EE550 - Binary Hopfield Neural Network Implementation with Python

March 6, 2017

0.1 *Mine Melodi Çalışkan - 2015705009*

Create $\mu = 4$ sample patterns. Pick a graphical grid of 8 * 8 pixels.
Images of numerals “0,1,2,3” from a string representation:

```
In [1]: zero="""
        XXXXXXXX
        X______X
        X______X
        X______X
        X______X
        X______X
        X______X
        XXXXXXXX
        """
```

```
In [2]: one="""
        ____X___
        ____X___
        ____X___
        ____X___
        ____X___
        ____X___
        ____X___
        ____X___
        """
```

```
In [3]: two="""
        XXXXXXXX
        _______X
        _______X
        XXXXXXXX
        X_______
        X_______
        XXXXXXXX
        ________
        """
```

In [4]:
```
three="""
________
XXXXXXXX
_______X
_______X
XXXXXXXX
_______X
_______X
XXXXXXXX
"""
```

We should transform images of numerals to bipolar vectors. “bi_vec_pattern()” function converts “X” to 1 and “_" to -1 and merges rows together.

Resulting bipolar vectors are 64 dimensional.

In [5]:
```
def bi_vec_pattern(numeral):
    from numpy import array
    return array([+1 if i=='X' else -1 for i in numeral.replace('\n','')])
```

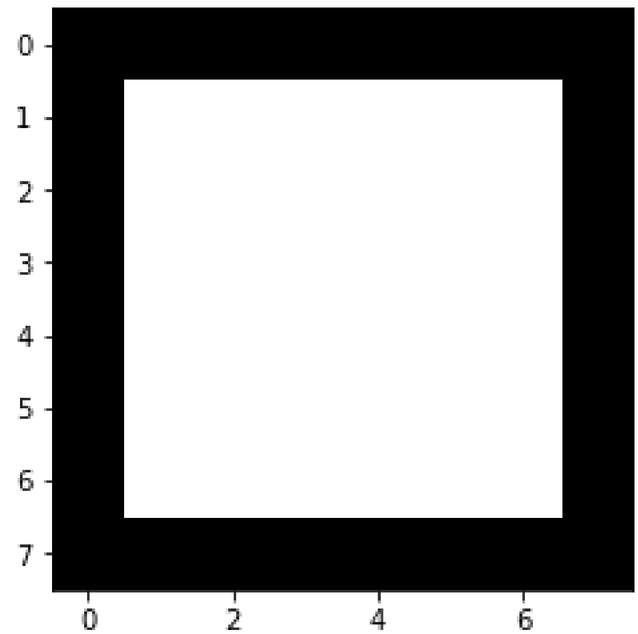
In [6]:
```
bipolar_0=bi_vec_pattern(zero)
bipolar_1=bi_vec_pattern(one)
bipolar_2=bi_vec_pattern(two)
bipolar_3=bi_vec_pattern(three)
```

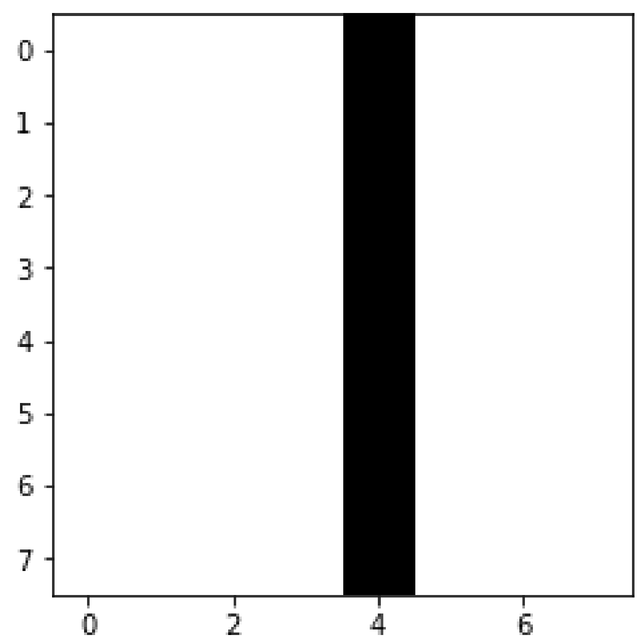
Print each sample graphically to visualize it.

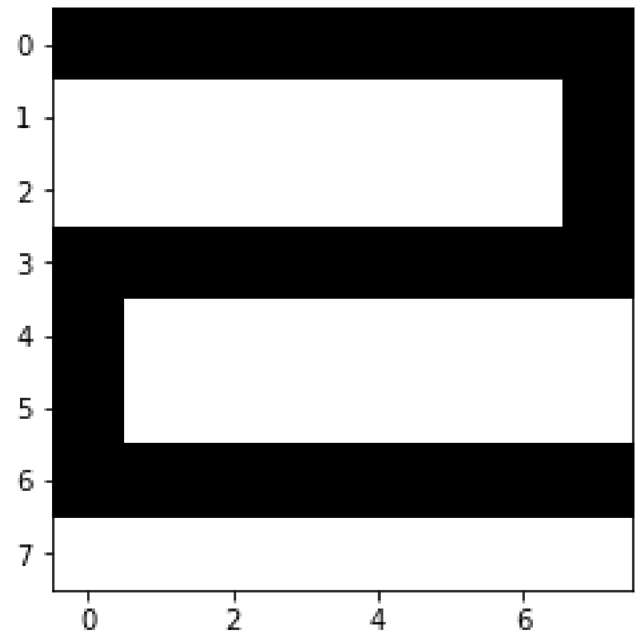
Following “visualize()” function reshapes the pattern vector to a matrix and plots it as an image.

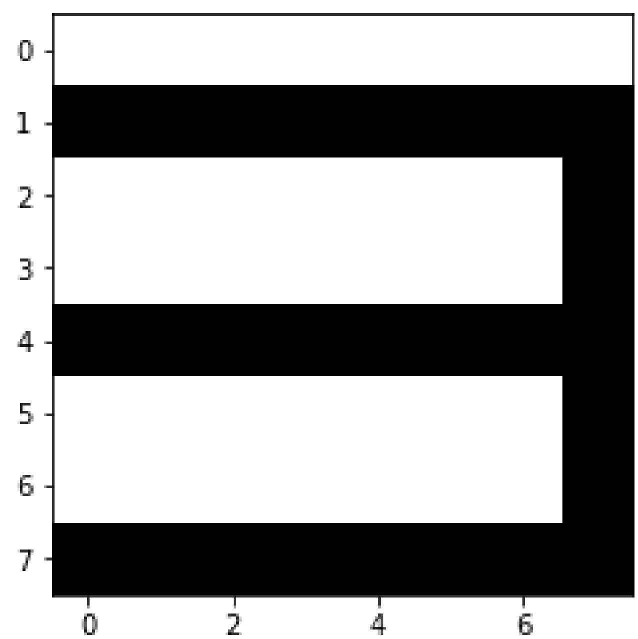
In [7]:
```
def visualize(pattern):
    from pylab import imshow, cm, show
    imshow(pattern.reshape((8,8)),cmap=cm.binary, interpolation='nearest')
    show()
```

In [8]:
```
visualize(bipolar_0)
visualize(bipolar_1)
visualize(bipolar_2)
visualize(bipolar_3)
```


0
1
2
3
4
5
6
7
0
2
4
6


0
1
2
3
4
5
6
7
0
2
4
6





Step 1) Find the connection weight $T_{ij} $ with learning rule:

$$T_{ij} = \begin{cases} \sum x_i^s x_j^s & i \neq j \\ 0 & i = j \end{cases}$$

In [9]:
```
import numpy as np
#store sample patterns as rows of the matrix sample_patterns
sample_patterns = np.array([bipolar_0,bipolar_1,bipolar_2,bipolar_3])
```

In [10]:
```
def network_train(sample_patterns):
    from numpy import zeros, outer, diag_indices
    i=sample_patterns.shape[1]
    j=sample_patterns.shape[0]
    T = zeros((i,i)) #create 8*8 zero matrix

    #weight matrix as summations of outer products of sample vectors.
    for s in sample_patterns:
        T = T + outer(s,s)

    #set diagonal entries to zero to avoid self inputs.
    T[diag_indices(i)] = 0

    return T/j #normalization
```

In [11]:
```
T=network_train(sample_patterns)
```

In [12]:
```
T
```

Out[12]:
```
array([[ 0.,  1.,  1., ...,  0.,  0.,  0.],
       [ 1.,  0.,  1., ...,  0.,  0.,  0.],
       [ 1.,  1.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  1.,  1.],
       [ 0.,  0.,  0., ...,  1.,  0.,  1.],
       [ 0.,  0.,  0., ...,  1.,  1.,  0.]])
```

Build the discrete dynamic model using: $\mu_j(k+1) = f_h[\sum T_{ij}\mu_i(k)]$

Sign function: $f_h = \begin{cases} 1 & x > 0 \\ -1 & x \leq 0 \end{cases}$

In the above formula patterns $\mu(k)$'s are column vectors and shows the iteration for an individual vector.

