KARIM_MANISHA___MINI_PROJECT_2

In [1]:
```
from scipy.io import loadmat
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image as im

from sklearn.model_selection import train_test_split
```

In [2]:
```
data = loadmat("C:\Class\Data Mining and Machine Learning\Project 2\Perceptron
```

In [3]:
```
data.keys()
```

Out[3]:
```
dict_keys(['__header__', '__version__', '__globals__', 'trainlabels', 'testla
bels', 'train', 'test'])
```

In [4]:
```
train = (data['train'])
train_labels = data['trainlabels']
test = (data['test'])
test_labels = data['testlabels']
```

In [5]:
```
np.unique(train_labels)
```

Out[5]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

In [6]:
```
train = train.T
train.shape
```

Out[6]:
```
(5000, 784)
```

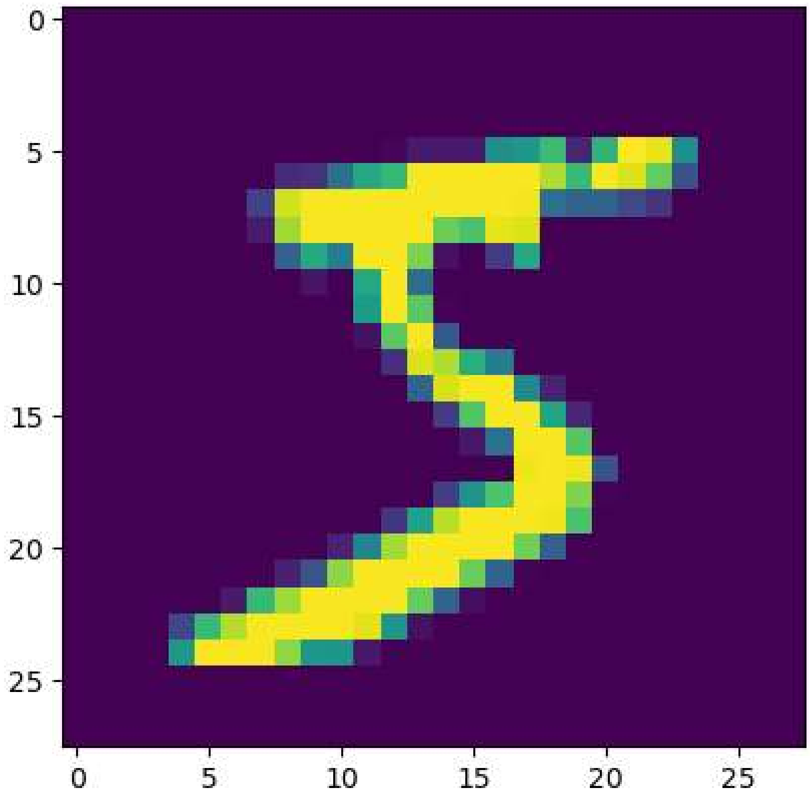
In [7]:
```
test = test.T
test.shape
```

Out[7]:
```
(1000, 784)
```

In [8]:
```
train = train/255
test = test/255
```

In [9]:
```
plt.imshow(train[0].reshape(28,28).T)
```

Out[9]: <matplotlib.image.AxesImage at 0x27127d85090>



Problem 1:

The activation function of perceptron is: y = sign(xT. w)

where, sign(x) = { 1 if x>0, -1 if x<0

The algorithm for updating weights in perceptron is:

1. Start iterations with w = 0
2. Dont change w if the prediction is correct
3. If the predicted value is -1 but the actual value is 1, w = w + w * learning_rate
4. If the predicted value is 1 but the actual value is -1, w = w - w * learning_rate

In [10]:
```
def change_labels(y, num):
    y = y.astype(int)
    for i in range(len(y)):
        if y[i] == num:
            y[i] = 1
        else:
            y[i] = -1
    return y
```

In [11]:
```
def init_params():
    w = np.zeros([1,784])
    return w

def func(v):
    if v>0:
        return 1
    else:
        return -1

def fit(train, train_labels, w, alpha):

    pred = []
    for i in range(len(train)):

        v = sum(np.dot(w, (train[i].T)))
        phi = func(v)
        pred = np.append(pred, phi)

        if phi == train_labels[i]:
            w=w

        elif ((phi == -1) & ( train_labels[i]== 1)):
            w = w + alpha * w


        else:
            w = w - alpha * w



    return (w , pred)

def get_accuracy(predictions, y):
    y = y.reshape(-1)
    return np.sum(predictions == y) / y.size
```

