```
import numpy as np


x1=[ [ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1,
x1 = np.array(x1).flatten()
x2=[ [ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -
x2 = np.array(x2).flatten()
x3=[ [ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1],[ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1, -1, -1, -1, -1, 1, 1, 1, -1, -1],[ -1, -1, -1,
x3 = np.array(x3).flatten()
x4=[ [ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1, -1, 1, 1, 1, 1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, -
x4 = np.array(x4).flatten()
x5=[ [ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1
x5 = np.array(x5).flatten()


patterns = np.vstack((x1, x2, x3, x4, x5))
P_num, P_length = patterns.shape
W = np.zeros((P_length, P_length))

for i in range(P_length):
    for j in range(P_length):
        if i != j:
            W[i, j] = (1 / P_length) * sum(patterns[p, i] * patterns[p, j]for p in range(P_num))

np.fill_diagonal(W,0)

print("Weight Matrix W:")
print(W)

max_iterations = 1000
```

```
Weight Matrix W:
[[ 0.       0.01875  0.00625 ...  0.00625  0.00625  0.01875]
 [ 0.01875  0.       0.01875 ...  0.01875  0.01875  0.00625]
 [ 0.00625  0.01875  0.      ...  0.03125  0.00625 -0.00625]
 ...
 [ 0.00625  0.01875  0.03125 ...  0.       0.00625 -0.00625]
 [ 0.00625  0.01875  0.00625 ...  0.00625  0.       0.01875]
 [ 0.01875  0.00625 -0.00625 ... -0.00625  0.01875  0.     ]]
```

```
Fed_1=[[1, -1, 1, -1, 1, -1, 1, -1, 1, -1], [1, -1, 1, 1, -1, 1, -1, -1, 1, -1], [1, -1, -1, 1, -1, 1, -1, 1, 1, -1], [1, 1, -
Fed_2=[[1, -1, 1, 1, -1, 1, -1, -1, 1, -1], [1, -1, 1, 1, -1, 1, -1, -1, 1, -1], [1, -1, 1, 1, -1, 1, -1, -1, 1, -1], [1, -1,
Fed_3=[[1, -1, 1, 1, 1, -1, 1, 1, -1, -1], [1, -1, 1, 1, 1, -1, 1, 1, 1, -1], [1, -1, -1, -1, -1, 1, 1, 1, 1, -1], [1, -1, -1,


S = np.array(Fed_3).flatten()
B = np.zeros((P_length,1))

for n in range(max_iterations):
    previous_S = S.copy()

    for i in range(P_length):
        B[i] = np.dot(W[i],S)
        if B[i] == 0:
            S[i] = 1
        else:
            S[i] = np.sign(B[i])

    if np.array_equal(S, previous_S):
        print("Network has reached a stable state:")
        print(n)
        break

print(S)
```

```
Network has reached a stable state:
1
[-1 -1  1  1  1  1  1  1 -1 -1 -1 -1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1
 -1 -1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1 -1 -1 -1  1  1
  1 -1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1
  1  1  1  1  1  1 -1 -1 -1 -1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  1  1  1 -1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1 -1 -1 -1  1  1  1 -1
 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1  1  1
  1  1  1  1  1 -1 -1 -1  1  1  1  1  1  1 -1 -1]
```

```
matched_p_index = None

for i, p in enumerate(patterns):
    if np.array_equal(S, p):
        matched_p_index = i+1
```

```
            matched_p = p
            break
        elif np.array_equal(S, -p):
            matched_p_index = i+1
            matched_p = p
            break

if matched_p_index is not None:
    if np.array_equal(S, patterns[matched_p_index]):
        print("Network has recognized a stored pattern:")
    else:
        print("Network has recognized an inverted stored pattern:")
    print(f"Pattern Index: {matched_p_index}")
    print(f"Matched Pattern: {matched_p}")
else:
    print("Network did not recognize any stored pattern or inverted form.")
```

```
 Network has recognized an inverted stored pattern:
 Pattern Index: 4
 Matched Pattern: [-1 -1  1  1  1  1  1  1 -1 -1 -1 -1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1
  -1 -1  1  1  1 -1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1 -1 -1 -1  1  1
   1 -1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1
   1  1  1  1  1  1 -1 -1 -1 -1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1
   1  1  1 -1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1 -1 -1 -1  1  1  1 -1
  -1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1  1  1  1 -1 -1 -1  1  1
   1  1  1  1  1 -1 -1 -1  1  1  1  1  1  1 -1 -1]
```

```
matrix = S.reshape(16, 10)
matrix
```

```
 array([[-1, -1,  1,  1,  1,  1,  1,  1, -1, -1],
        [-1, -1,  1,  1,  1,  1,  1,  1,  1, -1],
        [-1, -1, -1, -1, -1, -1,  1,  1,  1, -1],
        [-1, -1, -1, -1, -1, -1,  1,  1,  1, -1],
        [-1, -1, -1, -1, -1, -1,  1,  1,  1, -1],
        [-1, -1, -1, -1, -1, -1,  1,  1,  1, -1],
        [-1, -1, -1, -1, -1, -1,  1,  1,  1, -1],
        [-1, -1,  1,  1,  1,  1,  1,  1, -1, -1],
        [-1, -1,  1,  1,  1,  1,  1,  1, -1, -1],
        [-1, -1, -1, -1, -1, -1,  1,  1,  1, -1],
        [-1, -1, -1, -1, -1, -1,  1,  1,  1, -1],
        [-1, -1, -1, -1, -1, -1,  1,  1,  1, -1],
        [-1, -1, -1, -1, -1, -1,  1,  1,  1, -1],
        [-1, -1,  1,  1,  1,  1,  1,  1,  1, -1],
        [-1, -1,  1,  1,  1,  1,  1,  1, -1, -1]])
```