RISHI RAJ SHARMA
M.Sc CS (2402519)
CSM803

Experiment-08: Hopfield Networks

Design a Hopfield network to store the fundamental memory patterns: (1 1 0 0 0), (0 1 1 1 0), and (0 0 0 1 1), using bipolar outer product encoding. Write a MATLAB program that can simulate the network in both synchronous as well as asynchronous mode. Assume bipolar decoding. Answer the following questions:

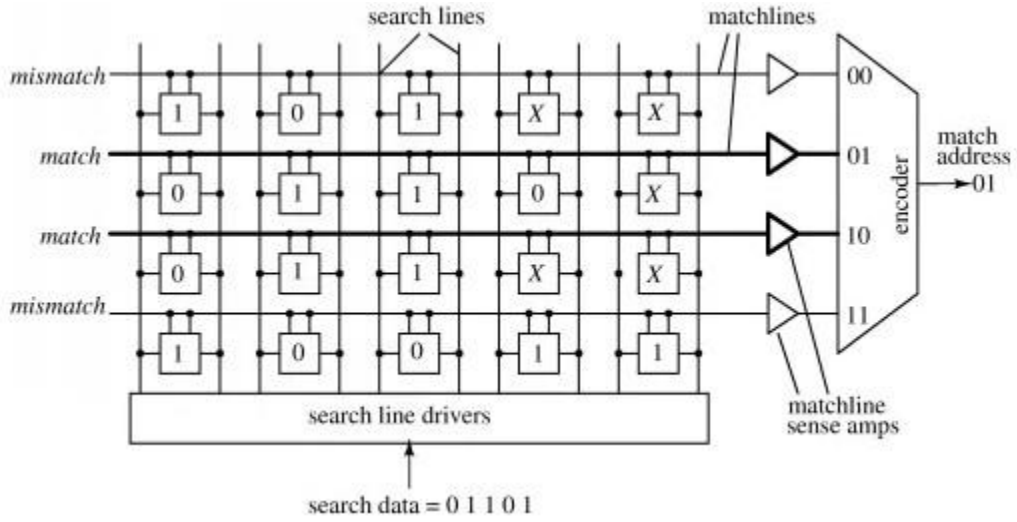
a) **What are the Lyapunov energies of each of the encoded vectors?**

b) **Present each of the 32 possible binary vectors (in dimension 5) to the system. What stable states are reached for synchronous update and also for asynchronous update?**

c) **How does the recall behavior change if the diagonal elements are not zeroed out? In each case, determine the number of limit cycles and fixed-point attractors.**

d) **Finally, if binary decoding were employed, how would the recall behavior change.**

Answers:



Content Addressable Memory

Conversion Binary to Bipolar: Implementing a function to convert binary vectors to bipolar vectors.
Computing the Weight Matrix: Using the Bipolar outer product to computing the weight matrix and zero the diagonal.
Initializing the Signal Vector: Converting the probe vector to bipolar (Encoding) form.
Iterate to Recall the Memory: Network iteratively update the signal vector until it converges.
Final output: The recalled bipolar vector is converted back to binary for final output.

Weight matrix generated according to this equation:

$$W = \sum_{K=1}^{Q} X_k X_k^T - QI$$

(a) The Lyapunov energies for the encoded memory vectors are calculated to assess their stability. Lower energy indicates a more stable state.
    For [1,1,0,0,0]: -10.0
    For [0,1,1,1,0]: -6.0
    For [0,0,0,1,1]: -10.0

(b) The stable state reached for each of the 32 possible binary vectors of dimension 5, It reached for both synchronous and asynchronous updates are determined. These states are stored in dictionaries for analysis, key tuples representing the input binary vector and the values are recalled stable states.

(c) When the diagonal elements of the weight matrix are not zeroed out, the recall behavior changes. The stable states for this scenario are stored separately. The numbers of limit cycles and fixed-point attractors can be determined by analyzing these stable states.

Fixed Points: The majority of the stable states in the dictionaries are fixed points, represents the memories stored in the network.
Limit Cycles: Limit cycles are less common but can be occurred, especially in synchronous updates or when the diagonal is not zeroed, leading to oscillatory behavior.

(d) If binary decoding, employed instead of bipolar decoding, the recall behavior would likely change. Binary decoding might lead to different stable states and convergence properties compared to bipolar decoding. Choice between binary and bipolar decoding affects the network's behavior as bipolar encoding can handle negative correlations between neurons potentially leads to more accurate recall.

Update Mode: Synchronous vs. Asynchronous
Synchronous Update: In this mode, all neurons are updated simultaneously. This can lead to faster convergence but may also result in oscillations or limit cycles, where the network does not settle into a stable state. Synchronous updates can sometimes lead to incorrect recall if the initial state is far from any stored memory.

Asynchronous Update: In this, neurons are updated once at time in a random sequence. It also tends to be more stable and can avoid some oscillations seen in synchronous updates. Asynchronous updates often lead to better recall, especially when the initial state is noisy version of a stored memory.

Decoding Method:
Bipolar Decoding: It uses values of +1 and -1, which allows the network to handle negative correlations between neurons. It may lead to more accurate recall, especially in cases where the memories have complex patterns.

Binary Decoding: This uses values of 0 and 1. It simplifies the network but may leads to less accurate recall, as it can't handle negative correlations. Binary decoding might result in more fixed points but can also lead to incorrect recall if the memories are not well separated.

Complexity: Bipolar decoding is more complex but can handle more intricate memory patterns, making it suitable for applications requiring high recall accuracy.

Analysis:
The Hopfield network's behavior is influenced by the update mode (synchronous vs. asynchronous), the diagonal elements of the weight matrix, and the decoding method (bipolar vs. binary). These factors affect the stability and recall accuracy of the network.

Codes:

```
import numpy as np
import itertools

def bin2bip(binary_vector):
    """Convert binary vectors to bipolar vectors."""
    return np.where(binary_vector == 0, -1, 1)

def bip2bin(bipolar_vector):
    """Convert bipolar vectors to binary vectors."""
    return np.where(bipolar_vector == -1, 0, 1)

def compute_weights(mem_vectors, zero_diagonal=True):
    """Compute the weight matrix using bipolar outer product encoding."""
    q, n = mem_vectors.shape
    W = np.zeros((n, n))
    for i in range(q):
        W += np.outer(mem_vectors[i], mem_vectors[i])
    if zero_diagonal:
        np.fill_diagonal(W, 0)  # Zero the diagonal
    return W

def lyapunov_energy(state, weights):
    """Calculate the Lyapunov energy of a state."""
    return -0.5 * np.dot(np.dot(state, weights), state)

def update_state_synchronous(state, weights):
    """Update the state synchronously."""
    activation = np.dot(state, weights)
    return np.where(activation >= 0, 1, -1)

def update_state_asynchronous(state, weights):
    """Update the state asynchronously."""
    n = state.shape[0]
    permindex = np.random.permutation(n)
    new_state = state.copy()
```

```
    for j in permindex:
        activation = np.dot(state, weights[j])
        new_state[j] = 1 if activation >= 0 else -1
    return new_state

def hopfield_network(mem_vectors, probe, mode='synchronous', max_iter=50, zero_diagonal=True):
    bip_mem_vecs = bin2bip(mem_vectors)
    weights = compute_weights(bip_mem_vecs, zero_diagonal=zero_diagonal)

    signal_vector = bin2bip(probe)
    for _ in range(max_iter):
        if mode == 'synchronous':
            new_state = update_state_synchronous(signal_vector, weights)
        elif mode == 'asynchronous':
            new_state = update_state_asynchronous(signal_vector, weights)
        else:
            raise ValueError("Mode must be 'synchronous' or 'asynchronous'.")

        if np.array_equal(signal_vector, new_state):
            break
        signal_vector = new_state

    recalled_vector = bip2bin(signal_vector)
    return recalled_vector, lyapunov_energy(signal_vector, weights)

# Fundamental memory patterns
mem_vectors = np.array([
    [1, 1, 0, 0, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 1, 1]
])

# Calculate Lyapunov energies for each memory vector
energies = [lyapunov_energy(bin2bip(mem), compute_weights(bin2bip(mem_vectors))) for mem in
mem_vectors]
print("Lyapunov Energies:", energies)

# Generate all possible binary vectors of dimension 5
all_binary_vectors = np.array(list(itertools.product([0, 1], repeat=5)))

# Determine stable states for synchronous and asynchronous updates
synchronous_states = {}
asynchronous_states = {}

for binary_vector in all_binary_vectors:
```

```
    recalled_sync, _ = hopfield_network(mem_vectors, binary_vector, mode='synchronous')
    recalled_async, _ = hopfield_network(mem_vectors, binary_vector, mode='asynchronous')
    synchronous_states[tuple(binary_vector)] = recalled_sync
    asynchronous_states[tuple(binary_vector)] = recalled_async

# Print stable states for synchronous updates
print("\nStable States for Synchronous Updates:")
for binary_vector, stable_state in synchronous_states.items():
    print(f"Input: {binary_vector} -> Stable State: {stable_state}")

# Print stable states for asynchronous updates
print("\nStable States for Asynchronous Updates:")
for binary_vector, stable_state in asynchronous_states.items():
    print(f"Input: {binary_vector} -> Stable State: {stable_state}")

# Analyze recall behavior with non-zero diagonal elements
synchronous_states_nonzero_diag = {}
asynchronous_states_nonzero_diag = {}

for binary_vector in all_binary_vectors:
    recalled_sync, _ = hopfield_network(mem_vectors, binary_vector, mode='synchronous',
zero_diagonal=False)
    recalled_async, _ = hopfield_network(mem_vectors, binary_vector, mode='asynchronous',
zero_diagonal=False)
    synchronous_states_nonzero_diag[tuple(binary_vector)] = recalled_sync
    asynchronous_states_nonzero_diag[tuple(binary_vector)] = recalled_async

# Print stable states for synchronous updates with non-zero diagonal
print("\nStable States for Synchronous Updates (Non-Zero Diagonal):")
for binary_vector, stable_state in synchronous_states_nonzero_diag.items():
    print(f"Input: {binary_vector} -> Stable State: {stable_state}")

# Print stable states for asynchronous updates with non-zero diagonal
print("\nStable States for Asynchronous Updates (Non-Zero Diagonal):")
for binary_vector, stable_state in asynchronous_states_nonzero_diag.items():
    print(f"Input: {binary_vector} -> Stable State: {stable_state}")
```

References:

- Content Addressable Memory (CAM) using Hopfield Network