Experiment 1: Implementing and Solving Combinatorial Problems Using Hopefield Network.

Tejas Pakhle, Rajat Kumar Thakur, Tanay Patel, Abhi Tundya

I. Week 6: Hopefield Theory and Network

The Hopfield network, introduced by John Hopfield in 1982, is a type of recurrent artificial neural network designed for associative memory and optimization tasks. It operates by storing patterns in a high-dimensional space and recalling them through an energy minimization process. The network consists of binary neurons that interact with each other through symmetric weights.

Hopfield networks are particularly useful in solving problems where an optimal solution needs to be found among a set of possible solutions, such as combinatorial optimization problems, pattern recognition, and constraint satisfaction. The network converges to a stable state, representing the solution, by minimizing an energy function. This makes it applicable to various NP-hard problems like the traveling salesman problem (TSP), N-Queens, and more.

A. Problem Definitions

1) **10x10 Binary Associative Memory:**
   - Implement a 10x10 binary associative memory using the Hopfield network.
   - Evaluate the network's capacity to store and retrieve distinct patterns, and its error-correcting ability.
2) **Error-Correcting Capability:**
   - Analyze the network's ability to correct corrupted or noisy patterns.
3) **Eight-Rook Problem:**
   - Solve the Eight-Rook problem on an 8x8 chessboard using the Hopfield network.
   - Define an energy function and discuss the choice of weights.
4) **Traveling Salesman Problem (TSP):**
   - Solve the TSP for 10 cities using the Hopfield network.
   - Derive the energy function and calculate the number of weights required.

Understanding Terminologies

Hopfield Architecture

- **Neurons:** Binary units representing network nodes, with states of 1 (active) or 0 (inactive). Neurons update based on inputs from other neurons.
- **Weights:** Symmetric connections ($w_{ij} = w_{ji}$) that determine interaction strength between neurons. Proper weights guide the network to stable states.
- **Capacity:** The network can store approximately $0.15 \times N$ distinct patterns, where N is the number of neurons.

Energy Function and Landscape

- **Energy Function:** A scalar value associated with the network's state, minimized during updates. Lower energy corresponds to more stable states.
- **Energy Landscape:** A representation of all possible states and their energy levels. Stable states are local minima, with the global minimum representing the optimal solution.

Related:

- **Convergence:** The network evolves to minimize energy, ensuring convergence to a stable state.
- **Associative Memory:** Enables pattern storage and recall, even with noisy or partial inputs.
- **Error Correction:** Corrects small errors in corrupted inputs by moving toward the closest stored pattern.

Energy Function in Optimization Problems

- For problems like the TSP and Eight-Rook Problem, the energy function incorporates constraints such as visiting each city once or avoiding conflicts, while optimizing the solution.

II. Implementing Addressable Memory

Algorithm 1 Training Algorithm for Hopfield Network

Require: Patterns $\{p_k\}$, number of neurons N
Ensure: Weight matrix W
Initialize $W \leftarrow \mathbf{0}_{N \times N}$
for each pattern p_k **do**
    $W \leftarrow W + p_k \otimes p_k$ ▷ Outer product update
end for
for $i = 1$ to N **do**
    $W_{ii} \leftarrow 0$ ▷ No self-connections
end for
return W

Statistical Physics Approach (Storkey Heuristic):

Algorithm 2 Recall Algorithm for Hopfield Network

Require: Initial pattern p, weight matrix W, number of steps T

Ensure: Final recalled pattern p

for $t = 1$ to T **do** ▷ Iterate for T steps
  for $i = 1$ to N **do**
    activation$_i \leftarrow \sum_{j=1}^{N} W_{ij} \cdot p_j$ ▷ Compute activation
    $p_i \leftarrow \begin{cases} +1 & \text{if activation}_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$
  end for
end for
return p

Figure 1: Working

- Capacity is approximately $0.138 \times N$
- For $N = 100$ neurons, estimated capacity is approximately 14 distinct patterns

