and node-j is identical whichever direction you go in the graph.
- In particular, what is identical is the number representing the strength of the connection between each unit. We can list these numbers in a matrix *W*.

Basic Hopfield Net and How It works…

- If we have a network like the one in Fig. 1, we can describe it with:

$$V = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix}, W = \begin{bmatrix} w_{aa} & w_{ab} & w_{ac} & w_{ad} \\ w_{ba} & w_{bb} & w_{bc} & w_{bd} \\ w_{ca} & w_{cb} & w_{cc} & w_{cd} \\ w_{da} & w_{db} & w_{dc} & w_{dd} \end{bmatrix}$$

A state vector and a weight matrix describe the graph at some point in time

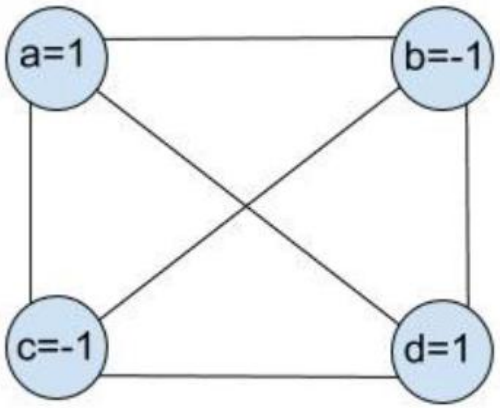


Fig.1 — A Hopfield Network

Basic Hopfield Net and How It works…

- **How do we build *W*?**
  - We do it by first noticing that there is no connection between a node and itself, so we make this explicit by zeroing *Waa*, *Wbb*, *Wcc*, *Wdd*.
  - We also know that links (or weights) are symmetric, for instance, *Wab* = *Wba*. So, ***W* is zero-diagonal and symmetric**.
  - The rest of the elements are filled in the way that: $Wab = Va \cdot Vb$. The result is matrix *W*.
  - This turns out to be a smart choice for the connection weights, as it follows the **Hebbian Rule**.

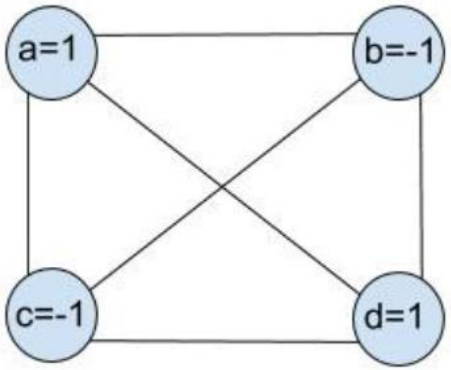


Fig.1 — A Hopfield Network

Basic Hopfield Net and How It works…

- **How do we build *W*?**
- By zeroing *Waa, Wbb, Wcc, Wdd* and *Wab = Va · Vb*.

$$W = \begin{bmatrix} 0 & -1 & -1 & +1 \\ -1 & 0 & +1 & -1 \\ -1 & +1 & 0 & -1 \\ +1 & -1 & -1 & 0 \end{bmatrix}$$

The weight matrix fully describes the network connections

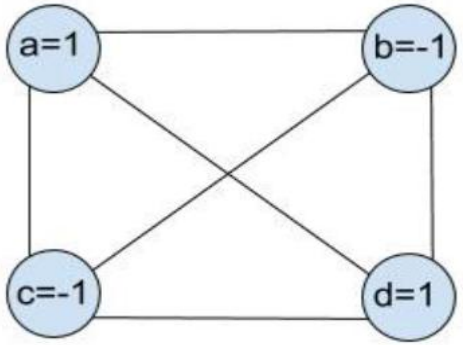


Fig.1 — A Hopfield Network

Basic Hopfield Net and How It works…

- Now we built up the weight matrix we need to define a rule to determine the states of each node.
- Since we have binary states we can use this law:

