Digit Recognition

Eric Blohm

September 9, 2024

Here is the Python code for the digit recognition problem:

```
1  import numpy as np
2
3  #need input matrices as vectors => flatten. x has
       shape (5, 10, 16)
4  def init_vectors(x):
5      arr = x
6      for i in range(0,len(x)):
7          x[i] = np.array(x[i])
8          x[i] = x[i].flatten()
9          arr[i] = x[i]
10     return arr
11
12
13 #From the formula, eq 2.26
14 def init_weight_matrix(x):
15     N = len(x[0])
16     W = np.zeros((N,N))
17     for i in range(0, len(W)):
18         for j in range(0, len(W[0])):
19             if i == j:
20                 W[i][j] = 0
21                 continue
22             sum_outer = 0
23             for pattern in x:
24                 sum_outer += pattern[i]*pattern[j]
25             W[i][j] = (1/N)*sum_outer
26     return W
27
28
29 def classify(patterns, x_transformed):
30     shape = (len(x_transformed),len(x_transformed[0]))
31     for i in range(0,len(patterns)):
32         patterns[i] = np.reshape(patterns[i],shape)
```

```
33          if np.array_equiv(patterns[i], x_transformed):
34              return i+1
35          elif np.array_equiv(-patterns[i],
                x_transformed):
36              return -(i+1)
37      return 6
38
39
40  def main():
41
42      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, 1, 1, 1, -1,
            -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1,
            1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[
            -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1,
            1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1,
            1, 1, 1, -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1,
            -1],[ -1, 1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1,
            1, 1, 1, -1, -1, 1, 1, 1, -1],[ -1, 1, 1, 1,
            -1, -1, 1, 1, 1, -1],[ -1, -1, 1, 1, 1, 1, 1,
            1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1,
            -1],[ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
            ];
43
44      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, -1, -1, 1, 1, 1,
            1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1,
            -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[
             -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1,
            -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1,
             1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1,
            -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1,
            -1],[ -1, -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1,
             -1, -1, 1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1,
            1, 1, 1, 1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1,
            1, -1, -1, -1],[ -1, -1, -1, 1, 1, 1, 1, -1,
            -1, -1] ];
45
46      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, -1, -1, 1, 1, 1,
            -1, -1],[ -1, -1, -1, -1, -1, 1, 1, 1, -1,
            -1],[ -1, -1, -1, -1, -1, 1, 1, 1, -1, -1],[
            -1, -1, -1, -1, -1, 1, 1, 1, -1, -1],[ 1, 1, 1,
```

```
         1, 1, 1, 1, 1, -1, -1],[ 1, 1, 1, 1, 1, 1, 1,
        1, -1, -1],[ 1, 1, 1, -1, -1, -1, -1, -1, -1,
        -1],[ 1, 1, 1, -1, -1, -1, -1, -1, -1, -1],[ 1,
         1, 1, -1, -1, -1, -1, -1, -1, -1],[ 1, 1, 1,
        -1, -1, -1, -1, -1, -1, -1],[ 1, 1, 1, -1, -1,
        -1, -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=[ [ -1, -1, 1, 1, 1, 1, 1, 1, -1, -1],[ -1, -1,
         1, 1, 1, 1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1,
         -1, 1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1, 1,
        1, 1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1,
        -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[
        -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1,
        1, 1, 1, 1, 1, 1, -1, -1],[ -1, -1, 1, 1, 1, 1,
         1, 1, -1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1,
        1, -1],[ -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[
         -1, -1, -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1,
         -1, -1, -1, -1, 1, 1, 1, -1],[ -1, -1, -1, -1,
         -1, -1, 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=[ [ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1,
        1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1, -1,
         -1, -1, -1, 1, 1, -1],[ -1, 1, 1, -1, -1, -1,
        -1, 1, 1, -1],[ -1, 1, 1, -1, -1, -1, -1, 1, 1,
         -1],[ -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[
        -1, 1, 1, -1, -1, -1, -1, 1, 1, -1],[ -1, 1, 1,
         1, 1, 1, 1, 1, 1, -1],[ -1, 1, 1, 1, 1, 1, 1,
        1, 1, -1],[ -1, -1, -1, -1, -1, -1, -1, 1, 1,
        -1],[ -1, -1, -1, -1, -1, -1, -1, 1, 1, -1],[
        -1, -1, -1, -1, -1, -1, -1, 1, 1, -1],[ -1, -1,
         -1, -1, -1, -1, -1, 1, 1, -1],[ -1, -1, -1,
        -1, -1, -1, -1, 1, 1, -1],[ -1, -1, -1, -1, -1,
         -1, -1, 1, 1, -1],[ -1, -1, -1, -1, -1, -1,
        -1, 1, 1, -1] ];

    x = [x1,x2,x3,x4,x5]

    # use later for classification
    patterns = x

    ######## Init ########
    shape = (len(x1),len(x1[0]))