Classic Gardner Bound:

- Capacity is approximately $0.15 \times N$
- For $N = 100$ neurons, capacity is approximately 15 distinct patterns

Computational Learning Theory Estimate:

- Capacity is roughly between $0.12 \times N$ and $0.14 \times N$
- For $N = 100$ neurons, capacity is around 12-14 patterns

The error-correcting capability of the network depends on the Hamming distance between stored patterns, d_H, and can be expressed as:

$$d_H < \frac{N}{2} \tag{1}$$

where d_H is the maximum Hamming distance the network can tolerate for error correction. For $N = 100$, the maximum Hamming distance is:

$$d_H < \frac{100}{2} = 50 \tag{2}$$

it can correct up to 49 errors (fliped bits)

8 Rook Problem

The Eight-Rook problem involves placing eight rooks on an 8x8 chessboard such that no two rooks threaten each other. A rook threatens another if it is in the same row or column. This is a classic combinatorial problem that can be solved by encoding the constraints into the network's energy function.

Energy Function

The energy function used in the Hopfield network ensures that the system converges to a valid configuration where the constraints of the Eight-Rook problem are satisfied. The energy function is defined as:

$$E = -\frac{1}{2}\mathbf{s}^T W \mathbf{s}$$

where:

- $\mathbf{s}$ is the state vector representing the chessboard (flattened into a single array).
- W is the weight matrix encoding the constraints that penalize two rooks being in the same row or column.

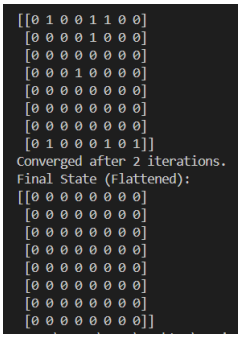

Figure 2: 8 Rook Problem

Algorithm 3 Hopfield Network for Solving the 8 Rooks Problem

Require: Chessboard size N, number of iterations $num\_iterations$

Ensure: Final configuration of rooks on the chessboard

Initialize the board with random rook placements: $p_0 \in \{0,1\}^{N^2}$ ▷ Random initial state
Initialize weight matrix W based on row and column constraints ▷ No two rooks in the same row/column
for $t = 1$ to $num\_iterations$ **do** ▷ Iterate for $num\_iterations$ steps
  for $i = 1$ to N^2 **do** ▷ For each neuron (position) on the chessboard
    activation$_i \leftarrow \sum_{j=1}^{N^2} W_{ij} \cdot p_j$ ▷ Compute the activation for the i-th position
    $p_i \leftarrow \begin{cases} +1 & \text{if activation}_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$ ▷ Update state based on activation
  end for
  if $p = p_{\text{new}}$ **then** ▷ Check for convergence
    Return p ▷ Solution found, return the final configuration
    Break ▷ Exit the loop if the network has converged
  end if
end for
return p ▷ Return the final state after all iterations

REASONING FOR WEIGHT INITIALIZATION IN THE 8 ROOKS PROBLEM

The weight matrix for the Hopfield network is initialized to encode the constraints of the 8 Rooks problem, where no two rooks can share the same row or column.

Weight Initialization

The weight matrix W is set as follows:

$$W_{ij,kl} = \begin{cases} -1 & \text{if } i = k \text{ and } j \neq l \text{ (same row, diff. column)} \\ -1 & \text{if } j = l \text{ and } i \neq k \text{ (same column, diff. row)} \\ 0 & \text{otherwise} \end{cases}$$

A. Reasoning Behind Negative Weights

The negative weights enforce the constraint that no two rooks can occupy the same row or column. When two neurons (representing positions on the chessboard) are in the same row or column, their negative weights cause them to "repel" each other, discouraging configurations where multiple rooks are placed in the same row or column.

Zero Weights on Diagonal

Diagonal weights are set to zero, as there is no need for a neuron to interact with itself. This ensures that the network's dynamics are only influenced by interactions between different positions.