Following "classification function" performs iteration on each individual vectors at once.

Here test patterns are contained as rows in "test_patterns" matrix.

And knowing that the weight matrix T is symmetric, it's convenient to take the formula as:

$\mu(k+1) = f_h[\sum \mu(k)T]$

In [13]:
```
from numpy import vectorize, dot
sgn = vectorize(lambda x: -1 if x<0 else +1)

def classify(T, test_patterns, iteration_steps):
    for i in xrange(iteration_steps):
        test_patterns = sgn(dot(test_patterns,T))
    return test_patterns
```

0.1.1 Case 1

In [14]:
```
zero_test_1="""
XXXXXXX_
X______X
X___X___
X_______
X______X
_______X
X______X
XXXXXX_X
"""
```

In [15]:
```
one_test_1="""
_____XXX
_______X
_______X
________
_______X
_______X
_______X
______X_
"""
```

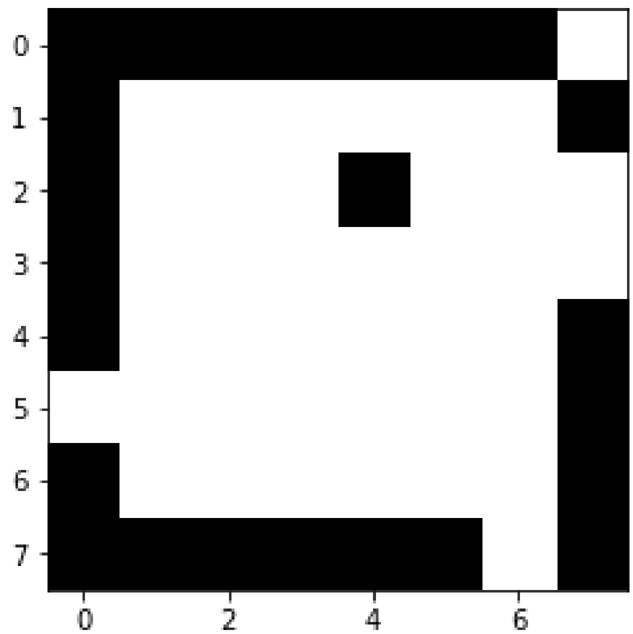
In [16]:
```
two_test_1="""
XXXXXXXX
______X_
_______X
X__XXXXX
X_______
X_______
XXXXXXXX
________
"""
```

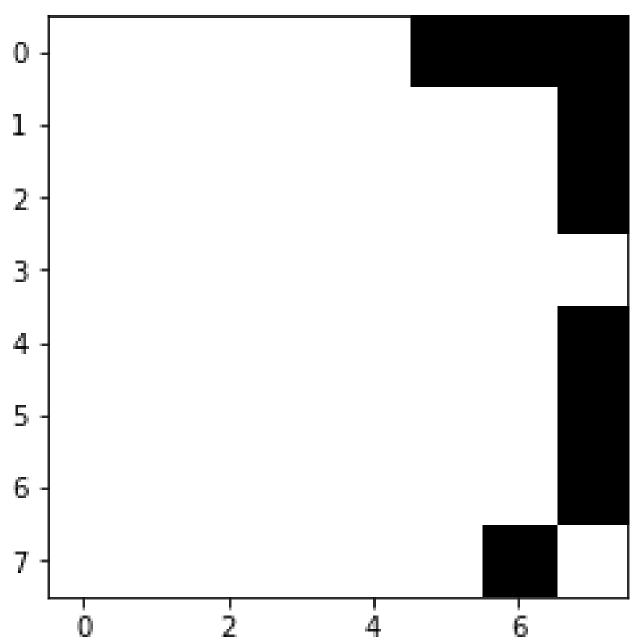
In [17]:
```
three_test_1="""
________
X___XXXX
_______X
_______X
X_XXXXXX
_______X
__X____X
XX_XXXXX
"""
```

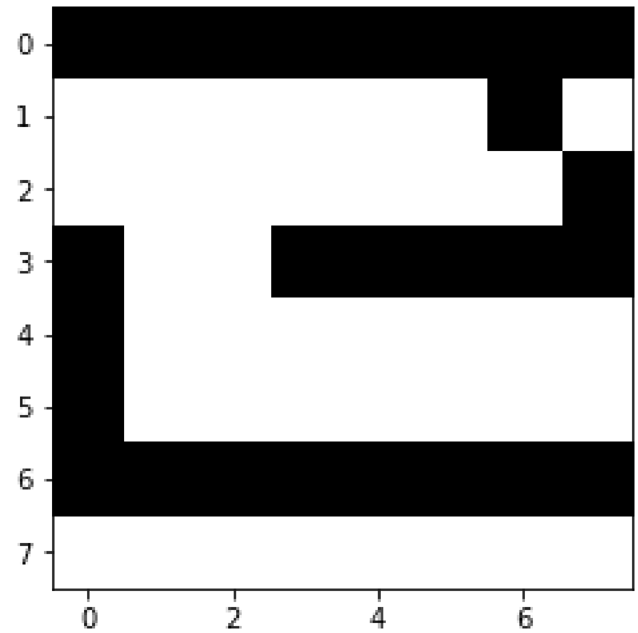
In [18]:
```
#Stores test patterns as rows in test_patterns matrix.

test_patterns_1= np.array([bi_vec_pattern(zero_test_1),
```

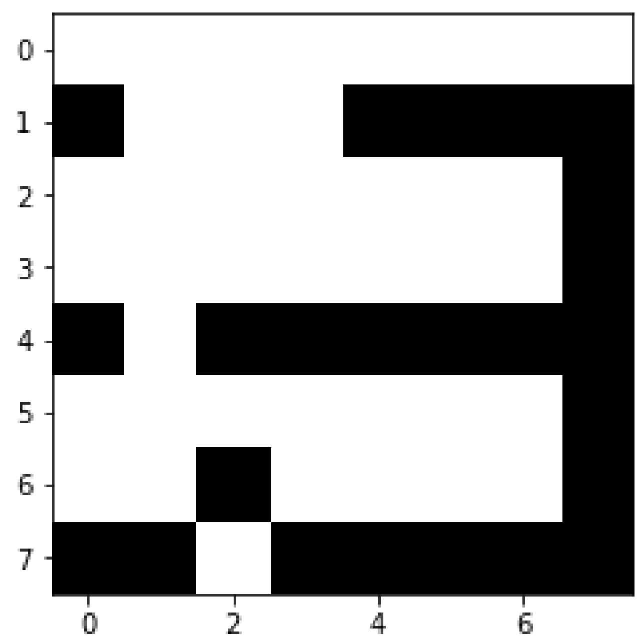
```
                                    bi_vec_pattern(one_test_1),
                                    bi_vec_pattern(two_test_1),
                                    bi_vec_pattern(three_test_1)])
```

```
In [19]: for i in range(4):
             visualize(test_patterns_1[i])
```





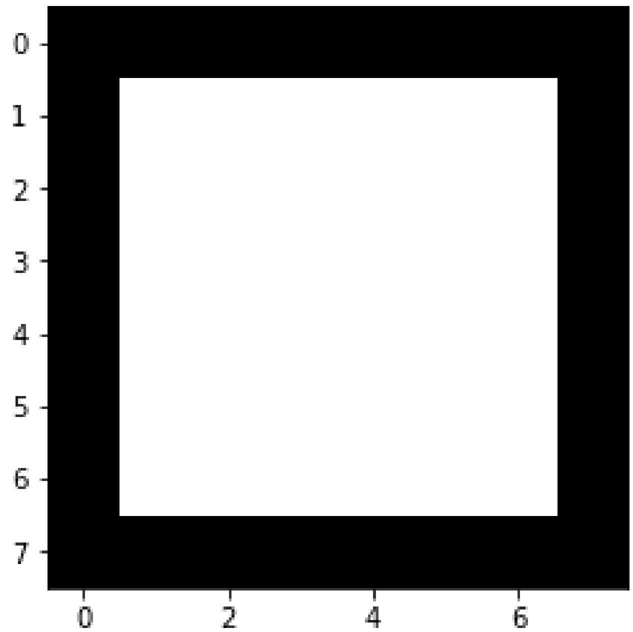


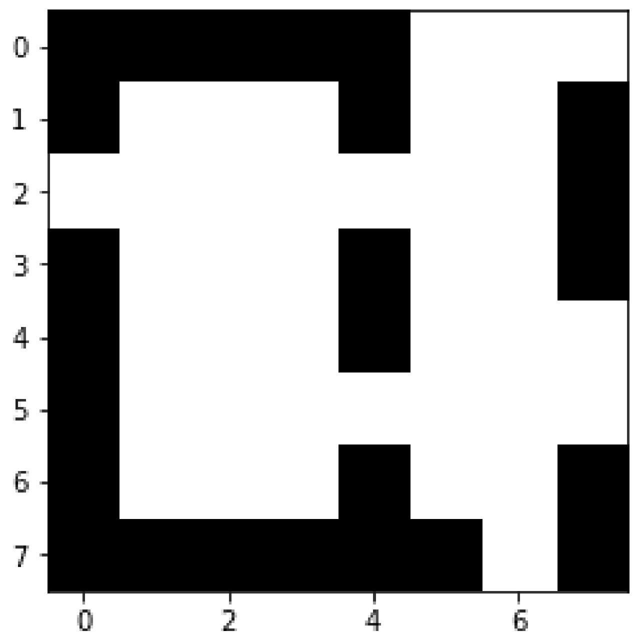


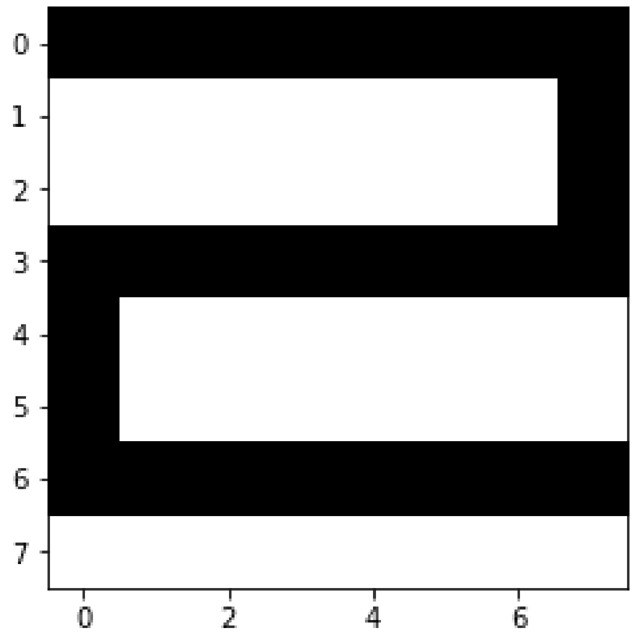
```
In [20]: result_1=classify(T, test_patterns_1, 1)
         result_2=classify(T, test_patterns_1, 2)
         result_3=classify(T, test_patterns_1, 3)
```

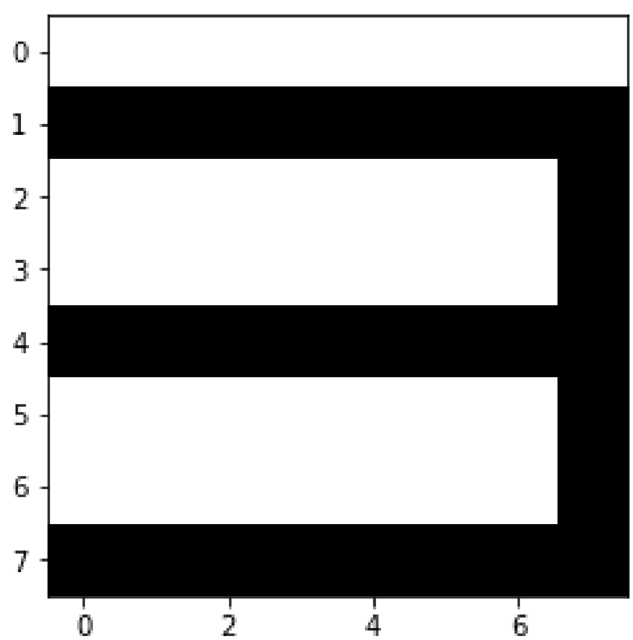
Iteration step=1

```
In [21]: for i in range(4):
             visualize(result_1[i])
```



