In [296…
```
#importing the required dependencies,numpy for  linear algebra,pandas for working with
import numpy as np
import pandas as pd
```

In [297…
```
data = pd.read_csv('MNIST_data/weight-height.csv') #load the MNIST data.
```

In [298…
```
data.head() #Getting an overview of how the dataframe looks.
```

Out[298]:

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

In [299…
```
data.tail()
```

Out[299]:

	Gender	Height	Weight
9995	Female	66.172652	136.777454
9996	Female	67.067155	170.867906
9997	Female	63.867992	128.475319
9998	Female	69.034243	163.852461
9999	Female	61.944246	113.649103

In [300…
```
data.shape
```

Out[300]:
```
(10000, 3)
```

In [301…
```
data.describe()
```

Out[301]:

	Height	Weight
count	10000.000000	10000.000000
mean	66.367560	161.440357
std	3.847528	32.108439
min	54.263133	64.700127
25%	63.505620	135.818051
50%	66.318070	161.212928
75%	69.174262	187.169525
max	78.998742	269.989699

In [302…
```
data.nunique()
```

Out[302]:
```
Gender        2
Height    10000
Weight    10000
dtype: int64
```

In [303…
```
data.dtypes
```

Out[303]:
```
Gender     object
Height    float64
Weight    float64
dtype: object
```

In [304…
```
data.isnull().sum()
```

Out[304]:
```
Gender    0
Height    0
Weight    0
dtype: int64
```

In [305…
```
data['Gender'].value_counts()
```

Out[305]:
```
Male      5000
Female    5000
Name: Gender, dtype: int64
```

In [306…
```
data.drop_duplicates(keep='first',inplace=True)
print(data.shape)
```

```
(10000, 3)
```

In [307…
```
data.columns
data['Gender'].replace(to_replace="Male",value=1,inplace=True)
data['Gender'].replace(to_replace="Female",value=0,inplace=True)
```

In [308…
```
data.head()
```

Out[308]:

	Gender	Height	Weight
0	1	73.847017	241.893563
1	1	68.781904	162.310473
2	1	74.110105	212.740856
3	1	71.730978	220.042470
4	1	69.881796	206.349801

In [309…
```
for col in ['Gender']:
    data[col] = data[col].astype('category')
for col in ['Height']:
    data[col] = data[col].astype('int')
for col in ['Weight']:
    data[col] = data[col].astype('int')
```

In [310…
```
data.dtypes
```

Out[310]:
```
Gender    category
Height       int32
Weight       int32
dtype: object
```

In [311…
```
data = np.array(data) #passsing the new numpyfied data to the new data variable.
```

In [312…
```
m,n=data.shape # mxn RowxCol.
```

In [313…
```
print(m,n) #m denoting the total data and n denoting the pixels per data.
```

```
10000 3
```

In [314…
```
data
```

Out[314]:
```
array([[  1,  73, 241],
       [  1,  68, 162],
       [  1,  74, 212],
       ...,
       [  0,  63, 128],
       [  0,  69, 163],
       [  0,  61, 113]], dtype=int64)
```

In [315…
```
data.T
```

Out[315]:
```
array([[  1,   1,   1, ...,   0,   0,   0],
       [ 73,  68,  74, ...,  63,  69,  61],
       [241, 162, 212, ..., 128, 163, 113]], dtype=int64)
```

In [320…
```
np.random.shuffle(data) # shuffle before splitting into dev and training sets.

#This is the Cross-Validation split
data_dev = data[0:1000].T #Transposing the data from row vector to coloumn vector.
Y_dev = data_dev[0] #Now this would be the first Row(the output/the label/the aim).
X_dev = data_dev[1:n] #Now this is the data part after seperating the label hence the
X_dev = X_dev / 255. #Normalizing the values.

#This is the Train Split
data_train = data[1000:m].T #Transposing the data from row vector to coloumn vector.
Y_train = data_train[0] #Now this would be the first Row(the output/the label/the aim)
X_train = data_train[1:n] #Now this is the data part after seperating the label hence
X_train = X_train / 255. #Normalization
_,m_train = X_train.shape
```

