Assignment3

April 5, 2025

0.1 Question1

[2]:
```python
import numpy as np

def maxnet_network(input_vector, epsilon=-0.15):
    x = np.array(input_vector)
    n = len(x)
    # Construct the weight matrix: off-diagonals are epsilon, diagonals are 1␣
↪(compensating for epsilon)
    w = np.full((n, n), epsilon) + np.eye(n) * (1 - epsilon)

    iteration = 0
    stop = False
    while not stop:
        iteration += 1
        # Compute input for each node: u = x * w^T
        u = np.dot(x, w.T)
        # Apply ReLU activation function
        v = np.maximum(0, u)
        x = v.copy()
        # Count the number of active (non-zero) neurons
        active_neurons = np.count_nonzero(v > 0)

        # Stop if only one neuron is active
        if active_neurons <= 1:
            stop = True
            winner_index = np.where(v > 0)[0]

    if winner_index.size > 0:
        print(f"MAXNET: The winning neuron is neuron {winner_index[0] + 1} with␣
↪a final activation of {v[winner_index[0]]:.4f}, after {iteration} iterations.
↪")
    else:
        print("MAXNET: No active neuron found.")

# Test the MAXNET network
input_vector = [0.1, 0.3, 0.5, 0.7, 0.9]
maxnet_network(input_vector)
```

```
MAXNET: The winning neuron is neuron 5 with a final activation of 0.4541, after
6 iterations.
```

[3]:
```python
def kWTA_network(input_vector, k, deltaT=0.05):
    u = np.array(input_vector)
    n = len(u)
    y = 0.0          # Initialize state variable
    iteration = 0
    stop = False
    # Initialize output vector as zeros
    x = np.zeros(n)

    while not stop:
        iteration += 1
        y = y + deltaT * (np.sum(x) - k)
        # Update outputs using an infinite gain activation function:
        # if u[i] >= y, output 1; otherwise 0.
        for i in range(n):
            if u[i] >= y:
                x[i] = 1
            else:
                x[i] = 0

        active_neurons = int(np.sum(x))

        # Stop when the number of active neurons is exactly equal to k
        if active_neurons == k:
            stop = True
            winners = np.where(x == 1)[0]

    print(f"kWTA (k={k}): Winning neurons are {winners + 1} with threshold y =␣
 ↪{y:.4f}, after {iteration} iterations.")

# Test the kWTA network for k=1 and k=2
print("For k = 1:")
kWTA_network(input_vector, 1)

print("\nFor k = 2:")
kWTA_network(input_vector, 2)
```

```
For k = 1:
kWTA (k=1): Winning neurons are [5] with threshold y = 0.7000, after 8
iterations.

For k = 2:
kWTA (k=2): Winning neurons are [4 5] with threshold y = 0.5500, after 9
iterations.
```