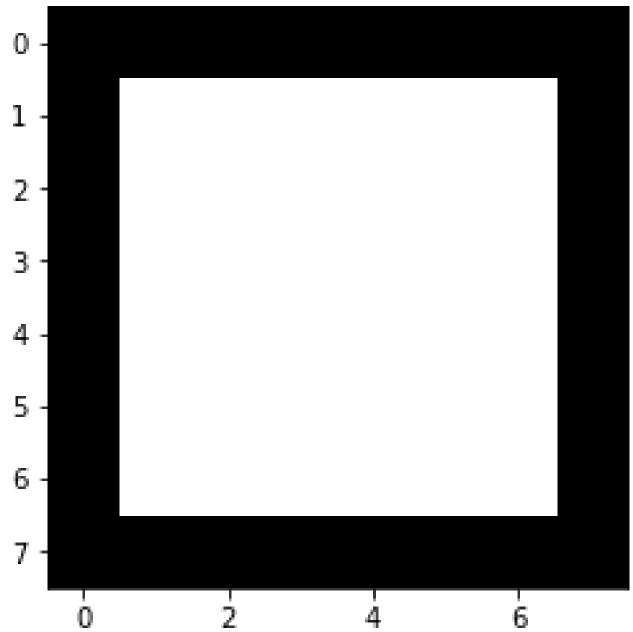


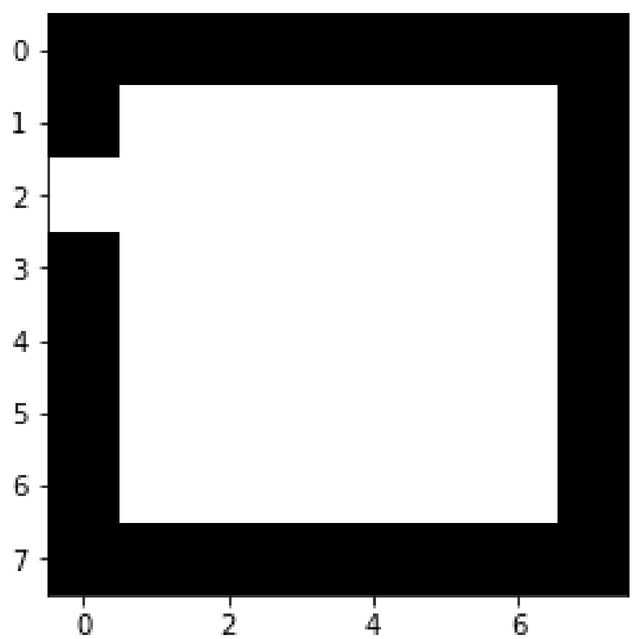


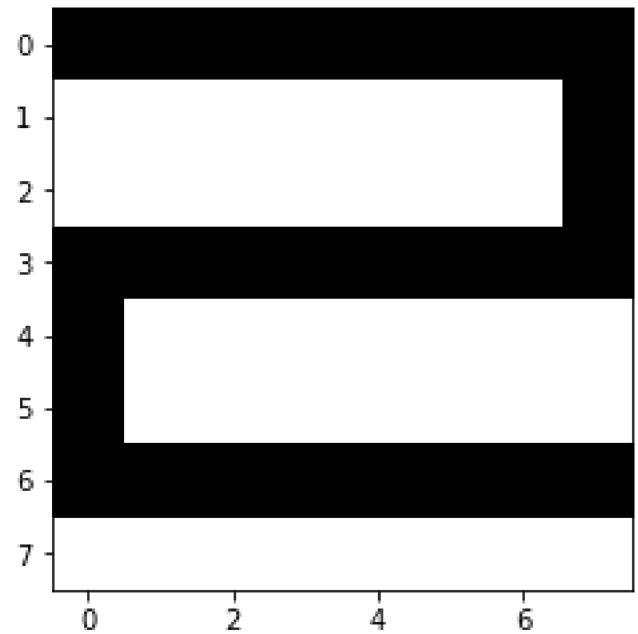


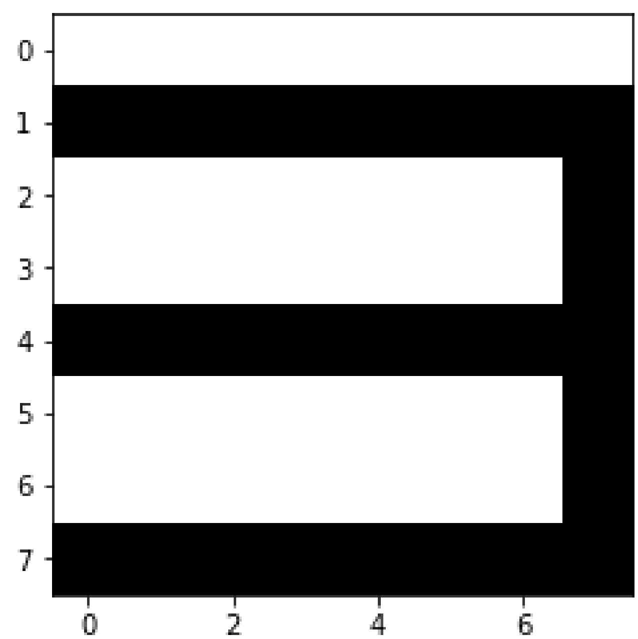
Iteration step=2

In [22]:
```
for i in range(4):
    visualize(result_2[i])
```

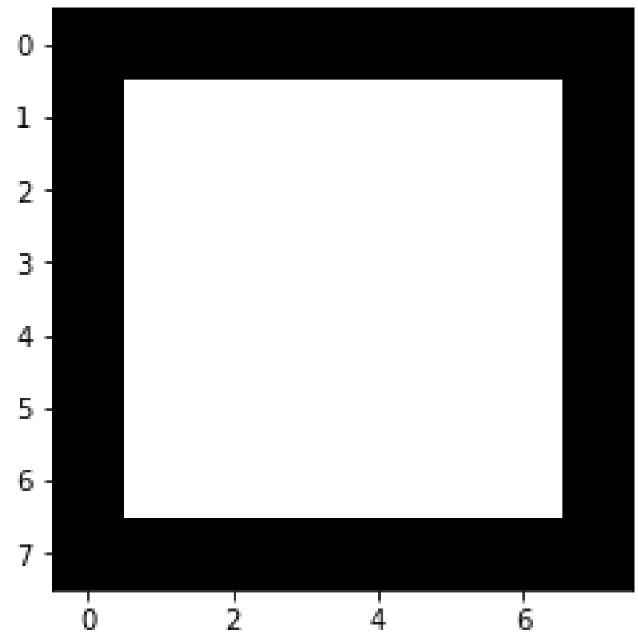


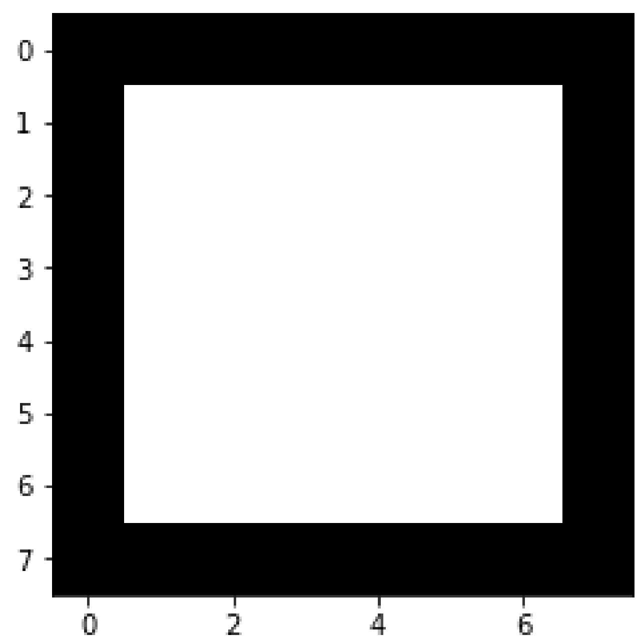


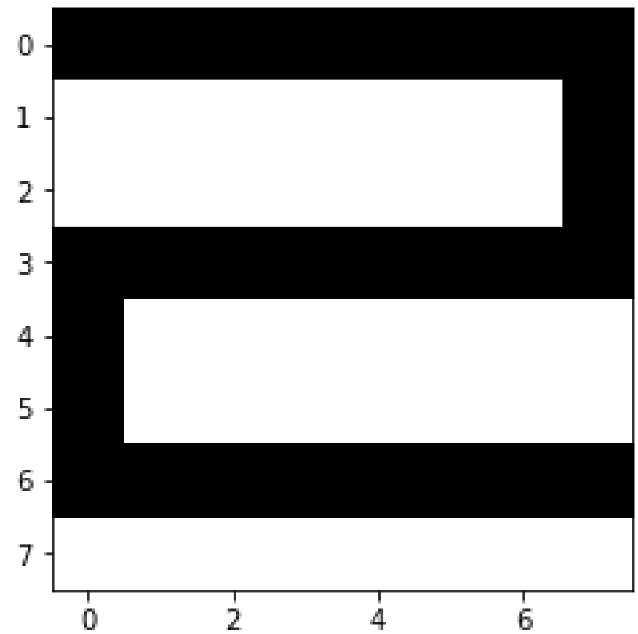


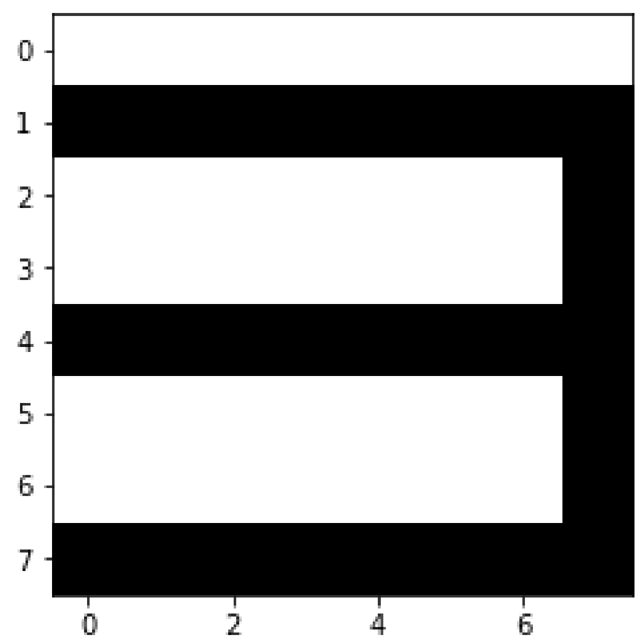



```
In [23]: for i in range(4):
             visualize(result_3[i])
```


0
1
2
3
4
5
6
7
0
2
4
6


0
1
2
3
4
5
6
7
0
2
4
6





In the first case there is a failure: "1" converged to "0"

0.1.2 Case 2

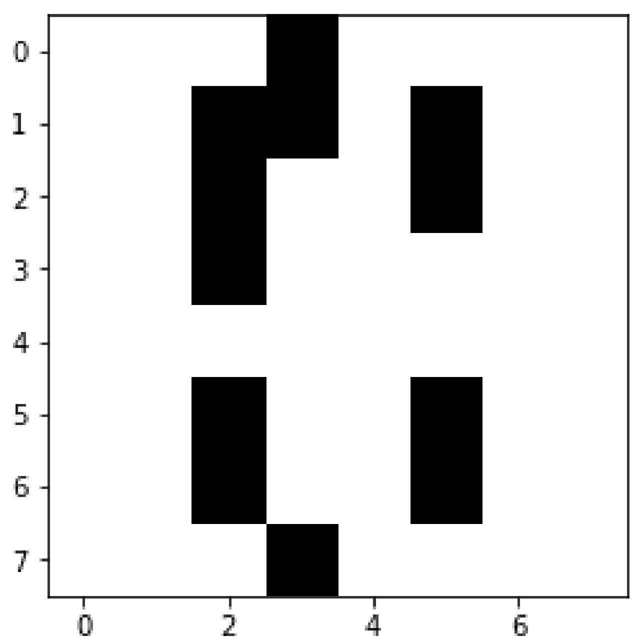
In [24]:
```
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
#8*8 images of digits data set from sklearn
digits = load_digits()
print(digits.data.shape)
```

