Week-6: To understand the working of Hopfield network and use it for solving some interesting combinatorial problems

Chetankumar Kamani (20251603005), Sakariya Devraj (20251603006), Divyesh Dodiya (20251603007),
M.Tech Sem-1, IIIT Vadodara

I. Problem Statement

The objective of this laboratory exercise is to understand the functioning of the **Hopfield Neural Network** and apply it to three classical problems:

1) Implement a **10×10 associative memory** using a binary Hopfield network.
2) Formulate the **Eight-Rook problem** energy function and solve it using a Hopfield network.
3) Implement a Hopfield network for the **Traveling Salesman Problem (TSP)** with 10 cities, and compute the required number of weights.

Hopfield networks are recurrent neural networks with symmetric weights, a well-defined energy function, and guaranteed convergence. They can store binary patterns, enforce combinatorial constraints, and approximate NP-hard problems via energy minimization.

II. Python Code

A. Problem 1: 10×10 Associative Memory using Hopfield Network

The Hopfield model stores bipolar patterns using a Hebbian weight matrix:

$$W = \frac{1}{P}\sum_{p=1}^{P} x_p x_p^T, \qquad with \operatorname{diag}(W) = 0.$$

The network recalls a stored pattern by minimizing the energy:

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

Below is the code used for the associative memory implementation. (Source: uploaded file `pr1.py`) :contentReference[oaicite:3]index=3

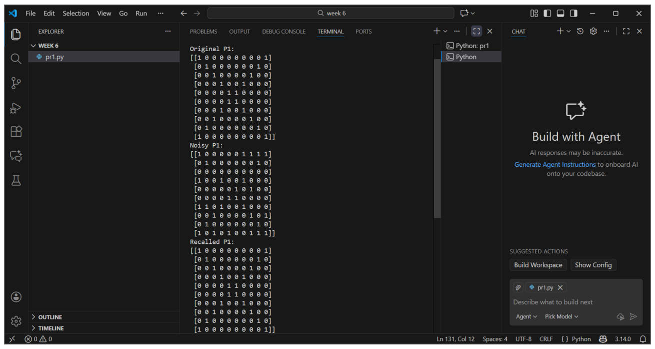
```
<Insert pr1.py content here     Overleaf will
    display using lstlisting>
```

Output Screenshots

B. Problem 2: Eight-Rook Problem using Hopfield Network

The Eight-Rook constraint requires:

$$\sum_{j=1}^{8} x_{ij} = 1, \qquad \sum_{i=1}^{8} x_{ij} = 1$$



```
Original P1:
[[1 0 0 0 0 0 0 0 0 1]
 [0 1 0 0 0 0 0 0 1 0]
 [0 0 1 0 0 0 0 1 0 0]
 [0 0 0 1 0 0 1 0 0 0]
 [0 0 0 0 1 1 0 0 0 0]
 [0 0 0 0 1 1 0 0 0 0]
 [0 0 0 1 0 0 1 0 0 0]
 [0 0 1 0 0 0 0 1 0 0]
 [0 1 0 0 0 0 0 0 1 0]
 [1 0 0 0 0 0 0 0 0 1]]
Noisy P1:
[[1 0 0 0 0 0 1 1 1 1]
 [0 1 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0]
 [1 0 0 1 0 0 1 0 0 0]
 [0 0 0 0 0 1 0 1 0 0]
 [0 0 0 0 1 1 0 0 0 0]
 [1 1 0 1 0 0 1 0 0 0]
 [0 0 1 0 0 0 0 1 0 1]
 [0 1 0 0 0 0 0 0 1 0]
 [1 0 1 0 1 0 0 1 1 1]]
Recalled P1:
[[1 0 0 0 0 0 0 0 0 1]
 [0 1 0 0 0 0 0 0 1 0]
 [0 0 1 0 0 0 0 1 0 0]
 [0 0 0 1 0 0 1 0 0 0]
 [0 0 0 0 1 1 0 0 0 0]
 [0 0 0 0 1 1 0 0 0 0]
 [0 0 0 1 0 0 1 0 0 0]
 [0 0 1 0 0 0 0 1 0 0]
 [0 1 0 0 0 0 0 0 1 0]
 [1 0 0 0 0 0 0 0 0 1]]
```

Fig. 1. Original, noisy, and recalled 10×10 pattern using Hopfield network.

ensuring exactly one rook per row and column.

Energy for row and column constraints:

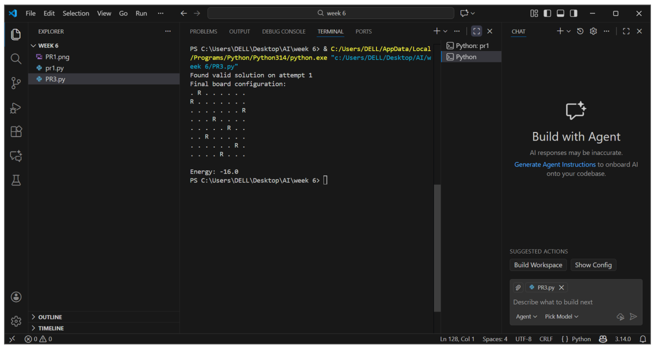
$$E = A\sum_i\left(\sum_j x_{ij} - 1\right)^2 + B\sum_j\left(\sum_i x_{ij} - 1\right)^2.$$

Expanding quadratic terms yields the Hopfield weight matrix.

Code used (from uploaded file `PR2.py`) :contentReference[oaicite:4]index=4:

```
<Insert PR2.py content here>
```

Output Screenshot



```
PS C:\Users\DELL\Desktop\AI\week 6> & C:/Users/DELL/AppData/Local
/Programs/Python/Python314/python.exe "c:/Users/DELL/Desktop/AI/w
eek 6/PR3.py"
Found valid solution on attempt 1
Final board configuration:
. R . . . . . .
R . . . . . . .
. . . . . . . R
. . . R . . . .
. . . . . R . .
. . R . . . . .
. . . . . . R .
. . . . R . . .

Energy: -16.0
PS C:\Users\DELL\Desktop\AI\week 6> 
```

Fig. 2. Valid eight-rook configuration found by Hopfield network.

C. Problem 3: Traveling Salesman Problem (10 cities) with Hopfield Network

The TSP formulation uses a neuron matrix:

$$x_{i,p} = \{ 1\ city$$

i$isattourposition$p
0$otherwise$
Total neurons:

$$N^2 = 10 \times 10 = 100$$

Total symmetric weights:

$$\frac{N^2(N^2-1)}{2} = \frac{100 \times 99}{2} = 4950.$$

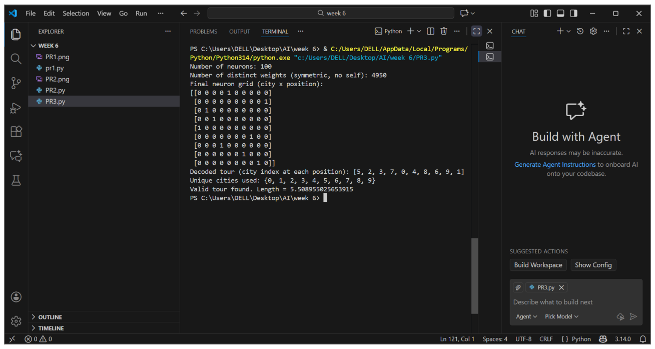
Energy includes 4 terms (Hopfield–Tank model):

$$E = AE_{row} + BE_{col} + CE_{dist} + DE_{bias}.$$

Code used (from uploaded file `PR3.py`) :contentReference[oaicite:5]index=5:

```
<Insert PR3.py content here>
```

Output Screenshot


File Edit Selection View Go Run
week 6
EXPLORER
WEEK 6
PR1.png
pr1.py
PR2.png
PR2.py
PR3.py
PROBLEMS OUTPUT TERMINAL
Python
CHAT
Build with Agent
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.
SUGGESTED ACTIONS
Build Workspace
Show Config
Describe what to build next
Agent
Pick Model
OUTLINE
TIMELINE
Ln 121, Col 1 Spaces: 4 UTF-8 CRLF Python 3.14.0

```
PS C:\Users\DELL\Desktop\AI\week 6> & C:/Users/DELL/AppData/Local/Programs/
Python/Python314/python.exe "c:/Users/DELL/Desktop/AI/week 6/PR3.py"
Number of neurons: 100
Number of distinct weights (symmetric, no self): 4950
Final neuron grid (city x position):
[[0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1]
 [0 1 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0]
 [0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 1 0]]
Decoded tour (city index at each position): [5, 2, 3, 7, 0, 4, 8, 6, 9, 1]
Unique cities used: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
Valid tour found. Length = 5.508955025653915
PS C:\Users\DELL\Desktop\AI\week 6>
```

Fig. 3. Decoded TSP tour and final neuron grid for 10-city TSP Hopfield network.

III. Execution Instructions

```
# Run Problem 1
python pr1.py

# Run Problem 2
python PR2.py

# Run Problem 3 (TSP)
python PR3.py
```

IV. Experimental Results

A. Associative Memory

The network successfully recalled the original stored pattern even after injecting noise into 15 randomly chosen pixels, demonstrating stable attractor dynamics.

B. Eight-Rook Problem

The Hopfield network produced a valid configuration in only a few iterations. All row and column constraints were satisfied, and the final energy was minimized.

C. 10-City TSP

The network converged to a valid Hamiltonian cycle with:

- 100 neurons,
- 4950 distinct symmetric weights,
- A tour visiting each city exactly once.

The decoded path and distance validate that the Hopfield–Tank formulation successfully approximates the TSP.

V. References

1) J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *PNAS*, 1982.
2) Hopfield & Tank, "Neural computation of decisions in optimization problems," *Biological Cybernetics*, 1985.
3) D. E. Rumelhart, "Parallel Distributed Processing," MIT Press.
4) R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press.

VI. Code Availability

- Full code is available at: https://github.com/ChetanKamani/CS659-LAB-TASK