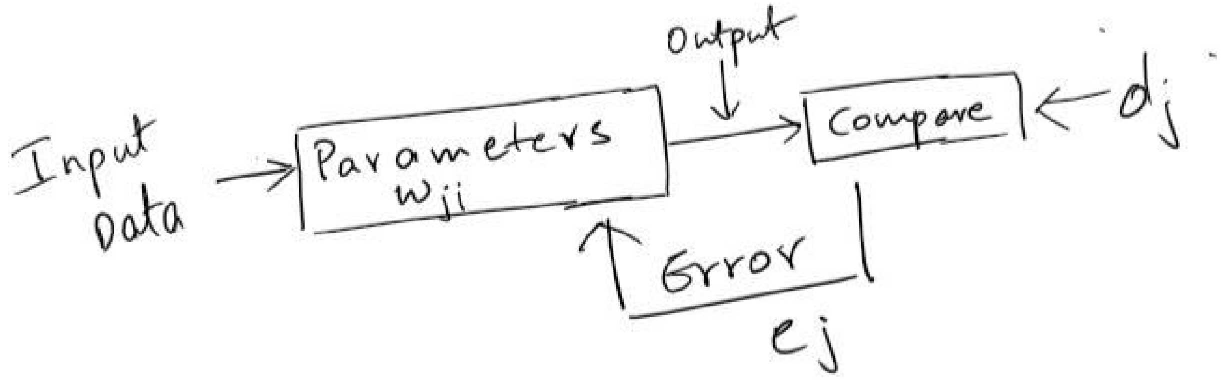
CIS 666 Artificial Intelligence

ASSIGNMENT 2

Submitted By

Bhuvana Chandra Atche

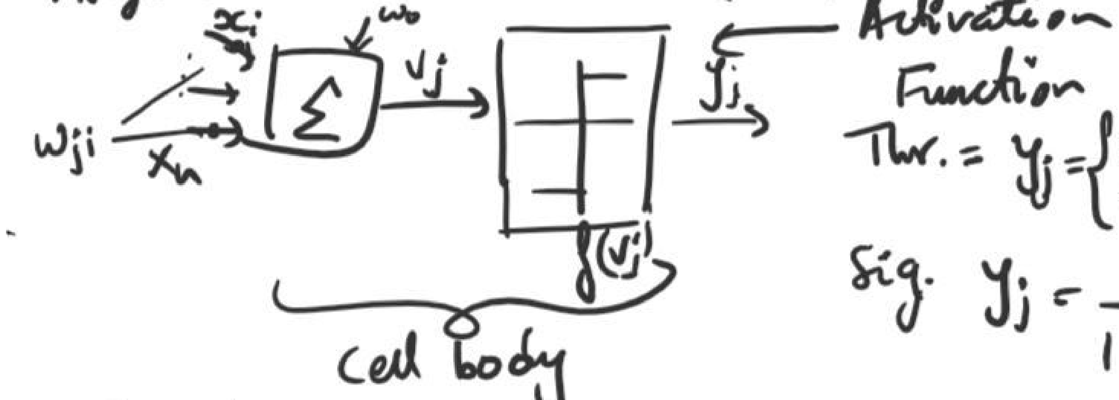
CSU ID 2754725



Steps:

1. Apply the input Xi
2. Compute output Yj
3. Compute error Ej
4. If ej>=E, modify wji => Ej = Dj - Yj if ej = 0, stop
5. Wji(t +1) = Wji(t) + change in (Wji(t))
6. Repeat steps 2 - 5 for all parameters
7. Repeat steps 2 - 6 till the error < = E(~0)

Artificial Neural Network (ANN)



Thr. = $y_j = \begin{cases} >1 \\ <0 \end{cases}$

Sig. $y_j = \frac{1}{1-e^{-v_j}}$

input $\Rightarrow x_i \cdot x_1, \ldots, x_N$

$$v_j = \sum_{i=1}^{N} x_j w_{ji} + w_0$$

$$y_j = f(v_j)$$

The Perceptron is inspired by the information processing of a single neural cell called a neuron. In a similar way, the Perceptron receives input signals from examples of training data that we weight and combined in a linear equation called the activation. The activation is then transformed into an output value or prediction using a transfer function, such as the step transfer function. In this way, the Perceptron is a classification algorithm for problems with two classes (0 and 1) where a linear equation (like or hyperplane) can be used to separate the two classes.