TRAVELLING SALESMAN PROBLEM

The Travelling Salesman Problem (TSP) is a classic optimization problem where a salesman must visit a set of cities exactly once, starting and ending at the same city, and minimize the total travel distance. Given its NP-hard nature, the TSP has important applications in logistics, route planning, and many other optimization problems.

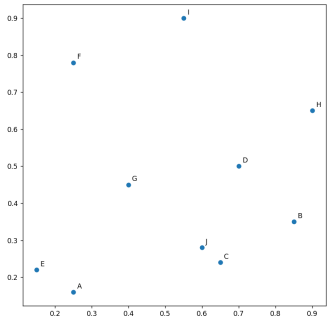

Figure 3: 10 cities

B. Energy Function for TSP in Hopfield Network

The energy function used in the provided code for solving the TSP within a Hopfield network is defined as:

$$E = \frac{A}{2} \sum_{x} \sum_{i \neq j} \mathbf{y}_{x,i} \cdot \mathbf{y}_{x,j} + \frac{B}{2} \sum_{i} \sum_{x \neq y} \mathbf{y}_{x,i} \cdot \mathbf{y}_{y,i}$$

$$+ \frac{C}{2} \left(\sum_{x} \sum_{i} \mathbf{y}_{x,i} - (\text{number\_of\_objects} + \sigma) \right)^2$$

$$+ \frac{D}{2} \sum_{x \neq y} \sum_{i} \mathbf{y}_{x,i} \cdot \mathbf{y}_{y,i+1}$$

Where:

- $\mathbf{y}_{x,i}$ represents the binary state of the system indicating whether city i is visited at position x.
- A, B, C, D are constants controlling the weights for row/-column inhibition and path constraints.
- σ is a constant related to the number of objects.

This energy function models the objective of minimizing the total distance traveled while adhering to the constraints of visiting each city exactly once and returning to the origin.

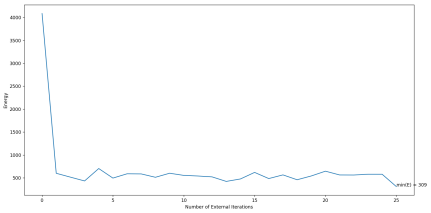

Figure 4: Energy-Iteration Plot

ALGORITHM FOR SOLVING THE TSP

Algorithm 4 TSP Solution using Modified Hopfield Network

Require: Cities' coordinates, distance matrix D, constants A, B, C, D, σ, number of iterations
Ensure: Optimal path and total distance

```
Initialize Hopfield network with number_of_cities, A, B, C, D, σ
Initialize distances matrix D between cities
Define energy function incorporating path constraints and city locations
for each iteration do
    Compute input potentials for each city and update output potentials
    Update output potentials based on the energy function
    Compute energy and check for convergence
    if Energy has stabilized or reached a minimum then
        Stop training
    end if
end for
Extract the optimal path from the output potentials
Calculate the total distance based on the optimal path
return Optimal path and total distance
```

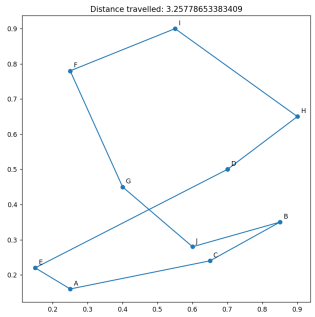

Figure 5: Enter Caption

III. Calculation of Weights in Hopfield Network for TSP

The total number of weights in the Hopfield network is calculated as follows:

$$\text{Inhibition Weights in the Same Row} = \frac{n^2(n-1)}{2} = 900$$

$$\text{Inhibition Weights in the Same Column} = \frac{n^2(n-1)}{2} = 900$$

$$\text{Path Constraint Weights} = n \times (2n-2) = 1620$$

$$\text{Total Weights} = \frac{n^2(n-1)}{2} + \frac{n^2(n-1)}{2} + n \times (2n-2) = 3420$$