In [1]:

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

In [2]:

```
# Patterns
x1=[ [ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1,

x2=[ [ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1]

x3=[ [ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1],[ 1, 1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1, -

x4=[ [ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1, -1, 1, 1, 1, 1, 1, 1, 1, -1],[ -1

x5=[ [ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1]
```

Hebb's rule

calculate the weight matrix W using Hebb's rule. To do so use the 5 patterns given above.

In [3]:

```
# convert the patterns to 1D numpy arrays
x1=np.array(x1).flatten()
x2=np.array(x2).flatten()
x3=np.array(x3).flatten()
x4=np.array(x4).flatten()
x5=np.array(x5).flatten()

x = np.array([x1,x2,x3,x4,x5])

N = x1.size

# create the weight matrix
W=np.zeros((x1.size,x1.size))
```

In [4]:

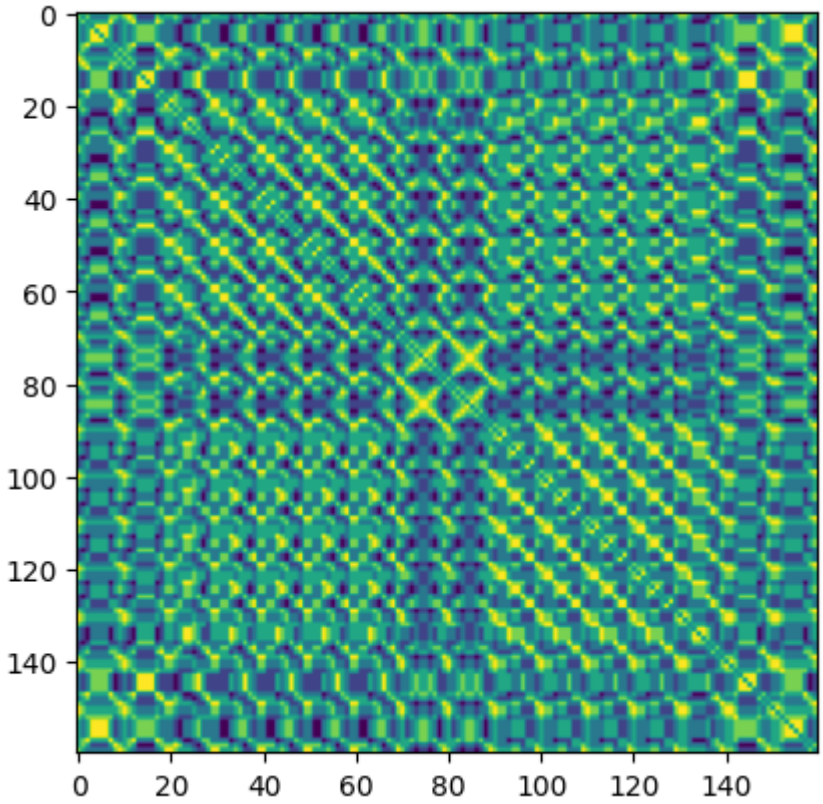
```
# update the weight matrix
for i in range(x1.size):
    for j in range(x1.size):
        for p in range(x.shape[0]):
            if i!=j:
                W[i,j]=W[i,j]+x[p][i]*x[p][j]
        # normalize with 1/N
        W[i,j]=W[i,j]/N
```

In [5]:

```
# print the weight matrix
plt.imshow(W)
```

Out[5]:

```
<matplotlib.image.AxesImage at 0x110818c10>
```



Calculate the asynchronous update

In [6]:

```
def getNextS(w, s):
    b = np.zeros(s.size)
    N = s.size
    for i in range(N):
        # calculate b_i
        for j in range(N):
            b[i] = b[i] + w[i,j]*s[j]
        # update s based on b_i
        tmp = 0
        if b[i] >= 0:
            tmp = 1
        else:
            tmp = -1
        s[i] = tmp
    return s
```

Test on custom pattern

In [7]:

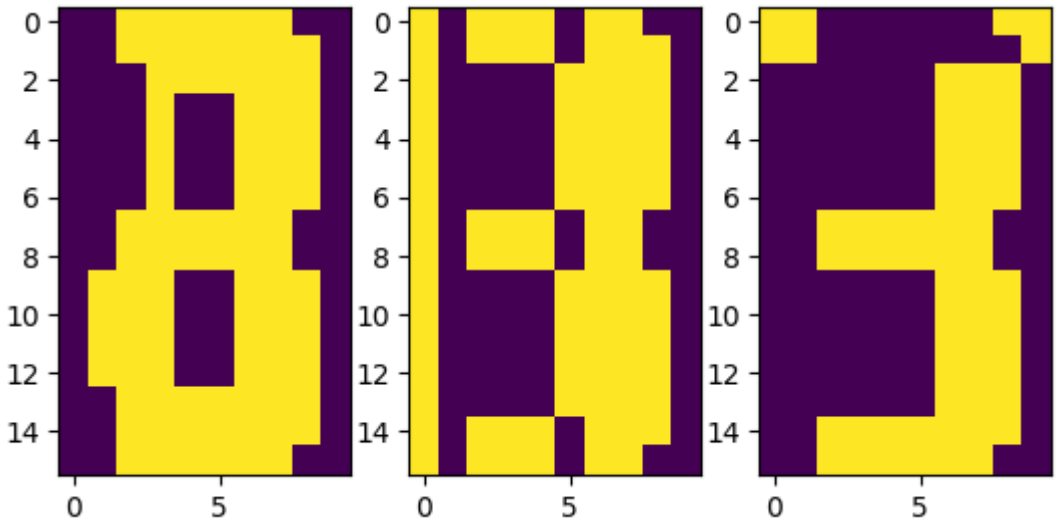
```
pattern_1A = [[-1, -1, 1, 1, 1, 1, 1, 1, -1, -1], [-1, -1, 1, 1, 1, 1, 1, 1, 1, -
pattern_2A = [[1, -1, 1, 1, 1, -1, 1, 1, -1, -1], [1, -1, 1, 1, 1, -1, 1, 1, 1, -
pattern_3A = [[1, 1, -1, -1, -1, -1, -1, -1, 1, 1], [1, 1, -1, -1, -1, -1, -1, -1

pattern_1A = np.array(pattern_1A).flatten()
pattern_2A = np.array(pattern_2A).flatten()
pattern_3A = np.array(pattern_3A).flatten()
```

In [8]:

```
# visualize the patterns
plt.subplot(1,3,1)
plt.imshow(pattern_1A.reshape(16,10))
plt.subplot(1,3,2)
plt.imshow(pattern_2A.reshape(16,10))
plt.subplot(1,3,3)
plt.imshow(pattern_3A.reshape(16,10))
```

Out[8]:

```
<matplotlib.image.AxesImage at 0x110961fc0>
```



In [9]:

```
# get the next state
S_1A = getNextS(W, pattern_1A)
S_1A = getNextS(W, S_1A)
S_2A = getNextS(W, pattern_2A)
S_3A = getNextS(W, pattern_3A)

# reshape the pattern to 16x10
S_1A = S_1A.reshape(16,10)
S_2A = S_2A.reshape(16,10)
S_3A = S_3A.reshape(16,10)

# save the pattern to a file in typescript format
np.savetxt('pattern_1A.txt', S_1A, fmt='%d', delimiter=',', newline='],\n [', he
np.savetxt('pattern_2A.txt', S_2A, fmt='%d', delimiter=',', newline='],\n [', he
np.savetxt('pattern_3A.txt', S_3A, fmt='%d', delimiter=',', newline='],\n [', he
```

In [10]:

```
# visualize the patterns
plt.subplot(1,3,1)
plt.imshow(S_1A)
plt.subplot(1,3,2)
plt.imshow(S_2A)
plt.subplot(1,3,3)
plt.imshow(S_3A)
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x110a02380>
```