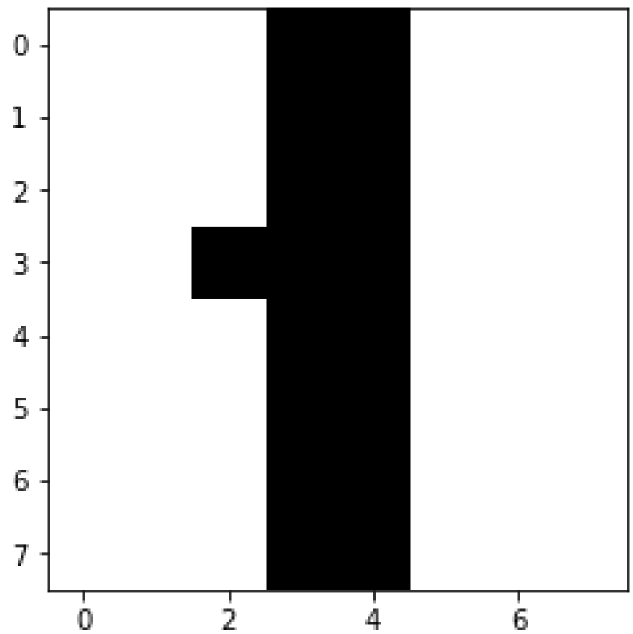
```
(1797L, 64L)
```

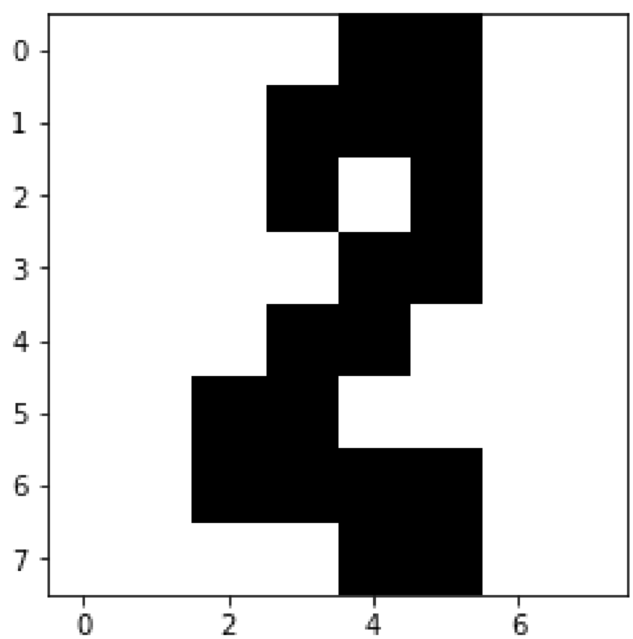
In [25]:
```
def to_bipolar(img, lower, upper): #convert images to bipolar arrays
    img=(lower < img) & (img < upper)
    img=img.astype(int)
    img=np.asarray([j for i in img for j in i])
    img[img==0]=-1
    return img
```

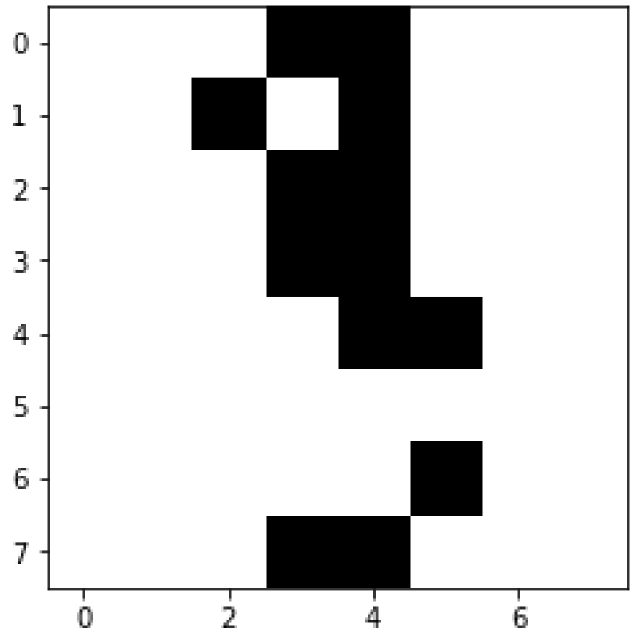
In [26]:
```
zero_test_3=to_bipolar(digits.images[0],10,100)
one_test_3=to_bipolar(digits.images[1],10,100)
two_test_3=to_bipolar(digits.images[2],10,100)
three_test_3=to_bipolar(digits.images[3],10,100)
```

In [27]:
```
visualize(zero_test_3)
visualize(one_test_3)
visualize(two_test_3)
visualize(three_test_3)
```




0
1
2
3
4
5
6
7
0
2
4
6


0
1
2
3
4
5
6
7
0
2
4
6

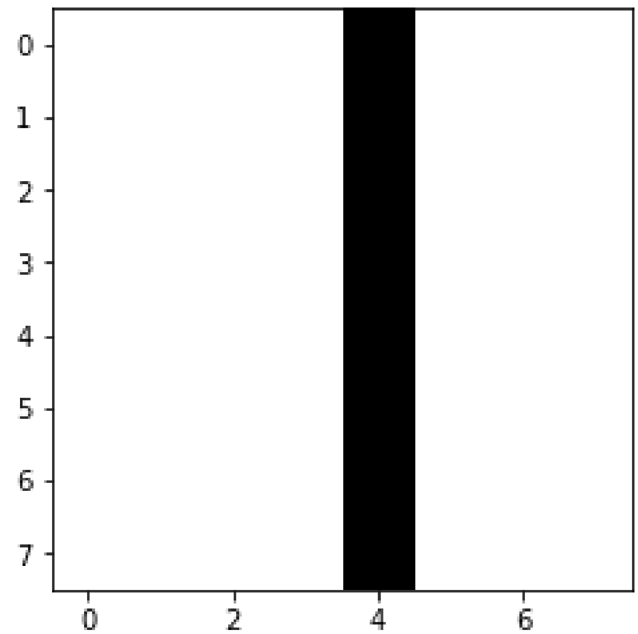


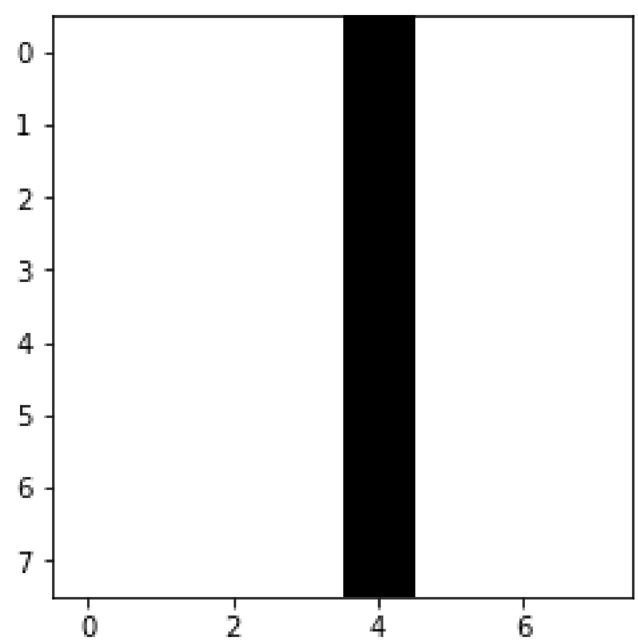

```
In [28]: test_patterns_3 = [zero_test_3, one_test_3,two_test_3,three_test_3]
```

```
In [29]: result1_=classify(T, test_patterns_3, 1)
         result2_=classify(T, test_patterns_3, 2)
```

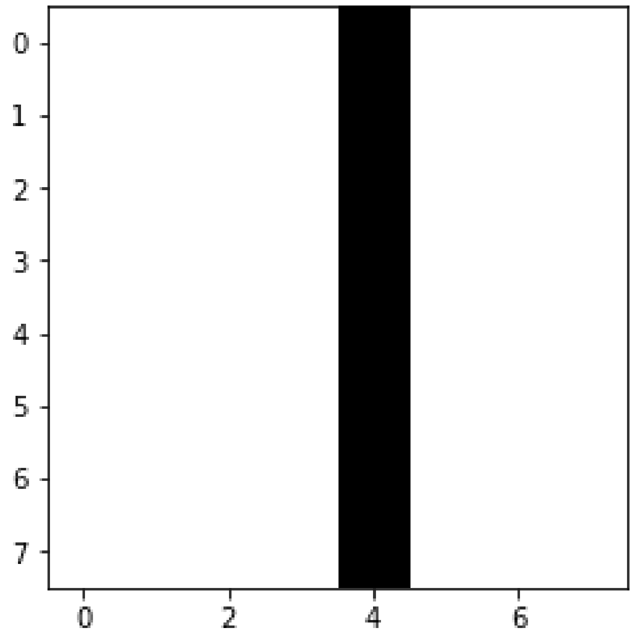
Iteration step=1

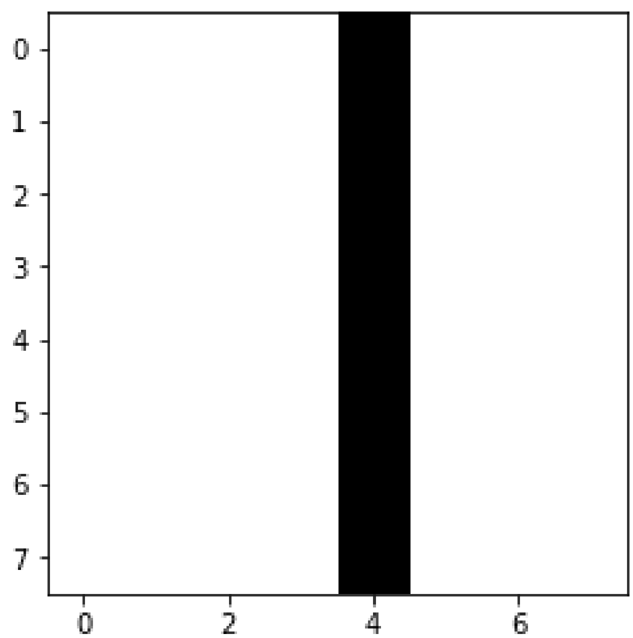
```
In [31]: for i in range(4):
             visualize(result1_[i])
```


0
1
2
3
4
5
6
7
0
2
4
6


0
1
2
3
4
5
6
7
0
2
4
6

0.1.3 CASE 3





In the 2nd case 3 out of 4 patterns converged to undesired patterns.

0.1.3 CASE 3

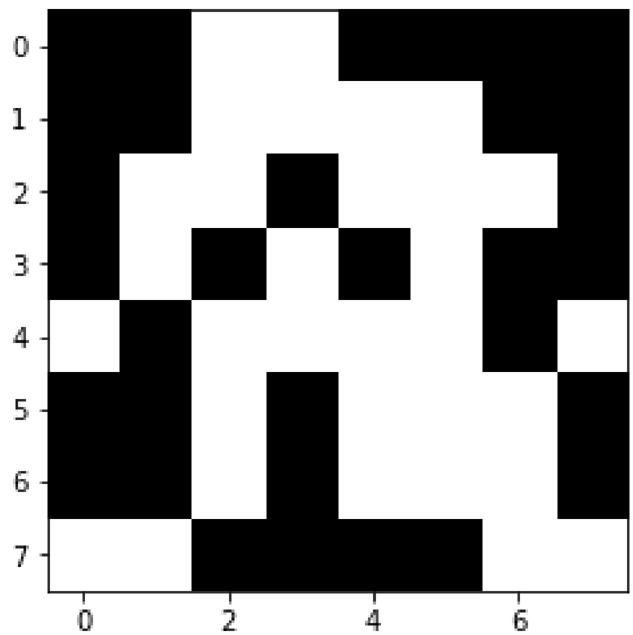
In this case test patterns are obtained by flipping elements with probability 0.4

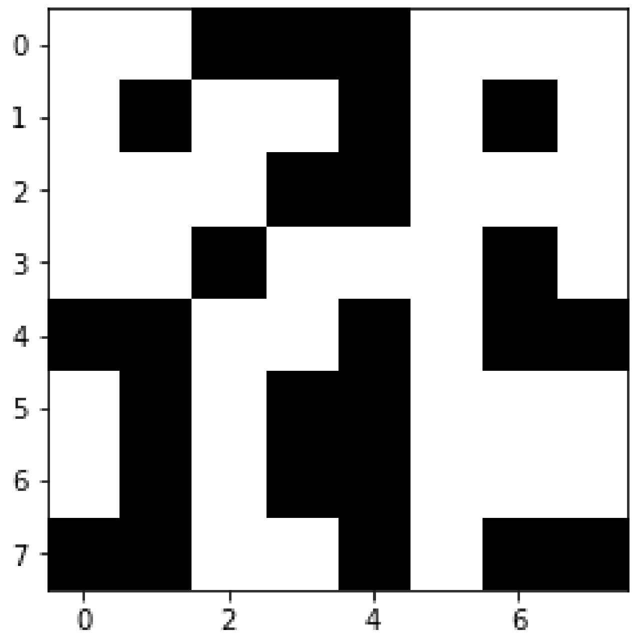
In [32]:
```
import random
for i in range(64):
    while random.random()<0.4:
        bipolar_0[i]=-bipolar_0[i]
        bipolar_1[i]=-bipolar_1[i]
        bipolar_2[i]=-bipolar_2[i]
        bipolar_3[i]=-bipolar_3[i]
```

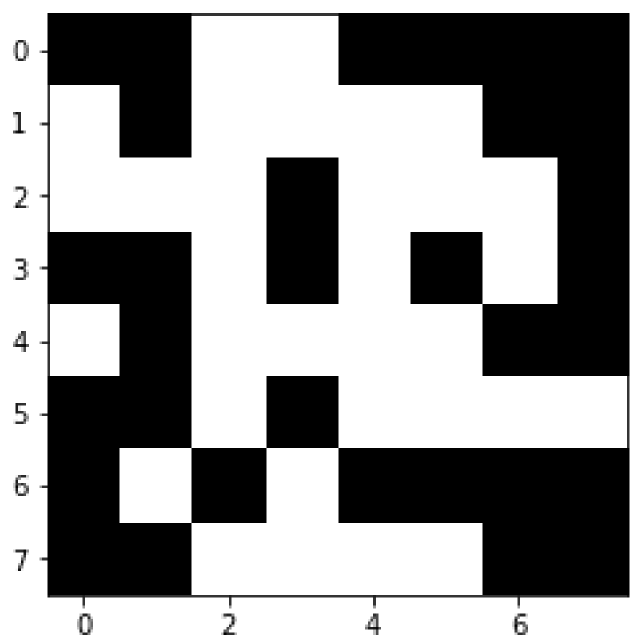
In [33]:
```
a,b,c,d=bipolar_0,bipolar_1,bipolar_2,bipolar_3
```

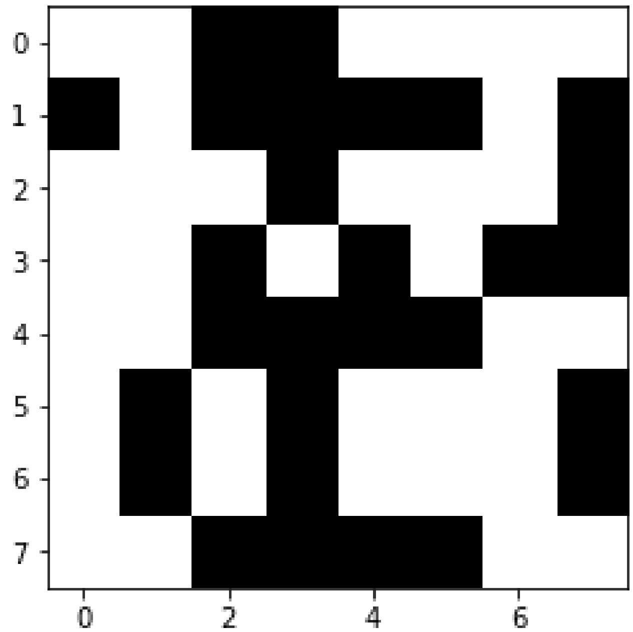
In [34]:
```
test_patternsw= np.array([a,b,c,d])
```

In [35]:
```
for i in [a,b,c,d]:
    visualize(i)
```

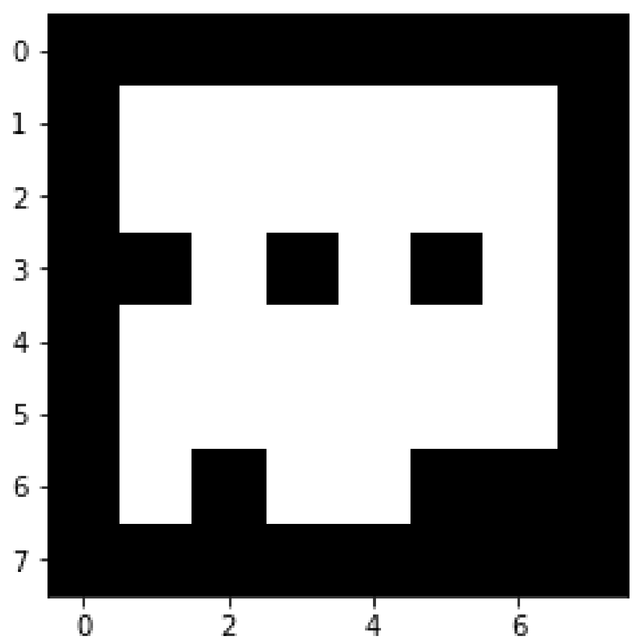


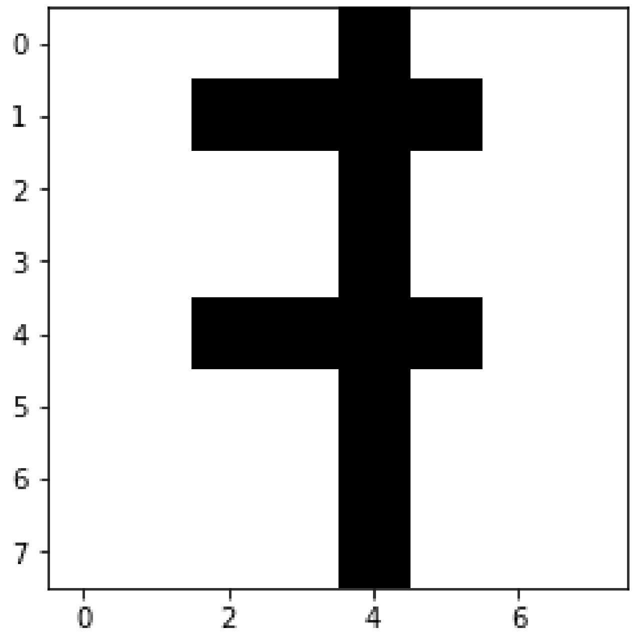

0
1
2
3
4
5
6
7
0
2
4
6

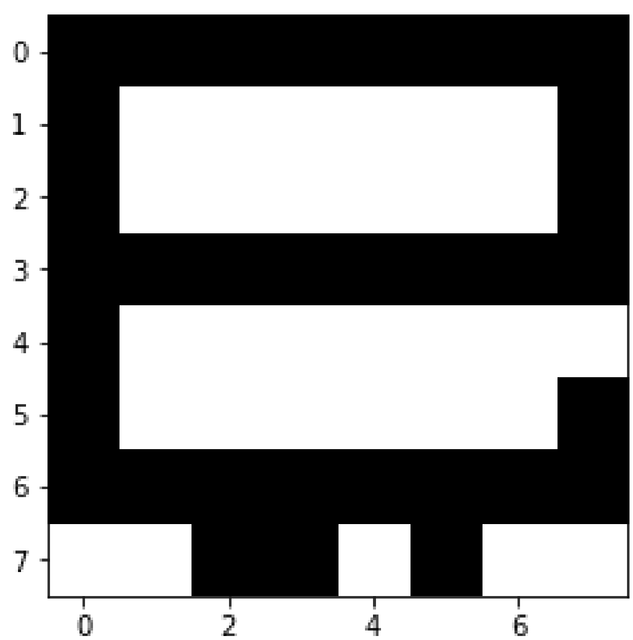

0
1
2
3
4
5
6
7
0
2
4
6

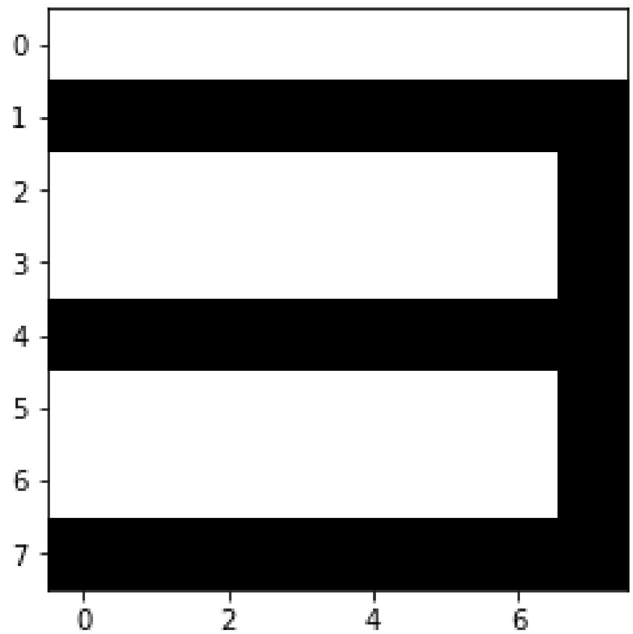



```
In [36]: r1=classify(T, test_patternsw, 1)
         r2=classify(T, test_patternsw, 2)

In [37]: for i in range(4):
             visualize(r1[i])
```