In [321…
```
print(f"The Y_Train data:{Y_train}")
print(f"The X_train data:{X_train[0]}") #Just checking the 0th pos as its a big array.
print(f"The X_train data shape:{X_train[:,0].shape}") #Checking the shape to see the t
```

```
The Y_Train data:[1 1 1 ... 1 1 0]
The X_train data:[0.25098039 0.27058824 0.2745098  ... 0.27843137 0.29411765 0.231372
55]
The X_train data shape:(2,)
```

In [322…
```
Y_train
```

Out[322]:
```
array([1, 1, 1, ..., 1, 1, 0], dtype=int64)
```

In [323…
```
a,b=X_train.shape
X_train.shape
```

Out[323]:
```
(2, 9000)
```

We need to run gradient descent

first we need to initialize some parameters.

In [424…
```
def init_params(): #1
    W1 = np.random.randn(10,2)
    W2 = np.random.randn(2,10)
    b1 = np.random.rand(10, 1) - 0.5 #(1,n_neurons)
    b2 = np.random.rand(1,1) - 0.5
    return W1, b1, W2, b2
```

In [425…
```
#Needed in the forward Propogation
def ReLU(Z): #3
    return np.maximum(Z, 0) #Just selecting the max function.
```

In [426…
```
def forward_prop(W1, b1, W2, b2, X): #2
    Z1 = W1.dot(X) + b1 #Z1 is the non-active set of neurons.X is the input as its the
    A1 = ReLU(Z1) #A1 is the activation function to pass the values
    Z2 = W2.dot(A1) + b2 #Z2 its the second layer,which takes the dot product from A1
    A2 = softmax(Z2) #A2 has softmax applied data for further classification.
    return Z1, A1, Z2, A2
```

In [427…
```
#Needed in the BackPropogation to undo the applied functions.
def ReLU_deriv(Z):#7
    return Z > 0
```

In [428…
```
#for backward propogation
def one_hot(Y): #6
    one_hot_Y = np.zeros((Y.size, Y.max() + 1)) #To create an empty matrix(filled with
    one_hot_Y[np.arange(Y.size), Y] = 1 #Setting the respective values as needed.
    one_hot_Y = one_hot_Y.T #Transposing the matrix
    return one_hot_Y
```

In [429…
```
one_hot(Y_train).shape
```

Out[429]:
```
(2, 9000)
```

In [430…
```
def backward_prop(Z1, A1, Z2, A2, W1, W2, X, Y): #5
    one_hot_Y = one_hot(Y)
    m=Y.size
    dZ2 = A2 - one_hot_Y #To find the error.
    dW2 = 1 / m * dZ2.dot(A1.T)
    db2 = 1 / m * np.sum(dZ2)

    dZ1 = W2.T.dot(dZ2) * ReLU_deriv(Z1) #Revert changes.
    dW1 = 1 / m * dZ1.dot(X.T)
    db1 = 1 / m * np.sum(dZ1)

    return dW1, db1, dW2, db2
```

In [431…
```
def update_params(W1, b1, W2, b2, dW1, db1, dW2, db2, alpha): #Alpha would act as the
    W1 = W1 - alpha * dW1
    b1 = b1 - alpha * db1
```

```
    W2 = W2 - alpha * dW2
    b2 = b2 - alpha * db2
    return W1, b1, W2, b2
```

In [432…

```
def get_predictions(A2):
    return np.argmax(A2, 0) #on the Xaxis
```

In [433…

```
def get_accuracy(predictions, Y):
    print(predictions, Y)
    return np.sum(predictions == Y) / Y.size
```

In [434…

```
#Needed in the forward Propogation
def softmax(Z): #4
    A = np.exp(Z) / sum(np.exp(Z))
    return A
```

In [435…

```
def gradient_descent(X, Y, alpha, iterations):

    W1, b1, W2, b2 = init_params()

    for i in range(iterations):
        Z1, A1, Z2, A2 = forward_prop(W1, b1, W2, b2, X)
        dW1, db1, dW2, db2 = backward_prop(Z1, A1, Z2, A2, W1, W2, X, Y)
        W1, b1, W2, b2 = update_params(W1, b1, W2, b2, dW1, db1, dW2, db2, alpha)
        if i % 10 == 0: #setting the denoter to occur every 10 steps.
            print("Iteration: ", i)
            predictions = get_predictions(A2)
            print(get_accuracy(predictions, Y))
    return W1, b1, W2, b2
```