In [12]:
```
def gradient_descent(train, train_labels, alpha, iterations):

    w = init_params()

    for i in range(iterations):
        w, pred = fit(train, train_labels, w, alpha)


        if i % 100 == 0:
            print("Iteration: ", i)
            print("acc",get_accuracy(pred, train_labels))

    return w,pred
```

In [13]:
```
def test_pred(train, train_labels, w):

    pred = []
    for i in range(len(train)):

        v = sum(np.dot(w, (train[i].T)))
        phi = func(v)
        pred = np.append(pred, phi)

    return (pred)
```

Problem 1d

In [14]:
```
train_labels1 = change_labels(train_labels, 0)
test_labels1 = change_labels(test_labels, 0)
```

In [15]:
```
w, pred = gradient_descent(train, train_labels1, 0.01, 1000)
```

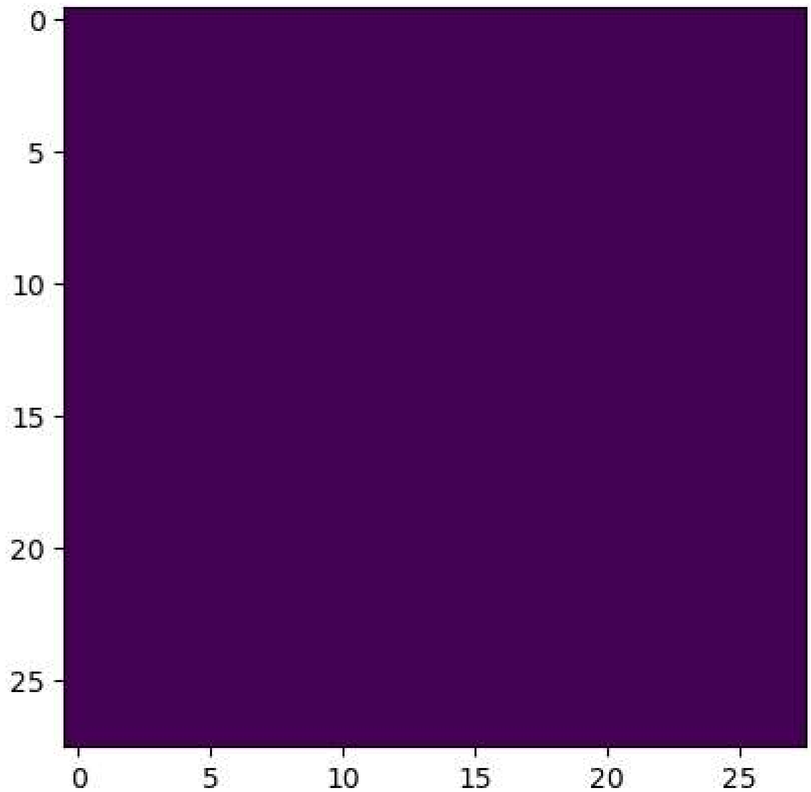
```
Iteration:  0
acc 0.9042
Iteration:  100
acc 0.9042
Iteration:  200
acc 0.9042
Iteration:  300
acc 0.9042
Iteration:  400
acc 0.9042
Iteration:  500
acc 0.9042
Iteration:  600
acc 0.9042
Iteration:  700
acc 0.9042
Iteration:  800
acc 0.9042
Iteration:  900
acc 0.9042
```

In [16]:
```
pred = test_pred(test, test_labels1, w)
print("acc",get_accuracy(pred, test_labels1))
```

```
acc 0.915
```

In [17]:
```
w  = w.T.reshape(28,28)
plt.imshow(w)
```

Out[17]: <matplotlib.image.AxesImage at 0x2712a363e90>



Problem 1e

In [18]:
```
train_labels1 = change_labels(train_labels, 8)
test_labels1 = change_labels(test_labels, 8)
```

In [19]:
```
w, pred = gradient_descent(train, train_labels1, 0.01, 1000)
```

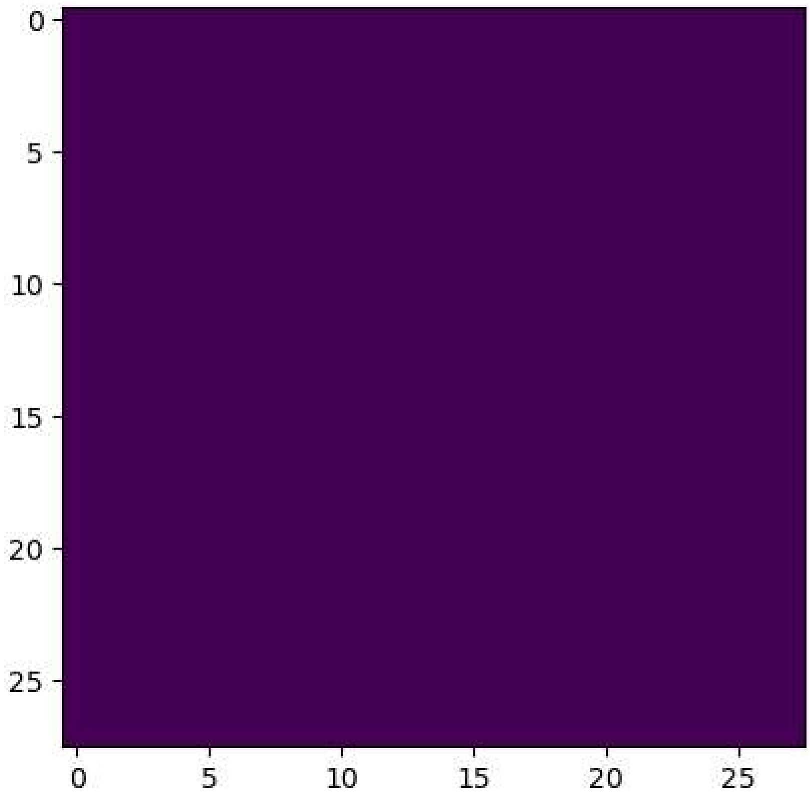
```
Iteration:  0
acc 0.9076
Iteration:  100
acc 0.9076
Iteration:  200
acc 0.9076
Iteration:  300
acc 0.9076
Iteration:  400
acc 0.9076
Iteration:  500
acc 0.9076
Iteration:  600
acc 0.9076
Iteration:  700
acc 0.9076
Iteration:  800
acc 0.9076
Iteration:  900
acc 0.9076
```

In [20]:
```
pred = test_pred(test, test_labels1, w)
print("acc",get_accuracy(pred, test_labels1))
```

```
acc 0.911
```

In [21]:
```
w  = w.T.reshape(28,28)
plt.imshow(w)
```

Out[21]: <matplotlib.image.AxesImage at 0x2712cc53490>



for 1

In [22]:
```
train_labels1 = change_labels(train_labels, 1)
test_labels1 = change_labels(test_labels, 1)
```

In [23]:
```
w, pred = gradient_descent(train, train_labels1, 0.01, 1000)
```

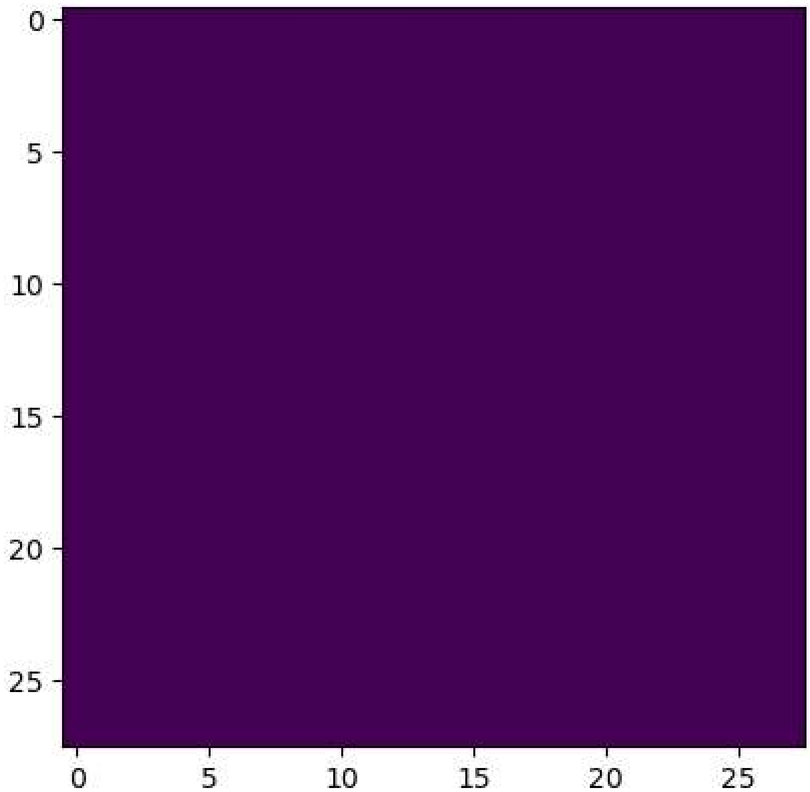
```
Iteration:  0
acc 0.8874
Iteration:  100
acc 0.8874
Iteration:  200
acc 0.8874
Iteration:  300
acc 0.8874
Iteration:  400
acc 0.8874
Iteration:  500
acc 0.8874
Iteration:  600
acc 0.8874
Iteration:  700
acc 0.8874
Iteration:  800
acc 0.8874
Iteration:  900
acc 0.8874
```

In [24]:
```
pred = test_pred(test, test_labels1, w)
print("acc",get_accuracy(pred, test_labels1))
```

```
acc 0.874
```

In [25]:
```
w  = w.T.reshape(28,28)
plt.imshow(w)
```

Out[25]: <matplotlib.image.AxesImage at 0x2712d56b390>



for 2

In [26]:
```
train_labels1 = change_labels(train_labels, 2)
test_labels1 = change_labels(test_labels, 2)
```

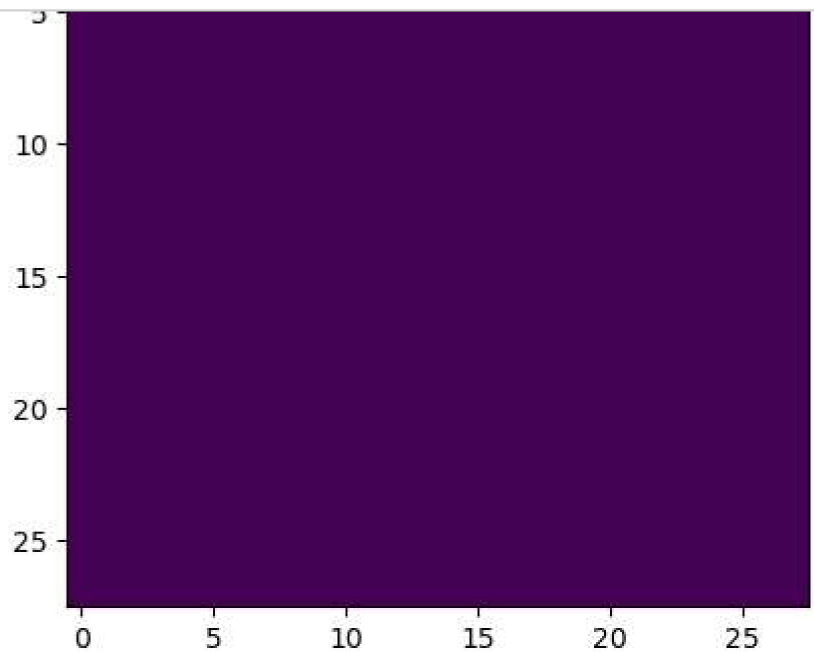
In [27]:
```
w, pred = gradient_descent(train, train_labels1, 0.01, 1000)
```

```
Iteration:  0
acc 0.9024
Iteration:  100
acc 0.9024
Iteration:  200
acc 0.9024
Iteration:  300
acc 0.9024
Iteration:  400
acc 0.9024
Iteration:  500
acc 0.9024
Iteration:  600
acc 0.9024
Iteration:  700
acc 0.9024
Iteration:  800
acc 0.9024
Iteration:  900
acc 0.9024
```

In [28]:
```
pred = test_pred(test, test_labels1, w)
print("acc",get_accuracy(pred, test_labels1))
```

```
acc 0.884
```

In [29]:
```
w  = w.T.reshape(28,28)
plt.imshow(w)
```



The image of w should represent the value the perceptron is trying to predict. Unfortunately, due to computational limitations this isnt the case. The perceptron doesnt converge and more training is required.

Problem 2

In [30]:
```
data = loadmat("C:\Class\Data Mining and Machine Learning\Project 2\Perceptron
```

In [31]:
```
data.keys()
```

Out[31]:
```
dict_keys(['__header__', '__version__', '__globals__', 'trainlabels', 'testla
bels', 'train', 'test'])
```

In [32]:
```
train = (data['train'])
train_labels = data['trainlabels']
test = (data['test'])
test_labels = data['testlabels']
```

In [33]:
```
np.unique(train_labels)
```

Out[33]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

In [34]:
```
train = train.T
```

In [35]:
```
#test = test
train_labels = train_labels.reshape(-1)
train_labels.shape
```

Out[35]:
```
(5000,)
```

In [36]:
```
train = train/255
test = test/255
```

In [37]:
```
m, n = train.shape
m, n
```

Out[37]:
```
(5000, 784)
```

Defined Sigmoid and Derivative of sigmoid

In [38]:
```
def sigmoid(x):
    return(1 /(1 + np.exp(-x)))

def one_hot(y):
    one_hot_y = np.zeros((y.size, y.max() + 1))
    one_hot_y[np.arange(y.size), y] = 1
    one_hot_y = one_hot_y.T
    return one_hot_y

def der_sigmoid(x):
    return (sigmoid(x) * (1 - sigmoid(x)))
```

The algorithm

In [39]:
```
def init_params():

    w1 = np.random.rand(25,784) - 0.5
    b1 = np.random.rand(25, 1) - 0.5

    w2 = np.random.rand(10, 25) - 0.5
    b2 = np.random.rand(10, 1) - 0.5


    return w1, b1, w2, b2


def forward_pass(w1, w2, b1, b2, x):

    v1 = np.dot(w1, x.T) + b1
    phi1 = sigmoid(v1)

    v2 = np.dot(w2, phi1) + b2
    phi2 = sigmoid(v2)

    return v1, phi1, v2, phi2

def backward_pass(v1, phi1, b1, w1, v2, phi2, b2, w2, x, y):

    one_hot_encoding_y = one_hot(y)


    dv2 = phi2 - one_hot_encoding_y
    dw2 = 1/m * np.dot(dv2, (phi1.T))
    db2 = 1/m * np.sum(dv2)


    dv1 = np.dot(w2.T, dv2) * der_sigmoid(v1)
    dw1 = 1/m * np.dot(dv1, (x))
    db1 = 1/m * np.sum(dv1)

    return dw1, db1, dw2, db2


def update_params(w1, b1, w2, b2, dw1, db1, dw2, db2, alpha):

    w2 = w2 - alpha * dw2
    b2 = b2 - alpha * db2

    w1 = w1 - alpha * dw1
    b1 = b1 - alpha * db1


    return w1, b1, w2, b2
```

In [40]:
```
def gradient_descent(x, y, alpha, iterations):

    w1, b1, w2, b2 = init_params()

    for i in range(iterations):
        v1, phi1, v2, phi2 = forward_pass(w1, w2, b1, b2, x)
        dw1, db1, dw2, db2 = backward_pass(v1, phi1, b1, w1, v2, phi2, b2, w2,
        w1, b1, w2, b2 = update_params(w1, b1, w2, b2, dw1, db1, dw2, db2, alp

        if i % 100 == 0:
            print("Iteration: ", i)
            predictions = get_predictions(phi2)
            print("acc",get_accuracy(predictions, y))
    return w1, b1, w2, b2
```

In [41]:
```
def get_predictions(phi2):
    pred = np.argmax(phi2, 0)
    return pred

def get_accuracy(predictions, y):

    print(np.sum(predictions == y))
    return np.sum(predictions == y) / y.size
```

In [42]:
```
w1, b1, w2, b2 = gradient_descent(train, train_labels, 0.2, 2500)
```

```
acc 0.9368
Iteration:  1900
4692
acc 0.9384
Iteration:  2000
4704
acc 0.9408
Iteration:  2100
4707
acc 0.9414
Iteration:  2200
4716
acc 0.9432
Iteration:  2300
4723
acc 0.9446
Iteration:  2400
4731
acc 0.9462
```

The accuracy of this model after training is 94.62

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: