Tutorial 8

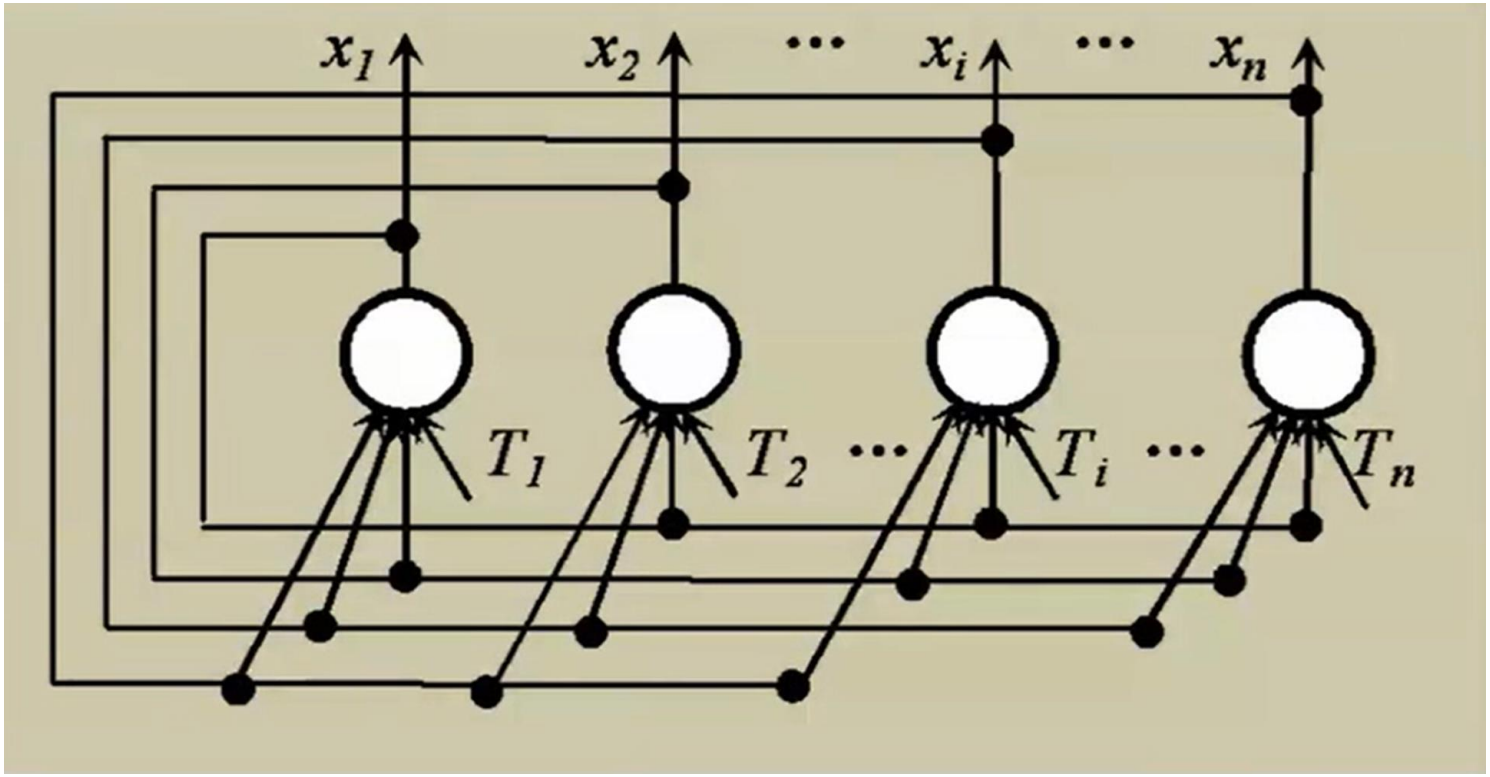
The Applications of Discrete Hopfield Neural Network (DHN)

Name: Fangyu ZHANG
E-mail: fzhang282-c@my.cityu.edu.hk

Department of Computer Science
City University of Hong Kong

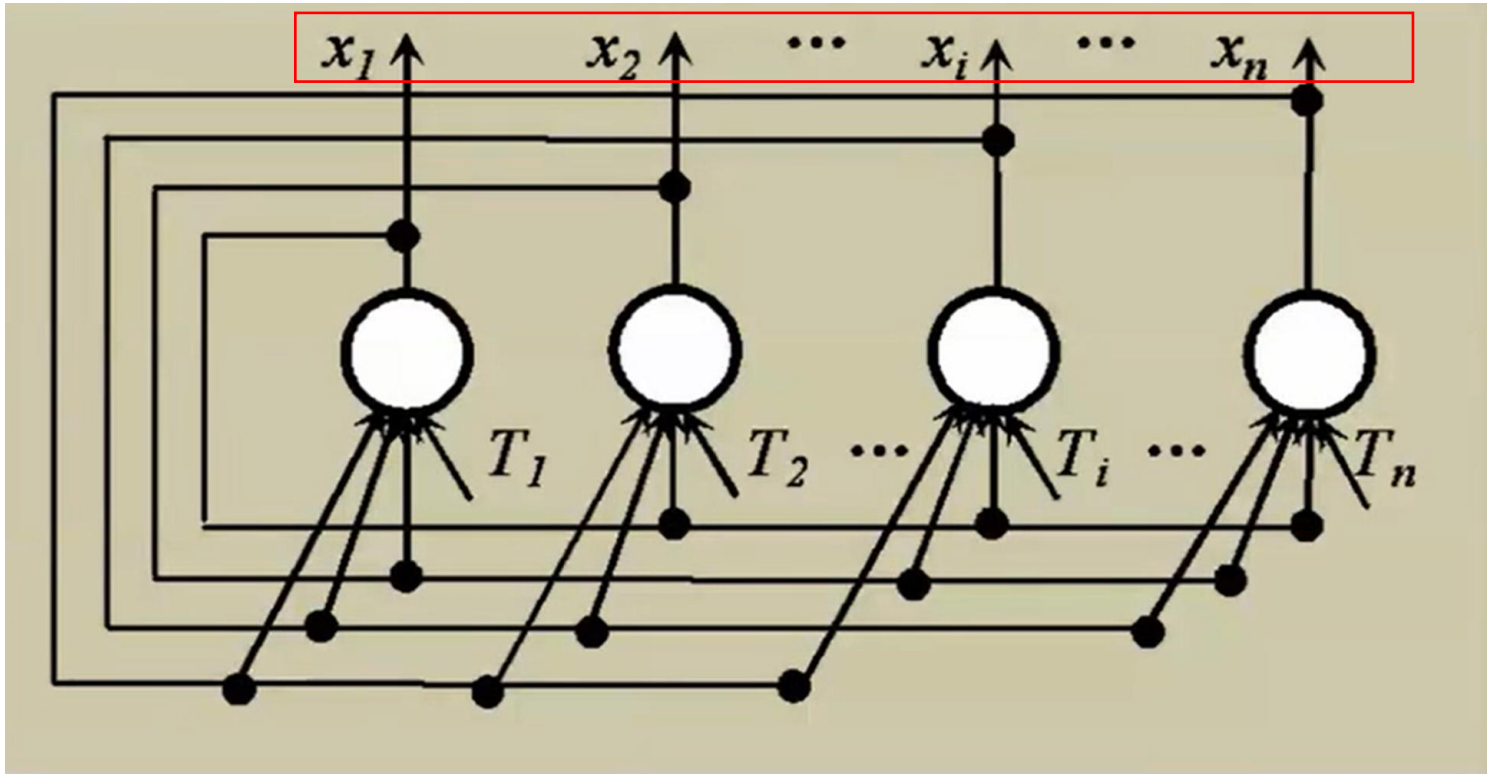
1. DHN

Structure of DHN



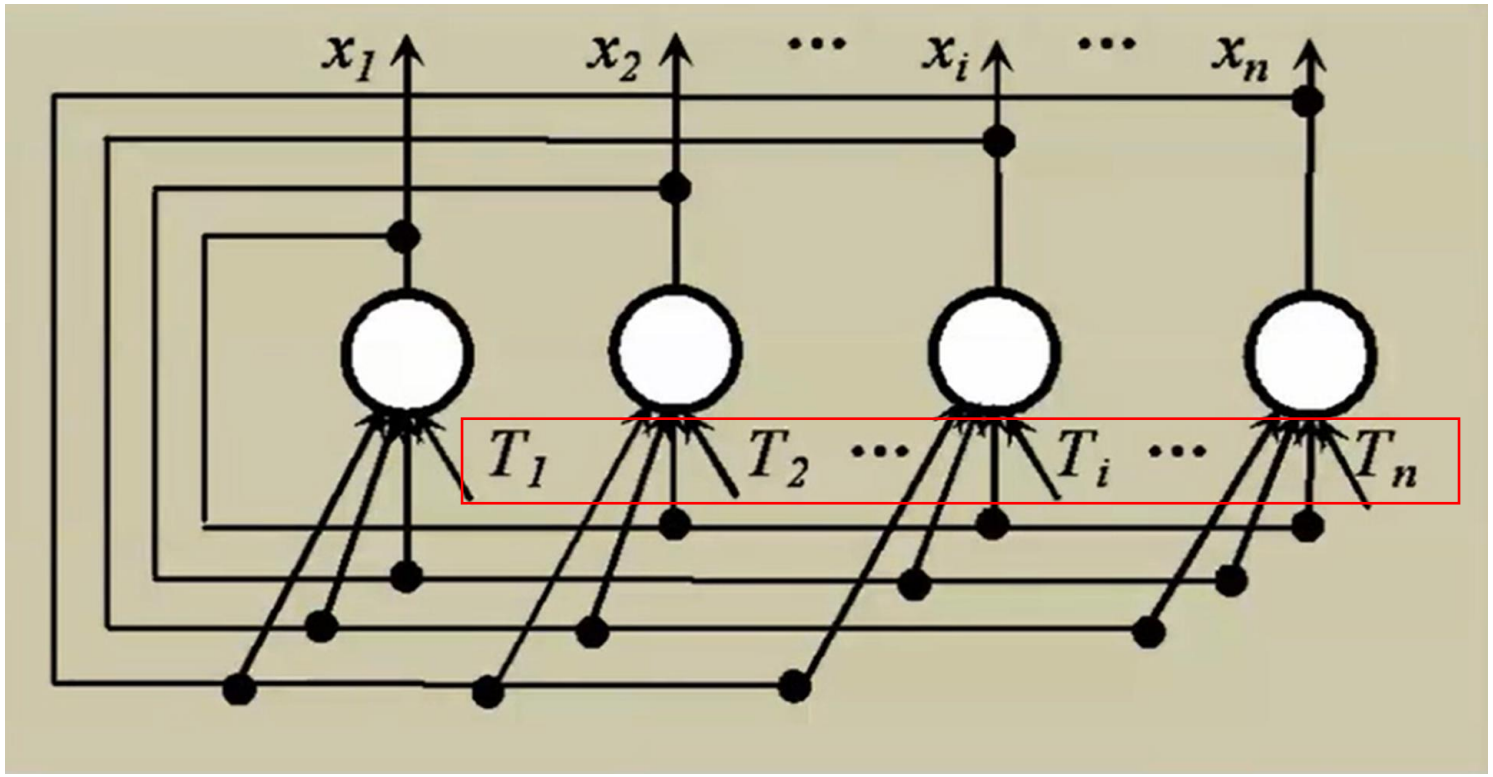
Structure of DHN

In a DHN, each neuron has the same functionality, and its output is called its **state**, denoted as x_j



