1.Simulate a 5-neuron MAXNET with lateral connection weight of −0.15, and external input vector of (0.1, 0.3, 0.5, 0.7, 0.9). Do the same for the kWTA network with k = 1 and 2.

```
import numpy as np

def maxnet(input_vector, lateral_weight=-0.15, iterations=10):
    n = len(input_vector)
    output = np.copy(input_vector)

    for _ in range(iterations):
        net_input = np.dot(lateral_weight * np.ones((n, n)) + np.eye(n), output)
        output = np.maximum(0, net_input)

    return output

input_vector = np.array([0.1, 0.3, 0.5, 0.7, 0.9])
maxnet_output = maxnet(input_vector)
print("MAXNET Output:", maxnet_output)
```

```
MAXNET Output: [0.         0.         0.         0.         0.08679375]
```

```
def kwta(input_vector, k, lateral_weight=-0.15):
    n = len(input_vector)
    output = np.copy(input_vector)

    for _ in range(10):    # Fixed number of iterations
        net_input = np.dot(lateral_weight * np.ones((n, n)) + np.eye(n), output)

        top_k_indices = np.argsort(net_input)[-k:]
        output = np.zeros_like(net_input)
        output[top_k_indices] = net_input[top_k_indices]

    return output

# k-WTA outputs for k=1 and k=2
kwta_output_k1 = kwta(input_vector, k=1)
kwta_output_k2 = kwta(input_vector, k=2)

print("kWTA Output (k=1):", kwta_output_k1)
print("kWTA Output (k=2):", kwta_output_k2)
```

```
kWTA Output (k=1): [0.         0.         0.         0.         0.1215989]
kWTA Output (k=2): [ 0.          0.          0.         -0.01459294  0.09940771]
```

2.Simulate a 9-neuron discrete Hopfield network as an associative memory of 3-by-3 digital images as shown in Fig. 1. First determine the connection weights using the outer product rule. Then retrieve the two store patterns by using two keys (probes) with variations of one pixel.

```
import  numpy  as  np

P1  =  np.array([+1,  +1,  +1,  -1,  +1,  -1,  -1,  +1,  -1])
P2  =  np.array([+1,  +1,  +1,  +1,  -1,  -1,  +1,  +1,  +1])

W  =  np.outer(P1,  P1)  +  np.outer(P2,  P2)  -  2  *  np.eye(9)

def  sign(x):
        return  np.where(x  >=  0,  1,  -1)

def  retrieve_pattern(W,  noisy_input,  max_iter=10):
        s  =  noisy_input.copy()
        for  _  in  range(max_iter):
                s_new  =  sign(np.dot(W,  s))
                if  np.array_equal(s,  s_new):
                        break
                s  =  s_new
        return  s

noisy_input1  =  np.array([+1,  +1,  +1,  -1,  +1,  -1,  +1,  +1,  -1])
retrieved1  =  retrieve_pattern(W,  noisy_input1)

noisy_input2  =  np.array([+1,  +1,  +1,  +1,  -1,  -1,  +1,  +1,  -1])
retrieved2  =  retrieve_pattern(W,  noisy_input2)

print("Retrieved  Pattern  1:",  retrieved1)
print("Retrieved  Pattern  2:",  retrieved2)
```

```
Retrieved Pattern 1: [ 1  1  1 -1  1 -1 -1  1 -1]
Retrieved Pattern 2: [ 1  1  1  1 -1 -1  1  1  1]
```