```

```
    #list of all patterns, i.e x[0] is the first
        pattern
    x = init_vectors(x)

    W = init_weight_matrix(x)
    ####################


    ####### Feed pattern #######
    # Question 1A
    x_dist1 = [[-1, -1, 1, 1, 1, 1, 1, 1, -1, -1],
        [-1, -1, 1, 1, 1, 1, 1, 1, 1, -1], [-1, -1, -1,
         -1, -1, -1, 1, 1, 1, -1], [-1, -1, -1, -1, -1,
         -1, 1, 1, 1, -1], [-1, -1, -1, -1, -1, -1, 1,
        1, 1, -1], [-1, -1, -1, -1, -1, -1, 1, 1, 1,
        -1], [-1, -1, -1, -1, -1, -1, 1, 1, 1, -1],
        [-1, -1, 1, 1, 1, 1, 1, 1, -1, -1], [-1, -1, 1,
         1, 1, 1, 1, 1, -1, -1], [-1, -1, -1, -1, -1,
        -1, 1, 1, 1, -1], [-1, -1, -1, -1, -1, -1, 1,
        1, 1, -1], [-1, -1, -1, -1, -1, -1, 1, 1, 1,
        -1], [-1, -1, -1, -1, -1, -1, 1, 1, 1, -1],
        [-1, -1, -1, -1, -1, -1, 1, 1, 1, -1], [1, 1,
        -1, -1, -1, -1, -1, -1, -1, 1], [1, 1, -1, -1,
        -1, -1, -1, -1, 1, 1]]
    # Flatten
    x_dist1 = np.array(x_dist1)
    x_dist1 = x_dist1.flatten()

    # Question 2A
    x_dist2 = [[1, -1, -1, 1, -1, -1, -1, -1, 1, 1],
        [-1, -1, -1, -1, -1, 1, 1, 1, -1, -1], [-1, -1,
         1, -1, -1, -1, -1, -1, -1, -1], [1, 1, -1, -1,
         1, -1, -1, -1, -1, 1], [1, -1, -1, 1, -1, 1,
        1, -1, -1, -1], [-1, 1, -1, -1, 1, -1, -1, -1,
        -1, -1], [1, 1, -1, -1, -1, -1, -1, 1, -1, -1],
         [1, -1, 1, -1, 1, 1, 1, 1, -1, 1], [-1, -1,
        -1, -1, 1, -1, 1, -1, -1, 1], [1, -1, 1, -1,
        -1, -1, 1, -1, -1, -1], [-1, 1, -1, -1, -1, -1,
         -1, 1, -1, -1], [-1, -1, -1, 1, 1, 1, -1, -1,
        -1, 1], [-1, 1, -1, 1, -1, 1, 1, -1, -1, 1],
        [-1, -1, -1, 1, 1, -1, -1, 1, 1, 1], [1, 1, -1,
         1, 1, -1, -1, -1, 1, 1], [-1, 1, 1, -1, -1,
        -1, -1, -1, 1, -1]]
    # Flatten
    x_dist2 = np.array(x_dist2)
    x_dist2 = x_dist2.flatten()
```

```
79
80      # Question 3A
81      x_dist3 = [[1, 1, 1, -1, -1, -1, -1, 1, 1, 1], [1,
             1, 1, -1, -1, -1, -1, 1, 1, 1], [1, 1, 1, -1,
            -1, -1, -1, 1, 1, 1], [1, 1, 1, -1, -1, -1, -1,
             1, 1, 1], [1, 1, 1, -1, -1, -1, -1, 1, 1, 1],
            [1, 1, 1, -1, -1, -1, -1, 1, 1, 1], [1, 1, 1,
            -1, -1, -1, -1, 1, 1, 1], [1, 1, 1, -1, -1, -1,
             -1, 1, 1, 1], [1, 1, 1, -1, -1, 1, 1, -1, -1,
            -1], [-1, -1, -1, 1, 1, 1, 1, -1, -1, -1], [-1,
             -1, -1, 1, 1, 1, 1, -1, -1, -1], [-1, -1, -1,
            1, 1, 1, 1, -1, -1, -1], [-1, -1, -1, 1, 1, 1,
            1, -1, -1, -1], [-1, -1, -1, 1, 1, 1, 1, -1,
            -1, -1], [-1, -1, -1, 1, 1, 1, 1, -1, -1, -1],
            [-1, -1, -1, 1, 1, 1, 1, -1, -1, -1]]
82      # Flatten
83      x_dist3 = np.array(x_dist3)
84      x_dist3 = x_dist3.flatten()
85
86      distorted_patterns = [x_dist1,x_dist2,x_dist3]
87      ############################
88
89
90      ####### Update States #######
91      for k in range(0,len(distorted_patterns)):
92          S = distorted_patterns[k]
93          #upper limit not specified.
94          for t in range(1,5):
95              #rows in W
96              for i in range(0,len(W)):
97                  b = 0
98                  #Columns of s
99                  for j in range(0,len(S)):
100                     b += W[i][j]*S[j]
101                 if b != 0:
102                     # i=m states that we only change
                            one neuron each iteration.
103                     S[i] = np.sign(b)
104                 else:
105                     S[i] = 1
106
107         #convert back to matrix (10 x 16)
108         S = np.reshape(S,shape)
109
110         print("Question", k+1,"A:")
111         # tolist so i can copy and paste from terminal
```

```
            to OpenTA
        print(S.tolist(),"\n")

        # Compare the result to the stored patterns
            and return the digit if found, otherwise
            return 6.
        digit = classify(patterns,S)
        print("Question", k+1,"B:")
        print(digit)

        print("______________")


if __name__ == "__main__":
    main()
```