$$V_i = f(x_i)$$

$$V_i = +1 \qquad if \qquad x_i = \sum_{j \neq i} w_{ij} \cdot V_j > 0$$

$$V_i = -1 \qquad if \qquad x_i = \sum_{j \neq i} w_{ij} \cdot V_j < 0$$

State update rule

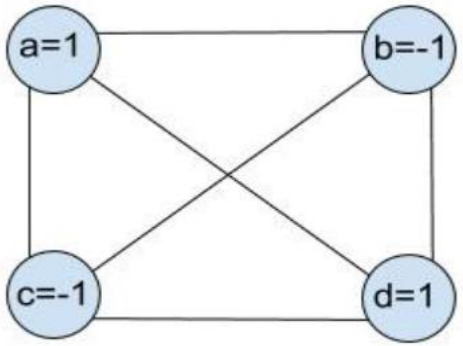
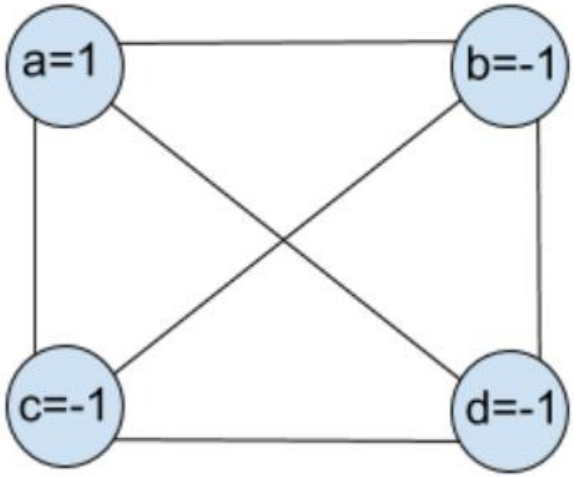


Fig.1 — A Hopfield Network

Basic Hopfield Net and How It works…

- Now we can get an intuition of how Hopfield Networks actually work.
- Say we have a corrupt version of *V*, and

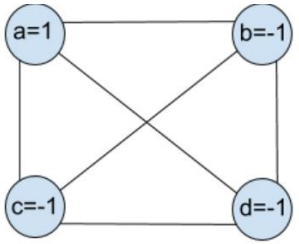
  we will call it *V'* = [1 -1 -1 -1] , such that the last bit was reversed.
- We can initialize the network states with *V'*.



Network initialized with corrupted states: d is now -1

Basic Hopfield Net and How It works…

- We can proceed updating the network states with the rules we established before. For Va we will have:



$$Va = f(wab \cdot Vb + wac \cdot Vc + wad \cdot Vd) = +1$$

- Note that $wab \cdot Vb$ and $wac \cdot Vc$ have both positive contributions to x, despite Vb and Vc being -1, pushing Va to be +1, while the corrupted bit contributes in the opposite direction.
- In other words, by building W as before, we force nodes to settle in opposite states when they are supposed to be of opposite sign, and to settle to the same state when they are supposed to be equal.
- Opposite states repel each other, while same states attract instead. This is one formulation of the Hebbian rule.

> Hebbian Rule: *Neurons that fire together, wire together*

Basic Hopfield Net and How It works

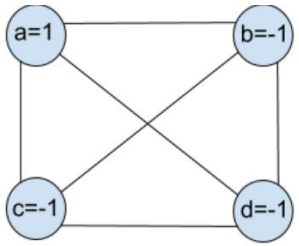
- **Similarly:**

$$Vb = f(wba \cdot Va + wbc \cdot Vc + wbd \cdot Vd) = -1$$

$$Vc = f(wca \cdot Va + wcb \cdot Vb + wcd \cdot Vd) = -1$$

Now the corrupted bit:

$$Vd = f(wda \cdot Va + wdc \cdot Vc + wdb \cdot Vb) = +1$$



- And ***Va* = +1** attracts node *d* to be positive by having **wda · Va>0**, while ***Vc=Vd*= -1** repel node ***d*** by giving an overall positive contribution ***wdc · Vc + wdb · Vd* >0**.
- Nodes that were originally the same, are driven to be the same, nodes that were originally of opposite sign repel each other to be opposite.
- **The original *V* is magically recovered!**

HOPFIELD NETWORKS

To be continued…