It is closely related to linear regression and logistic regression that make predictions in a similar way (e.g. a weighted sum of inputs).

The weights of the Perceptron algorithm must be estimated from your training data using stochastic gradient descent.

- **Learning Rate**: Used to limit the amount each weight is corrected each time it is updated.
- **Epochs**: The number of times to run through the training data while updating the weight.

These, along with the training data will be the arguments to the function.

There are 3 loops we need to perform in the function:

1. Loop over each epoch.
2. Loop over each row in the training data for an epoch.
3. Loop over each weight and update it for a row in an epoch.

As you can see, we update each weight for each row in the training data, each epoch.

Design a fully connected network structure of 784 input nodes and 10 output nodes. The input to the single layer network architecture will be a set of binary pixels representing a 28×28 image of handwritten digits. The output should indicate which of the digits (0,....,9) is in the input image.

```
  weights = np.random.uniform(0,1,(10,784))
```

Random weights are initialised based on the size of the input that is 28*28

1) Input data

```
f = gzip.open('mnist.pkl.gz', 'rb')
train_set, valid_set, test_set = p.load(f,encoding='iso-8859-1')
f.close()
```

This is used to load the input into train_set and

#Task 1: Repeat this experiment for different learning rate parameters
#(at least 3 experiments. Start with a large value and gradually decrease to a small value).

```
  for nr in range(0,10):
          for i in range(0,50000):
          x = images[i]
          t = targets[i]
          z = np.dot(weights[nr],x)
          output = activation(z)
          if nr == t:
          target = 1
          else:
          target = 0
          adjust = np.multiply((target - output) * learningRate[e], x)
```

2. Output is generated using the activation function

```
def activation(x):
          if x > 0:
               return 1
          return 0
```

3. Compute the error

```
          temp += 0.5*(target - output)*(target - output)
```

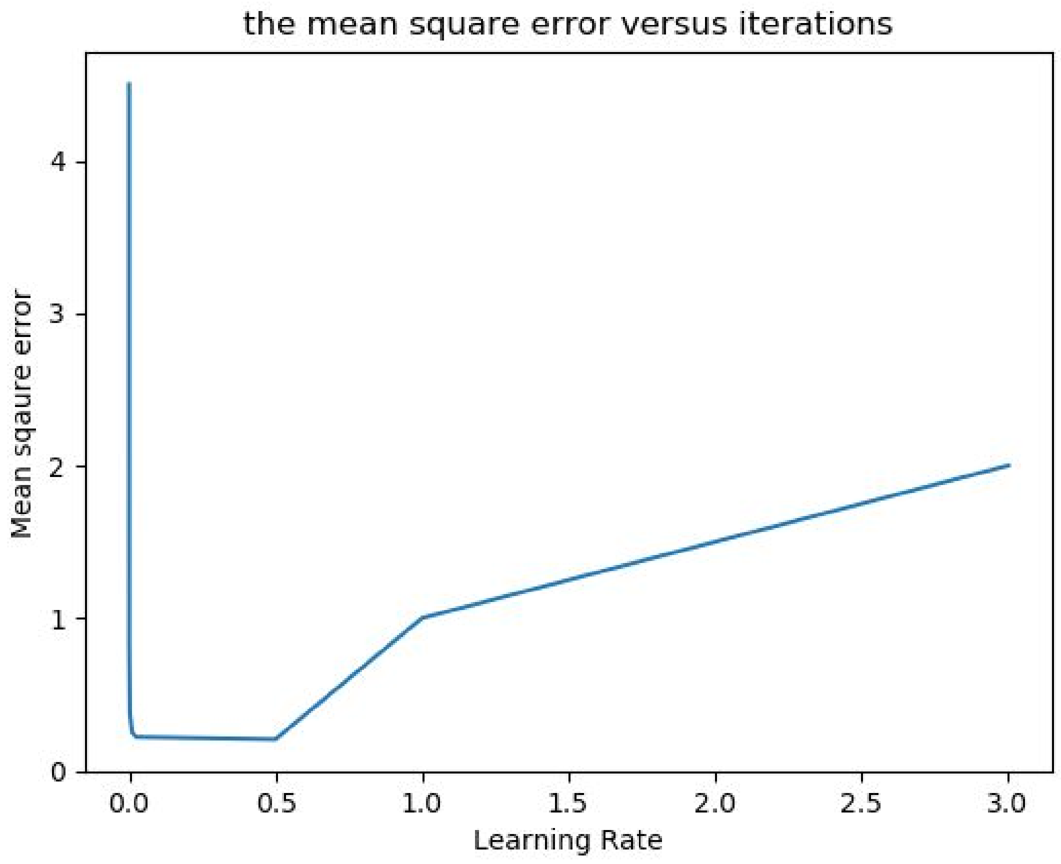
#target is desired output dj and output is yj applying 0.5(dj - yj)^2

$$E_j = \frac{1}{2}\left[d_j - y_j\right]^2$$

This the individual error and together makes it mean squared error which is marked against learning rate.

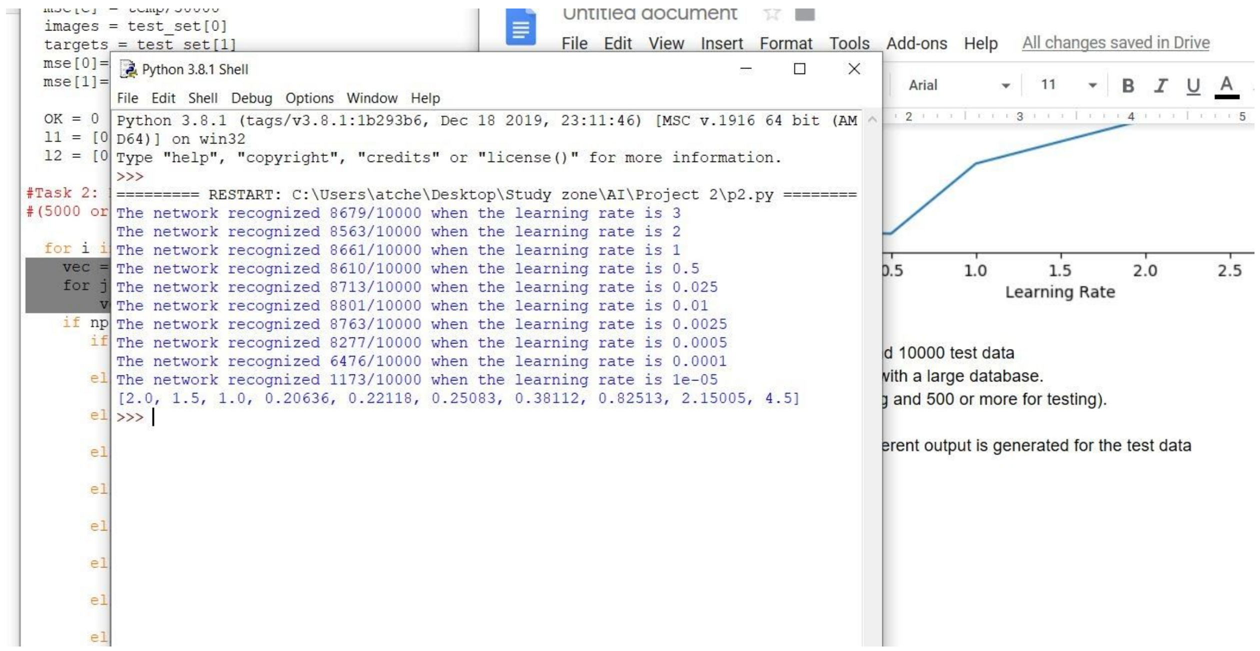
I used 10 different learning rate to better understand the scenario.

```
  learningRate = [3,2,1,0.5,0.025,0.01,0.0025,0.0005,0.0001,0.00001]
```



4.I used 50000 training and 10000 test data
#Task 2: Repeat Task #1 with a large database.
#(5000 or more for training and 500 or more for testing).

For each learning rate different output is generated for the test data



```
  images = test_set[0]
  targets = test_set[1]
  mse[0]=
  mse[1]=

  OK = 0
  l1 = [0
  l2 = [0

#Task 2:
#(5000 or

  for i i
    vec =
    for j
      v
    if np
      if
      el
```

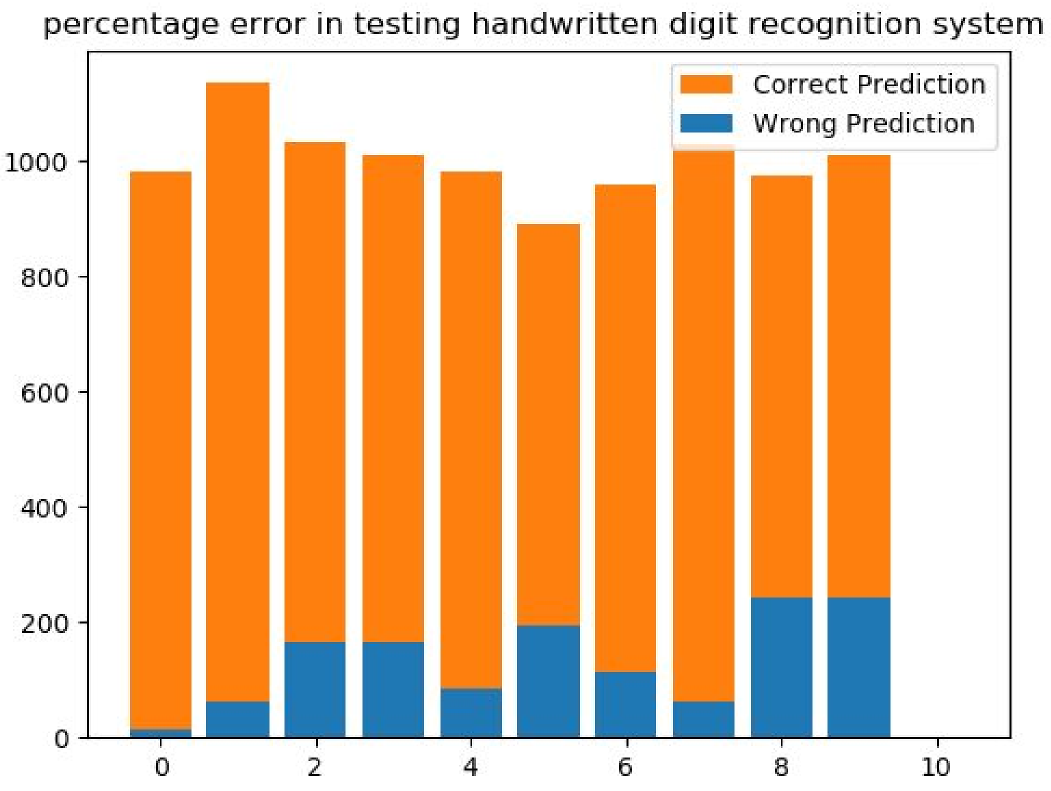
```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 23:11:46) [MSC v.1916 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========= RESTART: C:\Users\atche\Desktop\Study zone\AI\Project 2\p2.py ========
The network recognized 8679/10000 when the learning rate is 3
The network recognized 8563/10000 when the learning rate is 2
The network recognized 8661/10000 when the learning rate is 1
The network recognized 8610/10000 when the learning rate is 0.5
The network recognized 8713/10000 when the learning rate is 0.025
The network recognized 8801/10000 when the learning rate is 0.01
The network recognized 8763/10000 when the learning rate is 0.0025
The network recognized 8277/10000 when the learning rate is 0.0005
The network recognized 6476/10000 when the learning rate is 0.0001
The network recognized 1173/10000 when the learning rate is 1e-05
[2.0, 1.5, 1.0, 0.20636, 0.22118, 0.25083, 0.38112, 0.82513, 2.15005, 4.5]
>>>
```

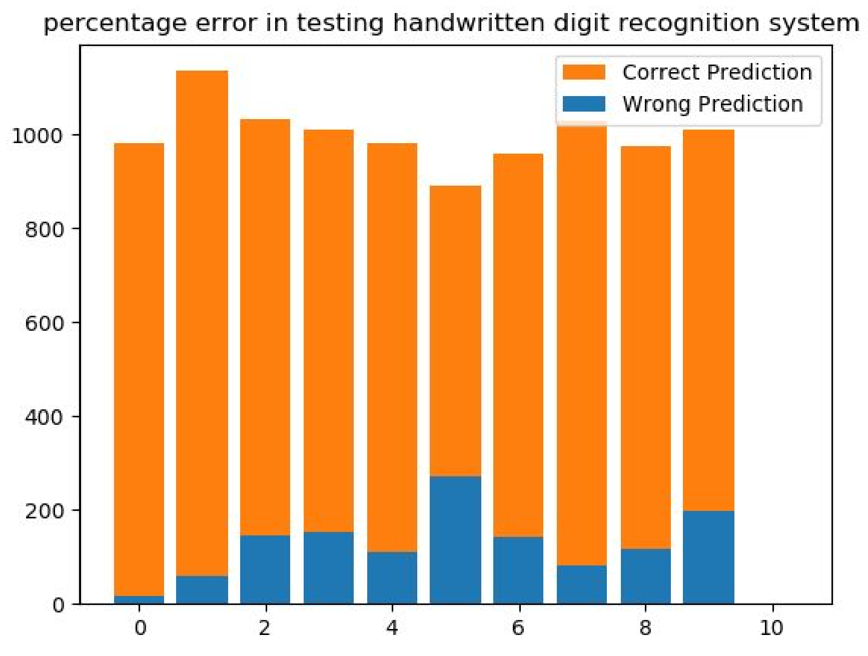
d 10000 test data
with a large database.
g and 500 or more for testing).

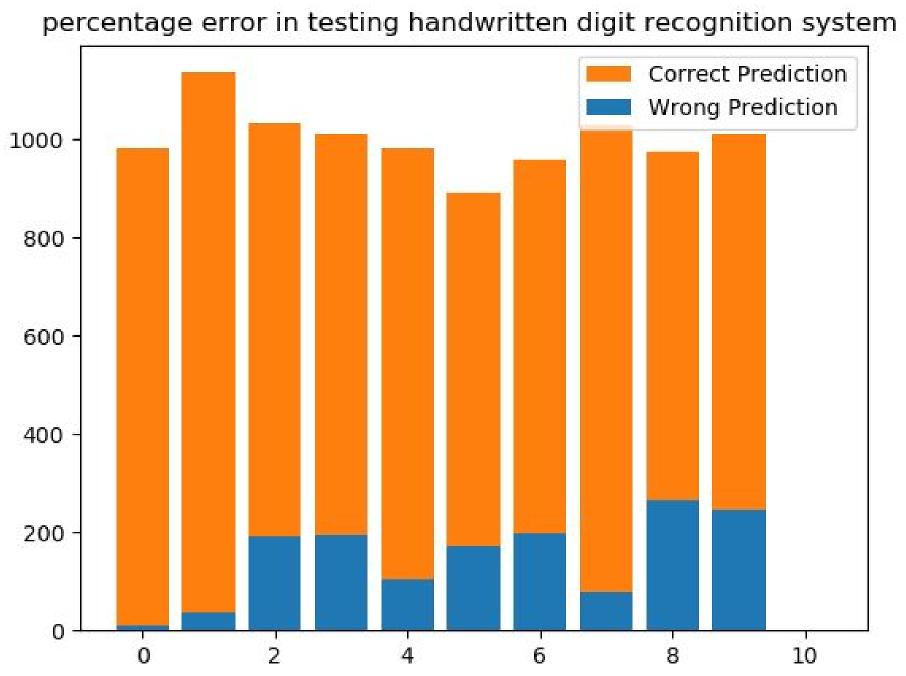
erent output is generated for the test data

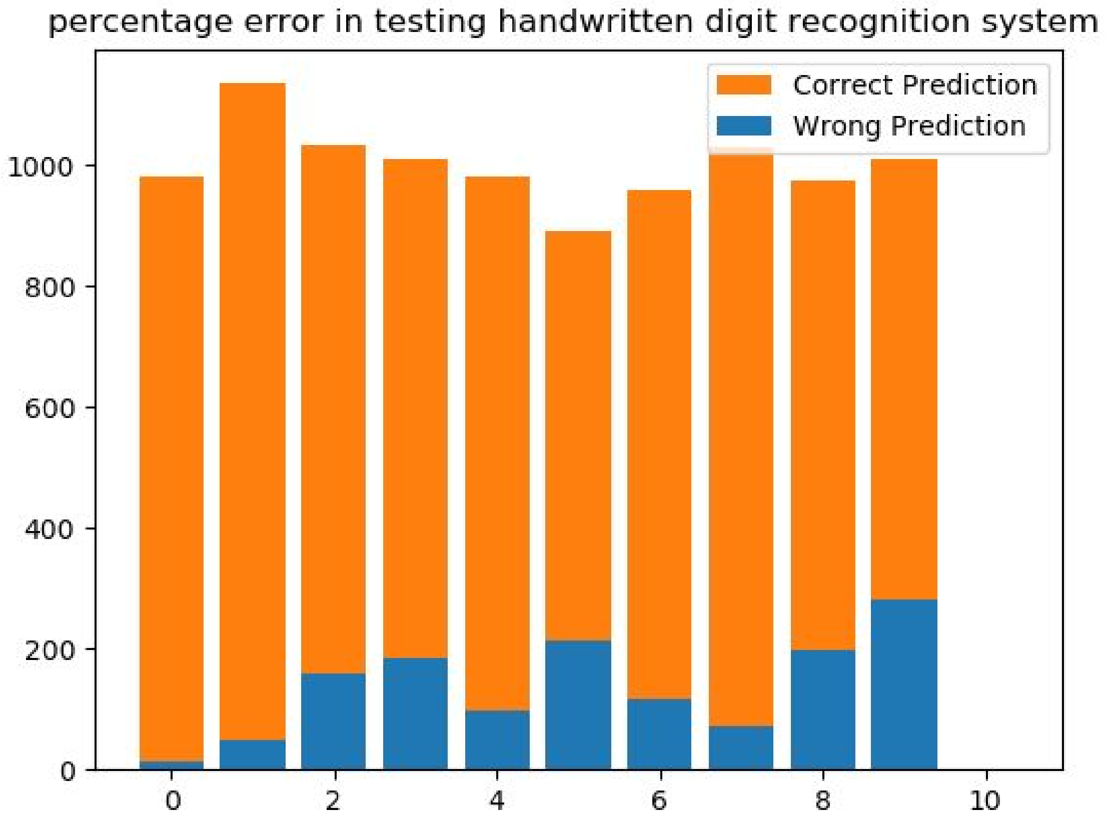
Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Untitled document
File Edit View Insert Format Tools Add-ons Help
All changes saved in Drive
Arial
11
0.5
1.0
1.5
2.0
2.5
Learning Rate

Plot the percentage error in testing your handwritten digit recognition system as a bar chart.


percentage error in testing handwritten digit recognition system
Correct Prediction
Wrong Prediction
0
200
400
600
800
1000
0
2
4
6
8
10


percentage error in testing handwritten digit recognition system
Correct Prediction
Wrong Prediction
0
200
400
600
800
1000
0
2
4
6
8
10


percentage error in testing handwritten digit recognition system
Correct Prediction
Wrong Prediction
0
200
400
600
800
1000
0
2
4
6
8
10


percentage error in testing handwritten digit recognition system
Correct Prediction
Wrong Prediction
0
200
400
600
800
1000
0
2
4
6
8
10