In [436…

```
W1, b1, W2, b2 = gradient_descent(X_train, Y_train, 0.10, 1000)
```

```
Iteration:  0
[0 0 0 ... 0 0 1] [1 1 1 ... 1 1 0]
0.44355555555555554
Iteration:  10
[1 0 0 ... 0 0 1] [1 1 1 ... 1 1 0]
0.21144444444444443
Iteration:  20
[1 0 0 ... 1 0 1] [1 1 1 ... 1 1 0]
0.2837777777777778
Iteration:  30
[1 1 1 ... 1 1 1] [1 1 1 ... 1 1 0]
0.49377777777777776
Iteration:  40
[1 1 1 ... 1 1 1] [1 1 1 ... 1 1 0]
0.5003333333333333
Iteration:  50
[1 1 1 ... 1 1 1] [1 1 1 ... 1 1 0]
0.5004444444444445
Iteration:  60
[1 1 1 ... 1 1 1] [1 1 1 ... 1 1 0]
0.5004444444444445
Iteration:  70
[1 1 1 ... 1 1 1] [1 1 1 ... 1 1 0]
0.5004444444444445
Iteration:  80
[1 1 1 ... 1 1 1] [1 1 1 ... 1 1 0]
0.5004444444444445
Iteration:  90
[1 1 1 ... 1 1 1] [1 1 1 ... 1 1 0]
0.5007777777777778
Iteration:  100
[1 1 1 ... 1 1 1] [1 1 1 ... 1 1 0]
0.6061111111111112
Iteration:  110
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.7275555555555555
Iteration:  120
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.7985555555555556
Iteration:  130
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.8374444444444444
Iteration:  140
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.8563333333333333
Iteration:  150
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.872
Iteration:  160
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.8831111111111111
Iteration:  170
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.8912222222222222
Iteration:  180
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.8956666666666667
Iteration:  190
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.8984444444444445
```

```
Iteration:  200
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.901
Iteration:  210
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9025555555555556
Iteration:  220
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9022222222222223
Iteration:  230
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9023333333333333
Iteration:  240
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.903
Iteration:  250
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9033333333333333
Iteration:  260
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9035555555555556
Iteration:  270
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.904
Iteration:  280
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9048888888888889
Iteration:  290
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9047777777777778
Iteration:  300
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9052222222222223
Iteration:  310
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9043333333333333
Iteration:  320
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.904
Iteration:  330
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.904
Iteration:  340
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9042222222222223
Iteration:  350
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9052222222222223
Iteration:  360
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9052222222222223
Iteration:  370
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9064444444444445
Iteration:  380
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  390
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
```

```
Iteration:  400
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  410
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  420
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  430
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  440
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  450
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  460
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  470
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  480
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  490
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  500
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  510
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  520
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  530
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  540
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  550
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  560
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  570
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  580
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  590
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
```

```
Iteration:  600
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  610
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  620
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  630
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  640
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  650
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9065555555555556
Iteration:  660
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  670
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  680
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  690
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  700
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  710
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  720
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  730
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  740
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  750
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  760
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  770
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  780
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  790
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
```

```
Iteration:  800
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  810
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  820
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  830
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  840
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  850
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  860
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  870
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  880
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  890
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  900
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  910
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  920
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  930
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  940
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  950
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  960
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  970
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  980
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
Iteration:  990
[1 1 1 ... 1 1 0] [1 1 1 ... 1 1 0]
0.9066666666666666
```

In [415…
```
print("The Captured weights/biases are")
print(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>")
print(W1)
print(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>")
print(W2)
print(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>")
print(b1)
print(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>")
print(b1)
```

```
The Captured weights/biases are
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
[[-2.08018147e+00  2.78368727e+00]
 [-8.28537925e-01  1.33358461e+00]
 [-1.64822022e+00  1.64970194e-01]
 [-1.32944485e+00  2.89307960e+00]
 [ 9.93985660e-02 -4.31928448e-01]
 [ 2.06761683e+00  1.41199930e+00]
 [-1.42386639e+00 -9.58807350e-01]
 [-2.84497944e+00 -1.79512331e-03]
 [-7.46147099e-01 -1.22181580e-01]
 [ 3.10015986e-01 -1.07530219e+00]]
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
[[-1.04111534 -0.90039223 -0.92987912 -2.82152529 -0.15497261  1.54001792
  -0.72235807 -0.20060766 -0.00869056  0.36603957]
 [ 3.1203      0.84275131  0.94231517  2.97996307  1.29272501 -2.16776978
  -0.84246713 -1.22626663 -1.16748294  0.28198729]]
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
[[-0.85504312]
 [-0.4811348 ]
 [-0.77340091]
 [-1.07648855]
 [-0.31686537]
 [-0.33045071]
 [-0.97781793]
 [-0.42258508]
 [-1.13118015]
 [-0.78069208]]
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
[[-0.85504312]
 [-0.4811348 ]
 [-0.77340091]
 [-1.07648855]
 [-0.31686537]
 [-0.33045071]
 [-0.97781793]
 [-0.42258508]
 [-1.13118015]
 [-0.78069208]]
```

In [437…
```
def make_predictions(X, W1, b1, W2, b2):
    _, _, _, A2 = forward_prop(W1, b1, W2, b2, X)
    predictions = get_predictions(A2)
    return predictions
```

In [455…
```
def test_prediction(index, W1, b1, W2, b2):
    prediction = make_predictions(X_dev[:, index, None], W1, b1, W2, b2)
    label = Y_dev[index]
    #Setting up the test files.
```

```
    print("Prediction: ", prediction)
    print("Label: ", label)
```

In [457…
```
test_prediction(2, W1, b1, W2, b2)
```

```
Prediction:  [1]
Label:  1
```

In [458…
```
test_prediction(10, W1, b1, W2, b2)
```

```
Prediction:  [1]
Label:  1
```

In [459…
```
test_prediction(3, W1, b1, W2, b2)
```

```
Prediction:  [1]
Label:  1
```

In [460…
```
test_prediction(0, W1, b1, W2, b2)
```

```
Prediction:  [1]
Label:  0
```

In [472…
```
dev_predictions = make_predictions(X_dev, W1, b1, W2, b2)
acc=get_accuracy(dev_predictions, Y_dev)
print("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<")
print("The Neural Network has an accuracy of :",acc*100,"%")
```

```
[1 0 1 1 0 0 0 0 0 1 1 0 1 1 1 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 1 0 0 1
 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 0 0 1 0 0 1 1 0 0 1 1 0 1 0 0 1 0 1 1
 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 1 0 1 1 0 0 0 0 0 0 1 1 1
 1 1 1 0 1 0 0 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 0
 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 1 0 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 0 0 0
 0 1 0 0 1 1 0 1 0 0 0 1 0 0 1 0 1 1 0 1 0 1 1 1 0 1 0 0 1 1 1 1 1 1 0 0 0
 0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1 1
 1 1 0 0 1 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1 1 0 0 0 1
 0 1 0 1 0 1 1 0 1 0 0 0 1 0 0 1 0 1 1 1 1 0 0 1 1 0 0 0 1 1 0 1 1 0 0 0 1
 1 0 0 0 0 0 1 1 0 0 0 1 0 0 1 1 0 0 0 1 0 0 1 0 1 0 1 1 0 1 0 1 1 0 0 1 1
 0 0 0 0 0 1 0 1 1 0 1 1 1 1 0 0 1 0 1 0 1 0 0 1 1 0 0 0 1 1 0 0 1 0 1 1 0
 0 0 0 1 0 1 0 0 1 1 1 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 0 0 0 1 0 0 1 1
 0 0 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 1 0 1 1 1 1 1 1 1 0 0 1
 0 0 0 0 1 1 0 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 1 1
 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 0 0 0 1 1 0 0 1 1 1 1 0 0 0 1 0
 1 0 1 0 1 0 0 0 0 1 0 1 1 0 0 1 0 1 1 1 0 1 1 0 0 0 0 0 0 0 1 1 0 1 0 0
 1 1 0 0 1 0 1 1 1 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 0 1 1 1 0 1 1 0 1 1 0 0 0
 0 0 0 0 1 0 1 1 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 0 1 0 0
 1 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 1 0 0 0 1 0 1 0 0 1 1
 0 1 0 1 1 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 1 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0
 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0 0 0 0 1 0 1 0 1 1 1 0 1 0 1 0 1 0 1
 0 1 1 0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 1 0
 0 1 1 1 0 0 1 0 1 1 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 0
 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1 0
 1 1 0 0 0 0 0 0 0 1 1 0 1 0 1 0 1 0 0 1 0 0 1 1 1 1 1 0 0 0 0 1 0 0 1 0 1
 0 0 1 1 0 0 0 1 0 1 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 0 1 1 0 0 1 1
 1 1 0 1 0 0 0 1 1 1 1 1 1 1 1 0 1 0 1 1 1 0 0 0 0 1 1 0 1 1 0 1 0 1 0 1 0 1
 1] [0 0 1 1 0 0 0 0 0 1 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 1
 0 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 0 0 1
 1 1 0 1 0 0 0 0 0 1 1 1 1 1 0 0 1 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 0 1 1 0
 1 1 1 0 1 0 0 1 1 1 1 1 0 0 1 1 0 1 1 1 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0
 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 1 0 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 0 0 0
 0 1 0 0 1 1 0 1 0 0 0 1 0 0 1 0 1 1 0 1 0 1 1 0 0 1 0 1 1 1 1 1 1 1 0 0 0
 0 1 1 1 1 1 1 0 0 0 0 1 0 1 0 1 1 1 0 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1
 1 1 0 0 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1 0 0 1 0 1 1 1 0 0 0 1
 1 1 0 1 0 1 1 0 0 0 0 0 1 0 0 1 0 1 1 1 1 1 0 1 1 0 0 0 1 1 0 1 0 0 0 0 1
 1 0 0 0 0 1 1 1 0 0 0 1 0 0 1 1 0 1 0 1 0 0 1 0 1 0 1 1 0 1 0 1 1 0 1 1 1
 0 1 0 0 0 1 0 0 1 0 1 1 1 1 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0
 0 0 0 0 0 1 0 0 1 0 1 1 1 0 0 1 1 0 1 1 1 1 1 1 0 1 1 0 0 0 0 0 1 0 0 1 1
 0 0 1 0 1 1 1 1 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 1 1 1 1 1 1 1 1 0 1
 0 0 0 0 1 1 0 0 0 1 1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0 1 1 0 0 1 1 1 0 1 1
 1 0 1 0 0 1 1 0 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1 0
 0 1 1 0 1 0 0 0 0 1 0 1 1 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0
 1 1 0 0 1 0 1 1 1 0 1 1 1 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 1 0 1 1 1 1 1 0 0 0
 0 0 0 0 1 0 1 1 0 0 1 1 1 0 0 1 1 0 0 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 1 0 0
 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 1 0 0 0 1 0 1 0 0 0 1
 1 1 0 1 1 0 0 0 0 1 0 1 0 0 1 1 0 0 1 1 1 0 1 1 1 0 0 0 0 1 0 0 0 0 0 1 0 0
 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 1 1 0 1 0 1 0 1 0 1
 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 1 0
 0 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 0 0 0 1 1 0 1 0 0 0 1 0 1 1 1 0 1 1 1 0
 1 1 0 0 0 0 1 0 0 1 1 1 0 0 1 1 1 1 0 0 0 1 1 1 0 0 1 1 0 1 1 1 0 1 0 1 0
 1 1 0 0 0 0 0 0 0 1 1 0 1 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 1 0 1
 1 0 1 0 0 0 0 1 0 1 1 1 0 0 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1
 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 0 1 0 0 1 1 0 0 0 0 1 1 0 1 1 1 1 1 0 1 0 1 0 1
 1]
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
The Neural Network has an accuracy of : 90.5 %
```

In [ ]: