HW_7

March 22, 2020

[2]:

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

# Functions that might be useful (please read the documentation)
# x.flatten() (take a N-dimensional numpy array and make it one-dimensional)
# numpy.random.choice -- choose from the list of images
# numpy.dot -- compute the dot product
# numpy.random.normal -- set up random initial weights

DIM = (28,28) #these are the dimensions of the image

def load_image_files(n, path="Assignment7-images/images/"):
    # helper file to help load the images
    # returns a list of numpy vectors
    images = []
    for f in os.listdir(os.path.join(path,str(n))): # read files in the path
        p = os.path.join(path,str(n),f)
        if os.path.isfile(p):
            i = np.loadtxt(p)
            assert i.shape == DIM # just check the dimensions here
            # i is loaded as a matrix, but we are going to flatten it into a␣
 ↪single vector
            images.append(i.flatten())
    return images


# Load up these image files
A = load_image_files(0)
B = load_image_files(1)

N = len(A[0]) # the total size
assert N == DIM[0]*DIM[1] # just check our sizes to be sure

# set up some random initial weights
```

```
## Your code here:
```

1 Question 1

[90]:

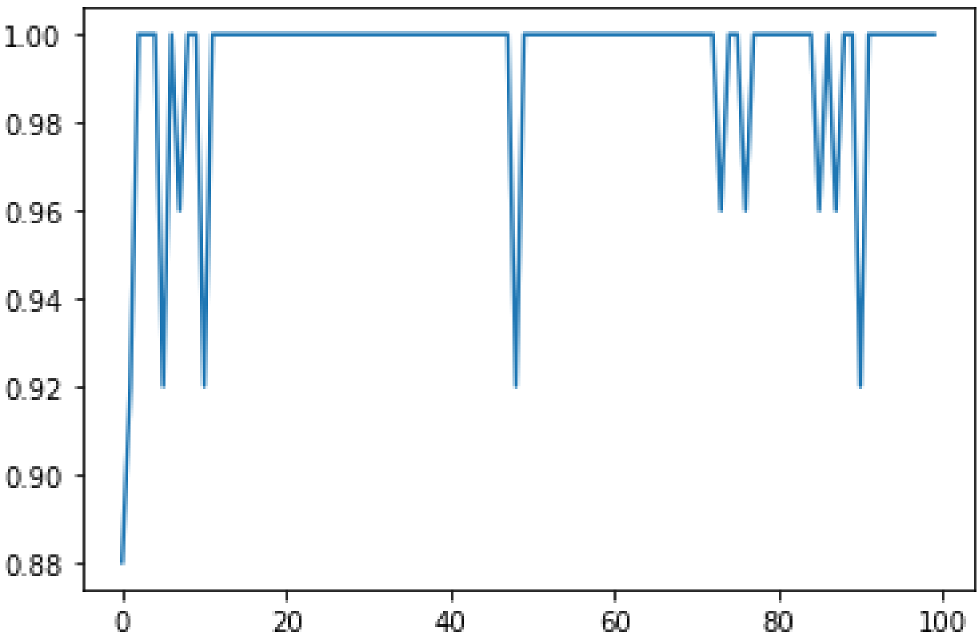
```
weights = np.random.normal(0,1,size=N)

def perceptron(image1, image2, w):
    vectors = image1 + image2
    indices = np.arange(0, len(vectors))
    average_accuracy = []
    for i in range(100):
        correct = 0
        images_seen = 0
        for i in range(25):
            choice = np.random.choice(indices, replace=False)
            current = vectors[choice]
            weighted_sum = np.dot(w, current)
            if weighted_sum >= 0:
                output = 1
            elif weighted_sum < 0:
                output = 0
            if output == 0 and choice >= len(image1):
                w += current
            elif output == 1 and choice < len(image1):
                w -= current
            else:
                correct += 1
        images_seen += 25
        accuracy = correct / images_seen
        average_accuracy.append(accuracy)
    return average_accuracy

plt.plot(perceptron(A, B, weights))
```

[90]:

```
[<matplotlib.lines.Line2D at 0x11dbd8b0>]
```



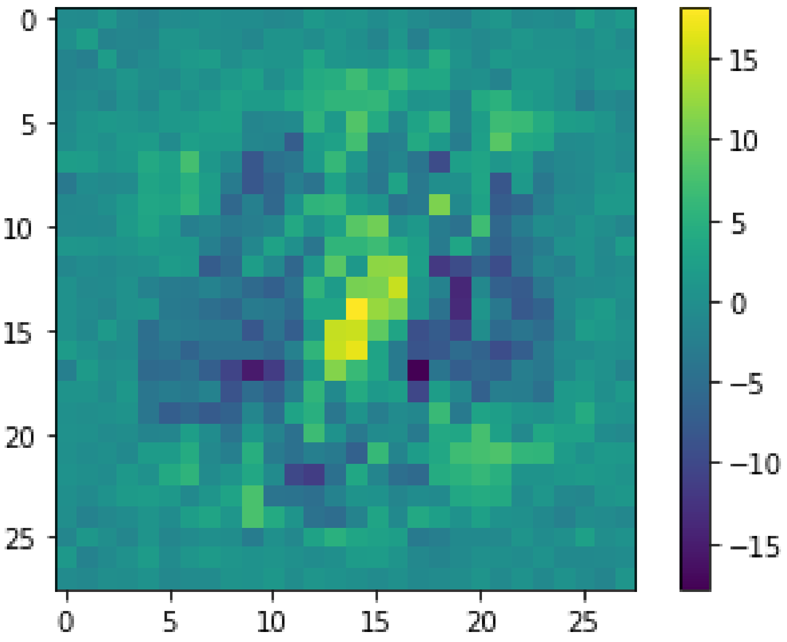
2 Question 2

The perceptron should not actually converge at 100% accuracy - intuitively, this makes sense because across “zero” and “one” data, the digits will share some feature space. Thus, “zero” and “one” do not have complete linear separability.

3 Question 3

[82]:
```
matrix = np.reshape(weights, (28, 28))
plt.imshow(matrix)
plt.colorbar()
```

[82]:
```
<matplotlib.colorbar.Colorbar at 0x11c70db0>
```



Looking at our weight plot, we can observe that the highest-value weights are clustered near the center of the graph while the lower value weights surround the center in a circular pattern - zero and near-zero weights make up the perimeter of the plot. Our perceptron will decrease weights whenever it detects a false positive and it will increase weights with a false negative. Based on these operations, weights with low and high values were repeatedly decreased/increased through training as these were considered the most “consequential” features spaces. Weights remained near zero in the periphery because this feature space was not consequential to digit discrimination.

4 Question 4

[116]:

```
def weight_changer(w, values):
    counter = 0
    temp = np.sort(abs(w))
    x = np.arange(values)
    np.delete(temp, x)
    return temp

step = np.arange(10, 780, 10)

new_weights = np.random.normal(0, 1, size=N)
accuracies = []
for i in step:
    new = perceptron(A, B, weight_changer(new_weights, i))
    accuracies.append(new)
```

5 Question 5

[117]:

```
zero = A
one = B
two = load_image_files(2)
three = load_image_files(3)
four = load_image_files(4)
five = load_image_files(5)
six = load_image_files(6)
seven = load_image_files(7)
eight = load_image_files(8)
nine = load_image_files(9)

x = [one, two, three, four, five, six, seven, eight, nine]
accuracies = []
```

[118]:

```
for i in x:
    for j in x:
        new_weights = np.random.normal(0,1,size=N)
        accuracies.append(perceptron(i, j, new_weights))
```

[120]:

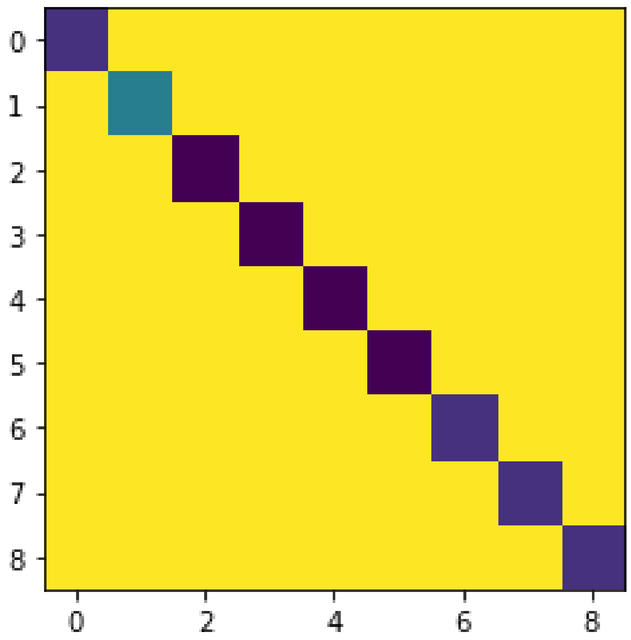
```
new_matrix = []
for i in accuracies:
    new_matrix.append(max(i))
```

[124]:

```
x = np.reshape(new_matrix, (9,9))
plt.imshow(x)
```

[124]:

```
<matplotlib.image.AxesImage at 0x25e24eb0>
```


0
1
2
3
4
5
6
7
8
0
2
4
6
8

```
[ ]:
```