1. DHN

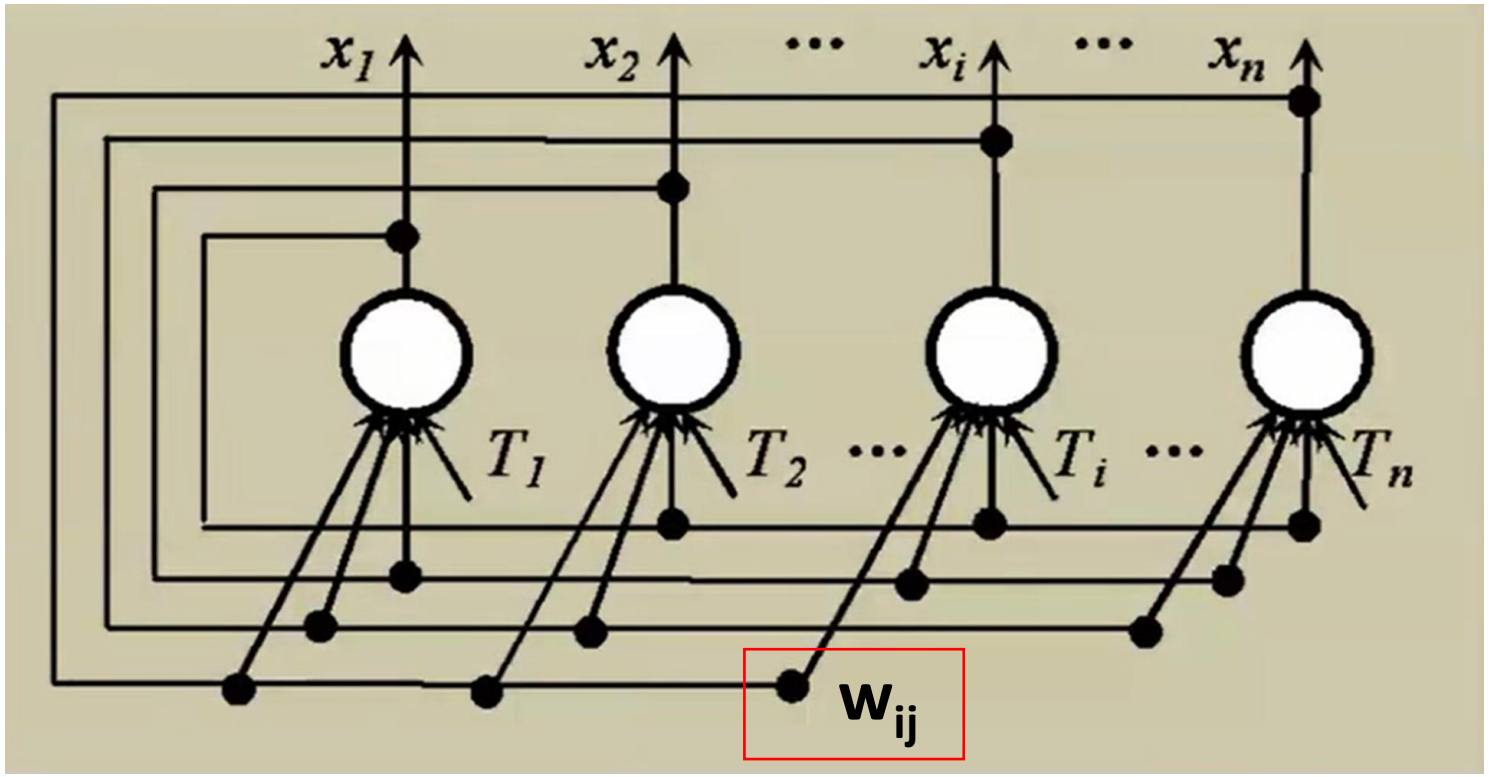
Structure of DHN



T_j is a threshold for neuron j.

1. DHN

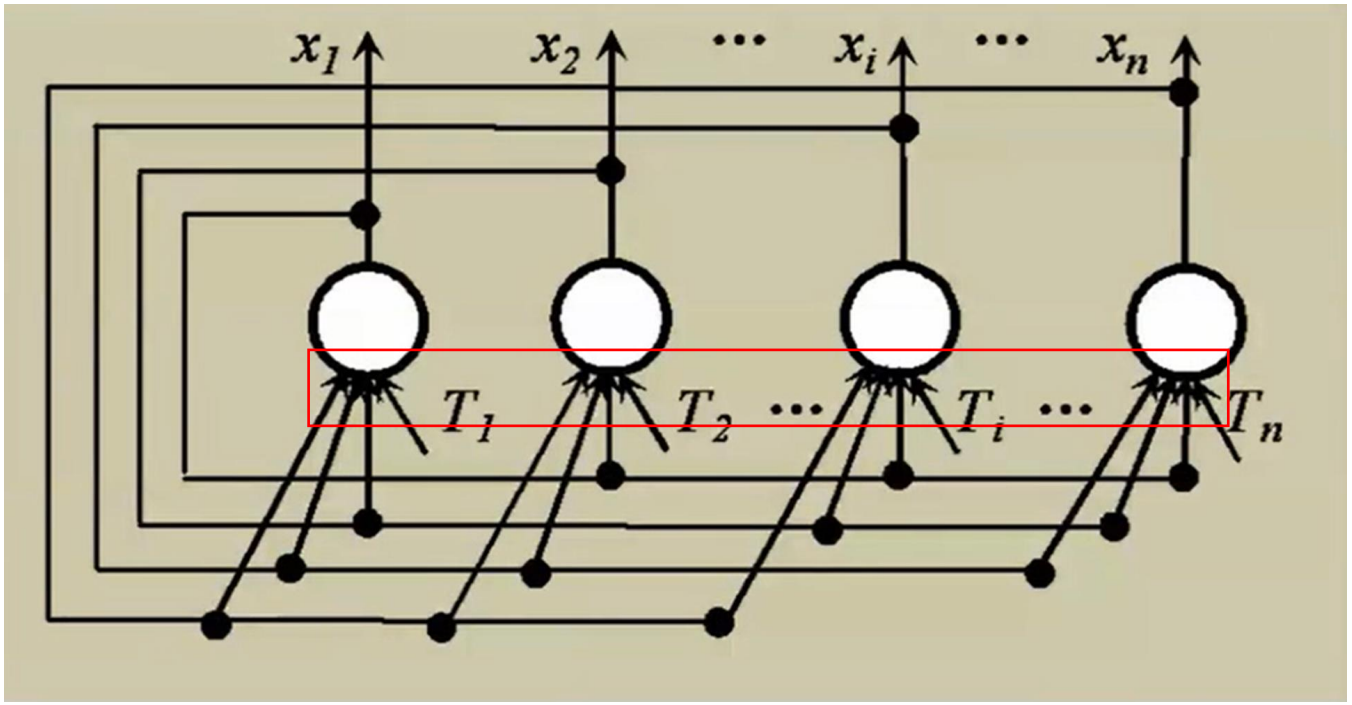
Structure of DHN



w_{ij} represents the **weight connection** between neurons i and j

1. DHN

Structure of DHN



- The **net input** represents the **total input each neuron receives,** computed as:

$$net_j = \sum_{i=1}^{n} (w_{ij} x_i) - T_j$$

- In a **DHN**, typically:
  - $w_{ii} = 0$ (no self-connections),
  - $w_{ij} = w_{ji}$ (symmetric weights).

Structure of DHN

The **state vector X** represents the collection of all neuron states in the network:

$$X = [x_1, x_2, ..., x_n]^T$$

The initial input to the feedback network is the **initial state** of the network, represented as:

$$X(0) = [x_1(0), x_2(0), ..., x_n(0)]^T$$

Under external input stimulation, the feedback network undergoes a **dynamic evolution process**, following the update rule:

$$x_j = f(net_j) \quad \text{where } j = 1, 2, ..., n.$$

Transition Function of DHN

The **net input** to neuron j is calculated as:

$$net_j = \sum_{i=1}^{n} (w_{ij} x_i) - T_j$$

where:

- w_{ij} represents the **weight connection** between neurons i and j,
- T_j is the **threshold** for neuron j.

The state update function commonly used in DHN is the sign function (sgn):

$$x_j = sgn(net_j) = \begin{cases} 1, & \text{if } net_j \geq 0 \\ -1, & \text{if } net_j < 0 \end{cases}$$

Stability and Attractors of the Network

- **Network Stability:**

  If the network starts from an initial state X(0) and, after a finite number of updates, its state no longer changes (i.e., X(t+1)=X(t)), the network is considered stable.

  A stable network will always converge to a stable state from any initial state.

Attractors and Energy Function

- **Attractors of the Network:**

  When the network reaches a stable state X, this state is called an "**attractor**" of the network.

- **Memory and Retrieval**

  - If memory patterns are stored as different attractors, then when a new input contains partial information of a stored pattern, the network's evolution will retrieve the complete information.
  - This process is known as **"associative memory recall."**

- **Definition of an Attractor:**

  A network state X is considered an attractor if it satisfies:

$$X = f(WX + T)$$

  where W denotes the weight matrix and T denotes a threshold vector.

Asynchronous and Synchronous Working Modes of the Network

- **Asynchronous Mode:**
  In this mode, at each time step, only one neuron j updates its state while all other neurons remain unchanged. Mathematically, this is represented as:

$$x_j(t+1) = \begin{cases} \text{sgn}[net_j(t)], & j = i \\ x_j(t), & j \neq i \end{cases}$$

- **Synchronous Mode:**
  In this mode, all neurons in the network update their states simultaneously. The update rule is:

$$x_j(t+1) = \text{sgn}[net_j(t)], \quad j = 1, 2, \ldots, n$$

1. DHN

- **Theorem 1: Convergence of DHNN with Asynchronous Update**

  For a **DHN,** if the network updates its state using an **asynchronous** method and the weight matrix W is **symmetric**, the network will always converge to an **attractor** from any initial state.

  Proof of Theorem 1:

  The **energy function** of the network is given by:

$$E(t) = -\frac{1}{2}X^T(t)WX(t) + X^T(t)T$$

  Define the change in energy as: $\Delta E(t) = E(t+1) - E(t)$

  Define the change in state as: $\Delta X(t) = X(t+1) - X(t)$

  Substituting and expanding:

$$\Delta E(t) = -\Delta x_j(t)\left[\sum_{i=1}^{n} w_{ij}x_i - T_j\right] - \frac{1}{2}\Delta x_j^2(t)w_{jj}$$

1. DHN

- **Theorem 1: Convergence of DHN with Asynchronous Update**

  For a **DHN,** if the network updates its state using an **asynchronous** method and the weight matrix W is **symmetric**, the network will always converge to an **attractor** from any initial state.

  Proof of Theorem 1:

$$\Delta E(t) = -\Delta x_j(t) \left[\sum_{i=1}^{n} w_{ij} x_i - T_j \right] - \frac{1}{2} \Delta x_j^2(t) w_{jj}$$

Possible Cases for $\Delta E(t)$

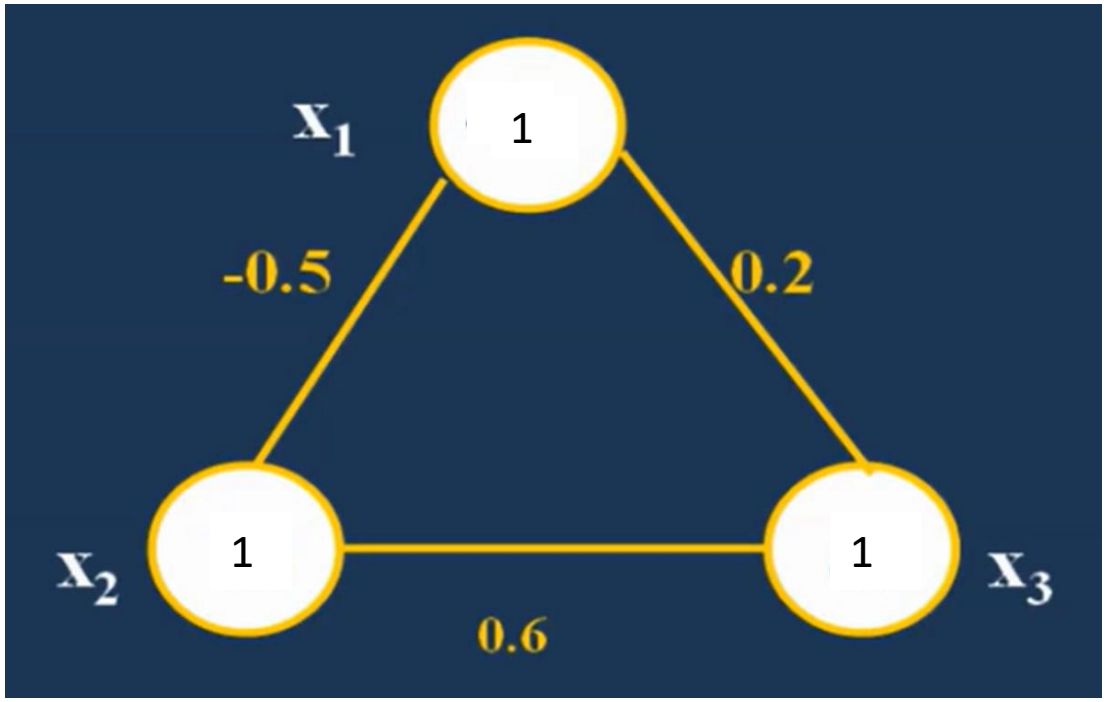
- **Case a**: $x_j(t) = -1$ and $x_j(t+1) = 1$, meaning $\Delta x_j(t) = 2$. If $net_j(t) \geq 0$, then $\Delta E(t) \leq 0$.
- **Case b**: $x_j(t) = 1$ and $x_j(t+1) = -1$, meaning $\Delta x_j(t) = -2$. If $net_j(t) < 0$, then $\Delta E(t) < 0$.
- **Case c**: $x_j(t) = x_j(t+1)$, meaning $\Delta x_j(t) = 0$, so $\Delta E(t) = 0$.

Since the energy function is bounded, the network must eventually reach a **constant energy value**, meaning the system will settle into an **attractor**.

1. DHN

- **Example**

A **DHN** with three nodes. The network is represented as an **undirected weighted graph (since $w_{ij} = w_{ji}$)**, where each node has a threshold value (0.5).



Weight Design for Networks

- To ensure the network converges when working asynchronously, the weight matrix (W) should be symmetric.

Taking a **3-node DHN** as an example:

- The attractors (desired stable states) are defined as: $\mathbf{X^a = (0\ 1\ 0)^T}$ **and** $\mathbf{X^b = (1\ 1\ 1)^T}$
- The weights (W) and thresholds (T) take values within **[-1,1]**
- The weight matrix is symmetric: $\mathbf{w_{ij} = w_{ji}}$
- The goal is to find the weight values and thresholds.

Weight Design for Networks

- To ensure the network converges when working asynchronously, the weight matrix (W) should be symmetric.

Taking a **3-node DHN** as an example:

- The attractors (desired stable states) are defined as: $\mathbf{X^a = (0\ 1\ 0)^T}$ **and** $\mathbf{X^b = (1\ 1\ 1)^T}$
- The weights (W) and thresholds (T) take values within **[-1,1]**
- The weight matrix is symmetric: $\mathbf{w_{ij} = w_{ji}}$
- The goal is to find the weight values and thresholds.

For State $X^a = (0\ 1\ 0)^T$

Each node's net input should satisfy:

- $net_1 = w_{12} \times 1 + w_{13} \times 0 - T_1 = w_{12} - T_1 < 0$ **(1)**
- $net_2 = w_{12} \times 0 + w_{23} \times 0 - T_2 = -T_2 > 0$ **(2)**
- $net_3 = w_{13} \times 0 + w_{23} \times 1 - T_3 = w_{23} - T_3 < 0$ **(3)**

Weight Design for Networks

- To ensure the network converges when working asynchronously, the weight matrix (W) should be symmetric.

Taking a **3-node DHN** as an example:

- The attractors (desired stable states) are defined as: $\mathbf{X^a = (0\ 1\ 0)^T}$ **and** $\mathbf{X^b = (1\ 1\ 1)^T}$
- The weights (W) and thresholds (T) take values within **[-1,1]**
- The weight matrix is symmetric: $\mathbf{w_{ij} = w_{ji}}$
- The goal is to find the weight values and thresholds.

For State $X^b = (1\ 1\ 1)^T$

Each node's net input should satisfy:

- $net_1 = w_{12} \times 1 + w_{13} \times 1 - T_1 > 0$ **(4)**
- $net_2 = w_{12} \times 1 + w_{23} \times 1 - T_2 > 0$ **(5)**
- $net_3 = w_{13} \times 1 + w_{23} \times 1 - T_3 > 0$ **(6)**

Weight Design for Networks

- To ensure the network converges when working asynchronously, the weight matrix (W) should be symmetric.

Taking a **3-node DHN** as an example:

- The attractors (desired stable states) are defined as: $\mathbf{X^a = (0\ 1\ 0)^T}$ **and** $\mathbf{X^b = (1\ 1\ 1)^T}$
- The weights (W) and thresholds (T) take values within **[-1,1]**
- The weight matrix is symmetric: $\mathbf{w_{ij} = w_{ji}}$
- The goal is to find the weight values and thresholds.

Solving for Weights and Thresholds

By solving the six inequalities above, we determine the valid range for each unknown parameter.

- **Choosing** $\mathbf{w_{12} = 0.5}$, from **(1)**, we get $\mathbf{0.5 < T_1 \leq 1}$, hence $\mathbf{T_1 = 0.7}$.
- **From (4):** $0.2 < w_{13} \leq 1$, choosing $\mathbf{w_{13} = 0.4}$.
- **From (2):** $-1 \leq T_2 < 0$, choosing $\mathbf{T_2 = -0.2}$.
- **From (5):** $-0.7 < w_{23} \leq 1$, choosing $\mathbf{w_{23} = 0.1}$.
- **From (6):** $-1 < T_3 \leq 0.5$, choosing $\mathbf{T_3 = 0.4}$.

1. Applications of Discrete Hopfield Networks

Optimization: e.g., solving the Traveling Salesman Problem

Application Scenario

Hopfield networks can be used for combinatorial optimization problems, such as the **Traveling Salesman Problem (TSP)**, which involves finding the shortest route that visits all cities exactly once and returns to the starting point.

- N cities are represented by an $N \times N$ matrix of neurons
- Each row has exactly one 1
- Each column has exactly one 1
- Matrix has exactly N 1's

CITY	ORDER VISITED 1	2	3	4
A	0	1	0	0
B	0	0	1	0
C	0	0	0	1
D	1	0	0	0

$\sigma_{kj} = 1$ if city k is in position j

$\sigma_{kj} = 0$ otherwise

The network iteratively minimizes its energy function to find an approximate optimal route

1. Applications of Discrete Hopfield Networks

Associative Memory

Application Scenario

Hopfield networks can store multiple binary patterns and retrieve the correct one when given a partial or noisy version.

Example

Assume we have three stored patterns:

1. $\mathbf{X_1}$ = **[1, -1, 1, -1]**
2. $\mathbf{X_2}$ = **[-1, 1, -1, 1]**
3. $\mathbf{X_3}$ = **[1, 1, -1, -1]**

If we input a noisy pattern, such as:
Y = [1, -1, -1, -1]

The Hopfield network will iteratively update and converge to the closest stored pattern, e.g., $\mathbf{X_1}$ = **[1, -1, 1, -1]**, demonstrating its associative memory function.