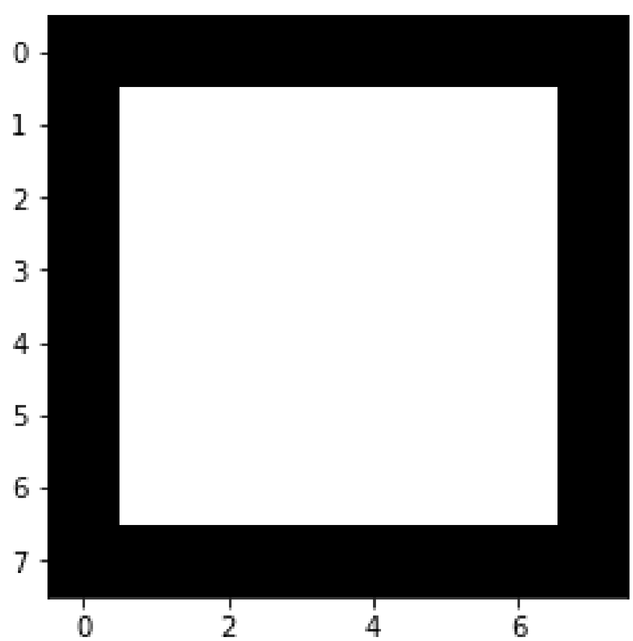


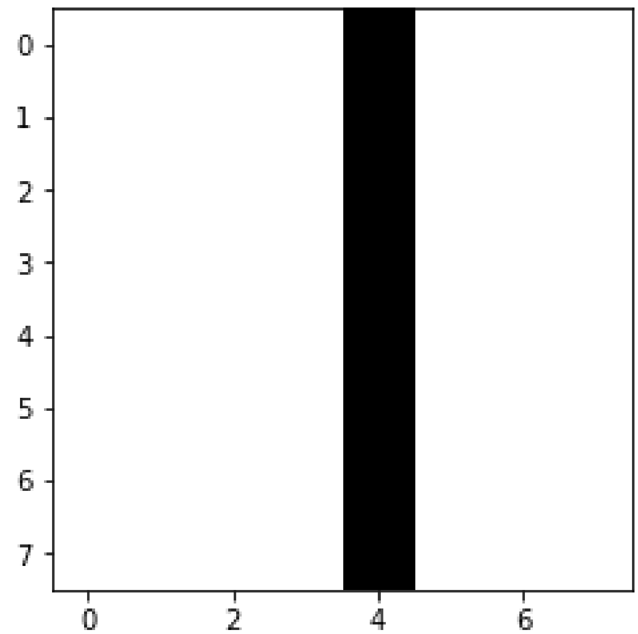

0
1
2
3
4
5
6
7
0
2
4
6

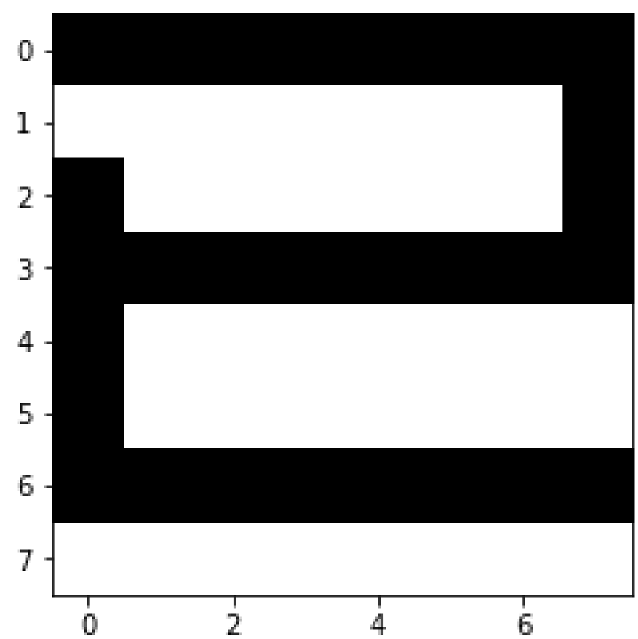

0
1
2
3
4
5
6
7
0
2
4
6

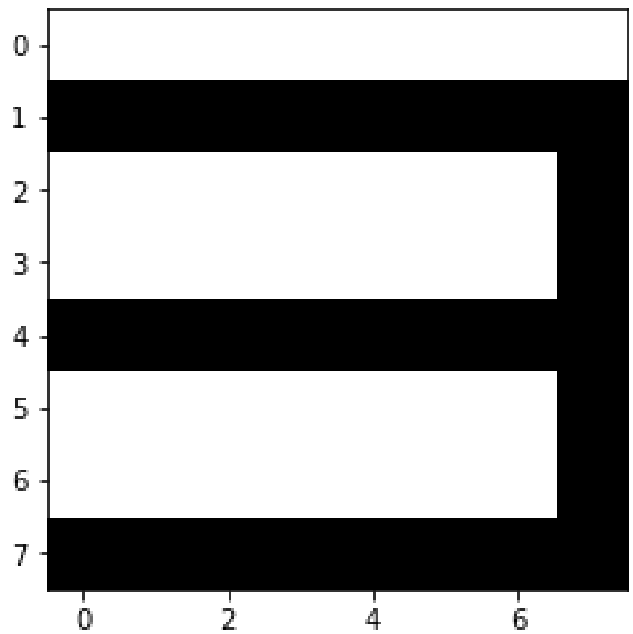



```
In [38]: for i in range(4):
             visualize(r2[i])
```

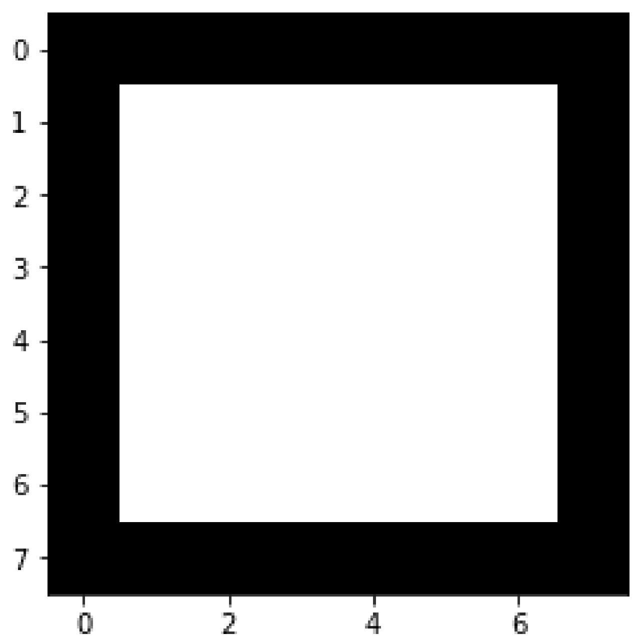


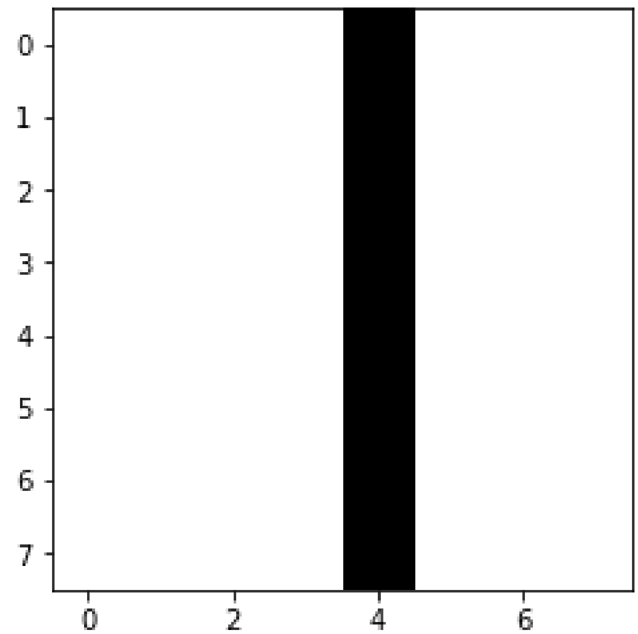

0
1
2
3
4
5
6
7
0
2
4
6

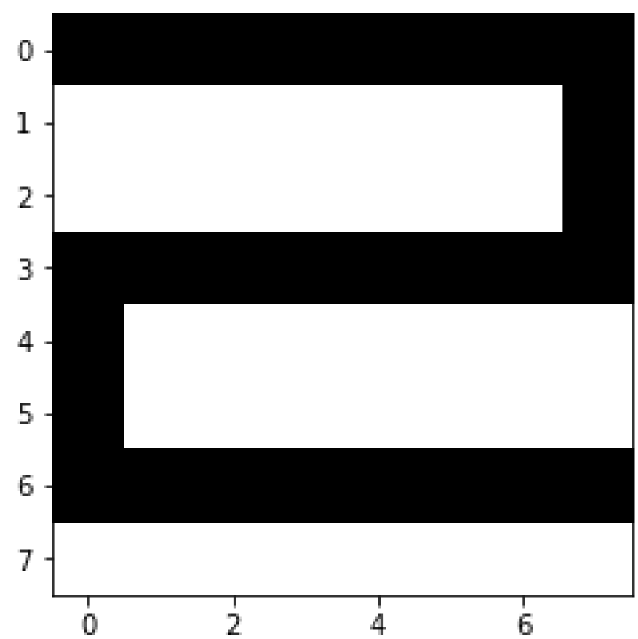

0
1
2
3
4
5
6
7
0
2
4
6

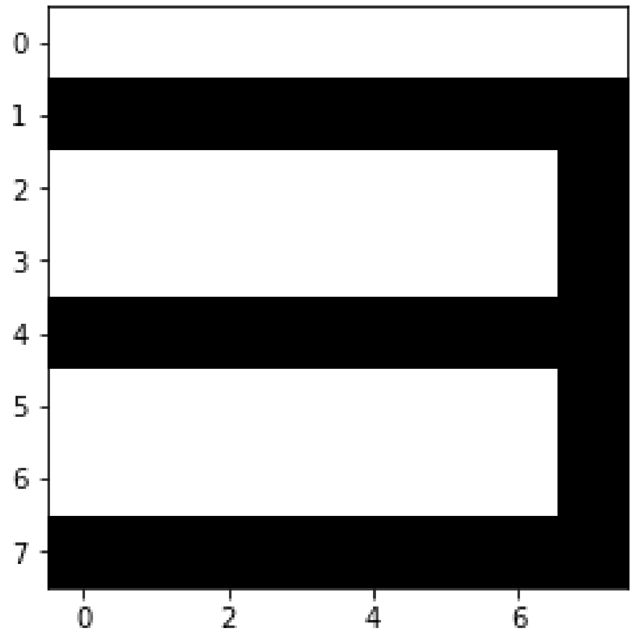



```
In [39]: r3=classify(T, test_patternsw, 3)

In [40]: for i in range(4):
             visualize(r3[i])
```




0
1
2
3
4
5
6
7
0
2
4
6


0
1
2
3
4
5
6
7
0
2
4
6



In the 3rd case all patterns converged to desired patterns.