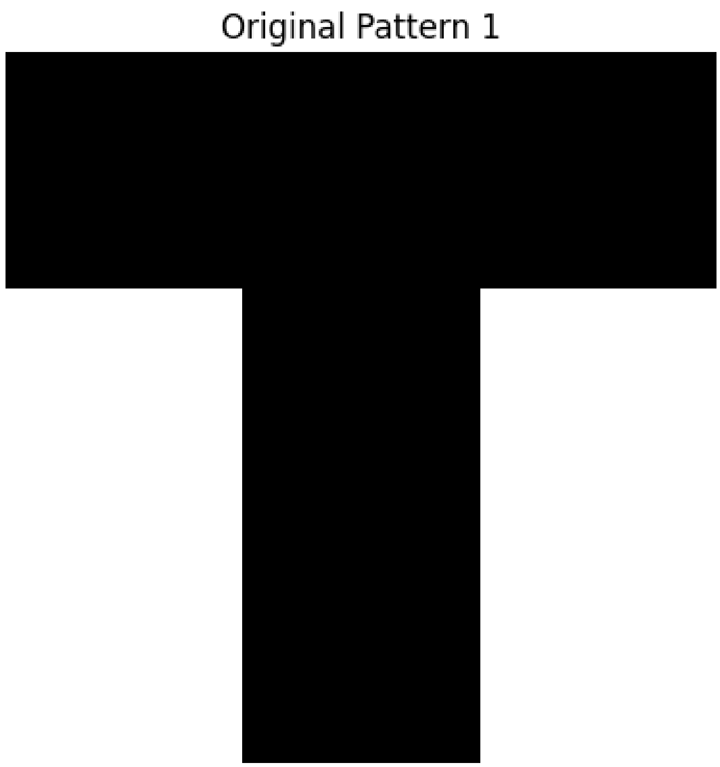
0.2 Question2

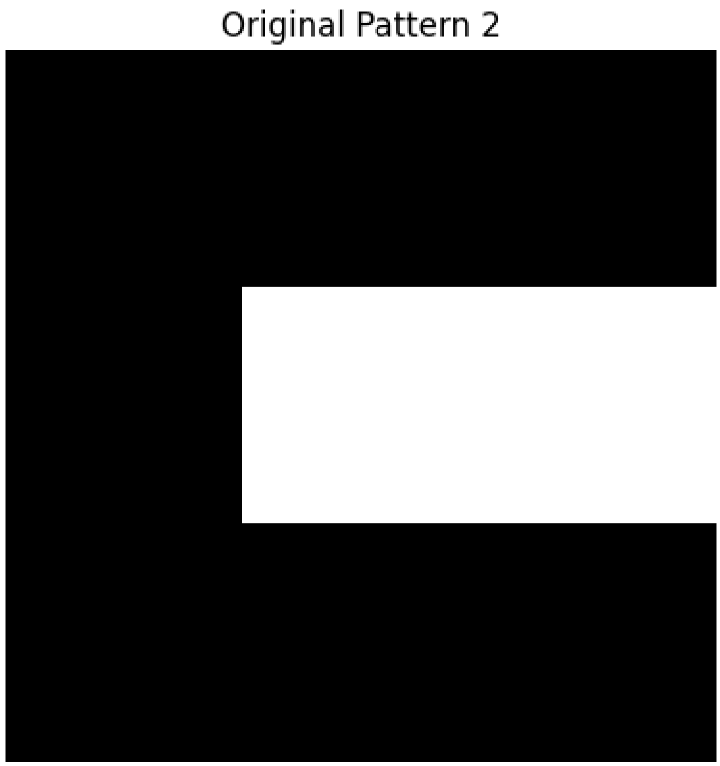
[6]:

```python
import numpy as np
import matplotlib.pyplot as plt

# Hopfield
def hopfield_network(patterns, probes, iterations=5):
    patterns = np.array(patterns)
    n = patterns.shape[1]

    #
    W = np.zeros((n, n))
    for p in patterns:
        W += np.outer(p, p)
    np.fill_diagonal(W, 0)  #    0

    #
    def plot_pattern(pattern, title):
        plt.imshow(pattern.reshape(3,3), cmap='gray')
        plt.title(title)
        plt.axis('off')
        plt.show()

    #
    print("Original Stored Patterns:")
    for i, p in enumerate(patterns):
        plot_pattern(p, f'Original Pattern {i+1}')

    #
    for idx, probe in enumerate(probes):
        print(f"\nRetrieving from Noisy Pattern {idx+1}:")
        s = probe.copy()
        for it in range(iterations):
            s = np.sign(W @ s)
            s[s==0] = 1  #   0  1
        plot_pattern(probe, f'Noisy Input {idx+1}')
        plot_pattern(s, f'Retrieved Pattern {idx+1}')

#      1  -1
pattern1 = np.array([-1, -1,  -1,
                      1, -1,   1,
                      1, -1,   1])

pattern2 = np.array([-1, -1, -1,
                    -1,  1,  1,
                    -1, -1, -1])
```

```
#     (     )
probe1 = np.array([-1, -1,  -1,
                    1, -1,   1,
                   -1, -1,   1])

probe2 = np.array([-1, -1, -1,
                    -1,  1,  1,
                    -1, -1,  1])

#  Hopfield
hopfield_network([pattern1, pattern2], [probe1, probe2])
```

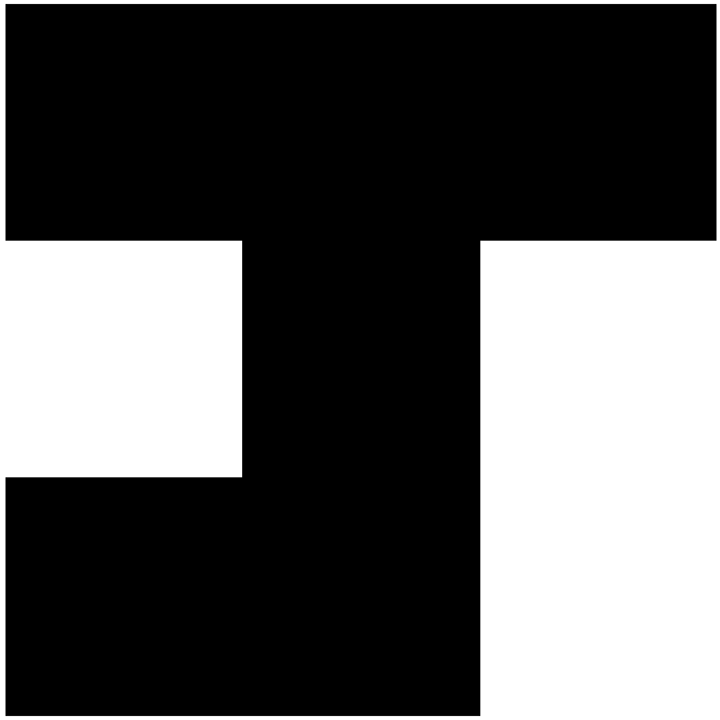
```
Original Stored Patterns:
```



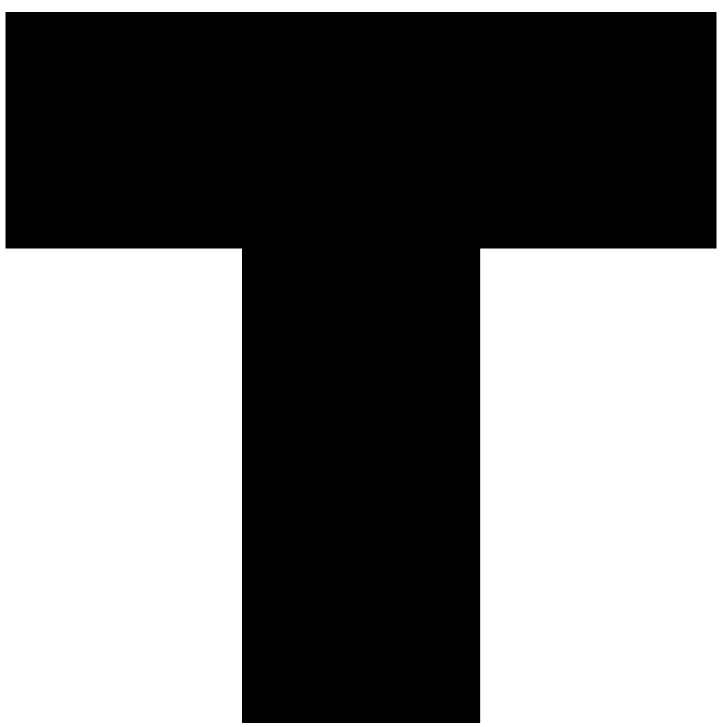



```
Retrieving from Noisy Pattern 1:
```

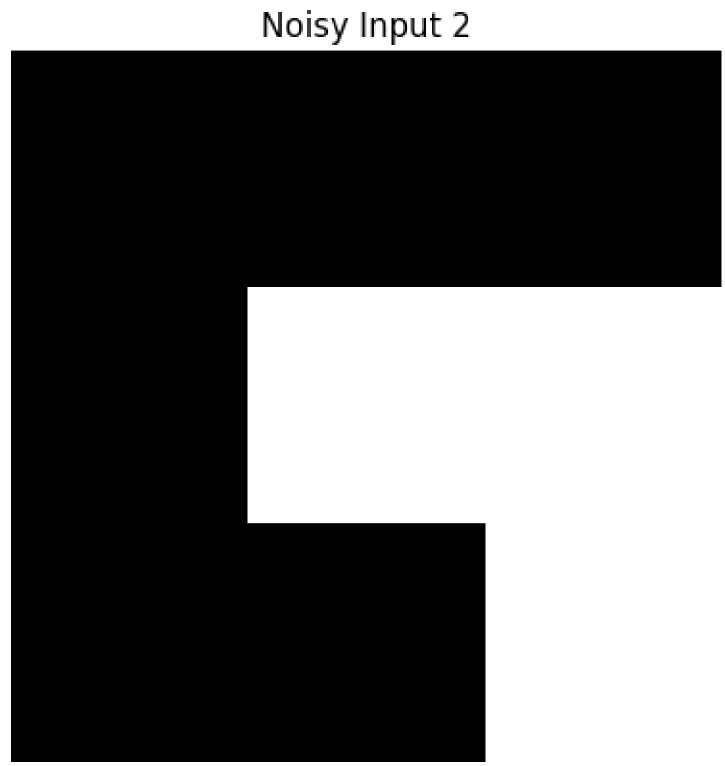
Noisy Input 1



Retrieved Pattern 1



```
Retrieving from Noisy Pattern 2:
```



Retrieved Pattern 2