1. Applications of Discrete Hopfield Networks

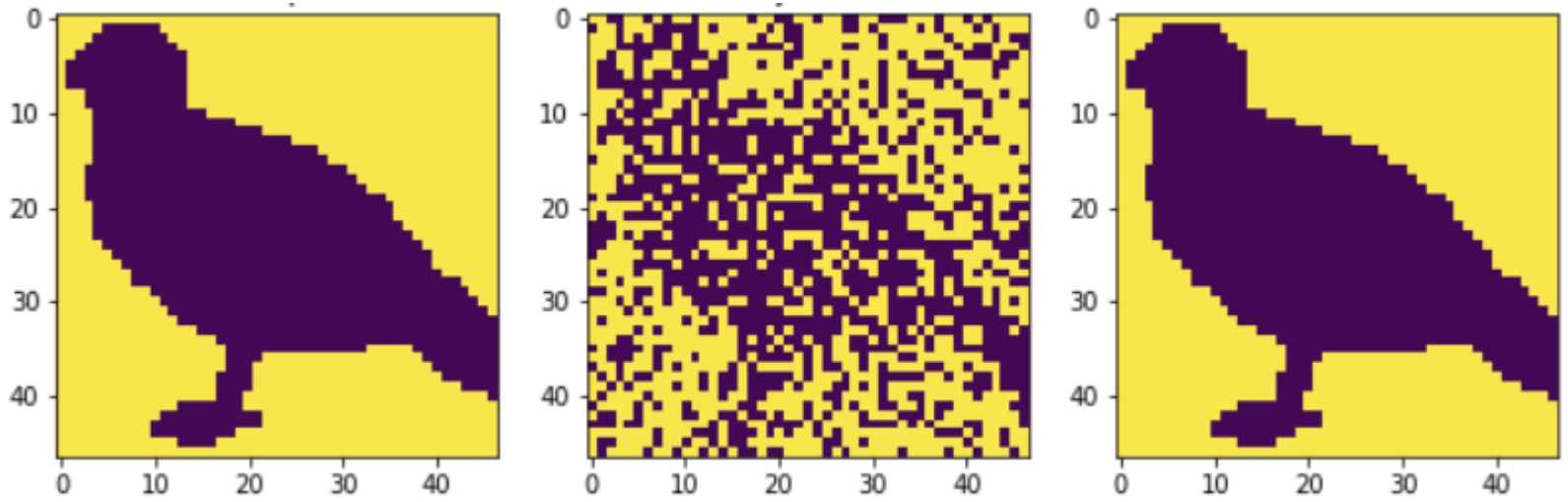
Image Denoising and Restoration

Application Scenario

Hopfield networks can be used to restore corrupted or noisy images by leveraging stored ideal versions of images.

Example

If an image has missing pixels or noise, a Hopfield network trained on the original image can reconstruct the missing parts by converging to the closest stored pattern. This method is useful in **medical imaging, satellite image reconstruction, and old photograph restoration**.



1. Applications of Discrete Hopfield Networks

Speech Recognition

Application Scenario

Hopfield networks can be used in **speech pattern recognition**, filtering noisy speech signals and recognizing key phonemes.

Example

If a stored pattern represents a clean speech signal, a noisy input can be corrected to match the stored version, improving clarity. This is useful in **voice assistants, hearing aids, and automatic transcription services**.

1. Applications of Discrete Hopfield Networks

Content Addressable Memory

Application Scenario

Hopfield networks work as a special type of memory that retrieves complete information based on partial input, similar to human memory recall.

Example

After storing multiple patterns, when given incomplete information, the network uses an energy minimization process to retrieve the most similar stored pattern. This is useful in applications like database searches and information retrieval.

1. Applications of Discrete Hopfield Networks

Fault Detection in Systems

Application Scenario

Hopfield networks can be applied in **industrial systems, electrical grids, and mechanical systems** to detect anomalies or faults.

Example

A system's normal operational states can be stored as patterns. If an input deviates significantly from these stored states, it indicates a fault. This is useful in **predictive maintenance, sensor-based monitoring, and automated diagnostics**.

1. Applications of Discrete Hopfield Networks

Robotics & Control Systems

Application Scenario

Hopfield networks can be used in **robotic path planning, control systems, and autonomous navigation**.

Example

A robot navigating through a maze can use a Hopfield network to **memorize safe paths** and correct its trajectory when encountering obstacles.