**MNIST Data**

**IMPORTING MODULES**

import numpy as np

np.random.seed(42)

import pandas as pd

**DEFINING PATH TO DATA AND READING**

path='/content/drive/MyDrive/digits-train.data'

train=pd.read\_csv(path, sep=',')

path='/content/drive/MyDrive/digits-test.data'

test=pd.read\_csv(path, sep=',')

y\_train=train.iloc[:,0]

y\_test=test.iloc[:,0]

X\_train1=train.iloc[:,1:]

X\_test1=test.iloc[:,1:]

print(X\_train1.shape)



def sigmoid(x):

    return 1 / (1 + np.exp(-x))

**COMPILING AND TRAINING**

# defining the model architecture

i\_neurons = X.shape[0]  # number of features in data set

h\_neurons = 10  # number of hidden layers neurons

o\_neurons = 1  # number of neurons at output layer

# initializing weight

wihidden = np.random.uniform(size=(i\_neurons, h\_neurons))

whoutput = np.random.uniform(size=(h\_neurons, o\_neurons))

# defining the parameters

Lr = 0.01

Epochs = 100

losses = []

for epoch in range(Epochs):

    # Forward Propogation

    # hidden layer activations

    hlinearTransform = np.dot(wihidden.T, X)

    hactivations = sigmoid(hlinearTransform)

    olinearTransform = np.dot(

        whoutput.T, hactivations

    )

    output = sigmoid(olinearTransform)

    # Backward Propagation

    # cal error

    err = np.square(y - output) / 2

    # cal rate of change of error w.r.t weight between hidden and output layer

    err\_wrt\_output = -(y - output)

    olt = np.multiply(output, (1 - output))

    oo1 = hactivations

    err\_wrt\_weights\_hidden\_output = np.dot(

        oo1,

        (err\_wrt\_output \* olt).T,

    )

    # calculating rate of change of error w.r.t weights between input and hidden layer

    olth = whoutput

    hhlt = np.multiply(

        hactivations, (1 - hactivations)

    )

    hlih = X

    err\_wrt\_weights\_input\_hidden = np.dot(

        hlih,

        (

            hhlt

            \* np.dot(

                olth,

                (olt \* err\_wrt\_output),

            )

        ).T,

    )

    # update weights

    whoutput = whoutput - Lr \* err\_wrt\_weights\_hidden\_output

    wihidden = wihidden - Lr \* err\_wrt\_weights\_input\_hidden

    # print error at every 10th epoch

    epoch\_loss = np.average(err)

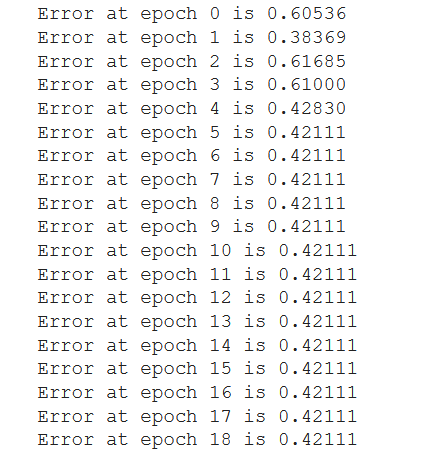
    if epoch % 1 == 0:

        print(f"Error at epoch {epoch} is {epoch\_loss:.5f}")

    # appending the error of each epoch

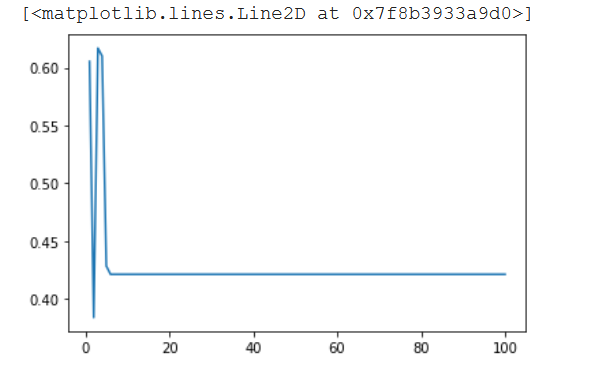
    losses.append(epoch\_loss)

**TRAINING RESULT**



import matplotlib.pyplot as plt

plt.plot(np.arange(1, Epochs + 1), np.array(losses))



**RESHAPING TESTING DATA**

X=X\_test1.T

y1=y\_test.values

y=y1.reshape((1,1796))

np.shape(y)

**TESTING**

hlinearTransform = np.dot(wihidden.T, X)

hactivations = sigmoid(hlinearTransform)

olinearTransform = np.dot(whoutput.T, hactivations)

output = sigmoid(olinearTransform)

err = np.average(np.square(y - output))

print(err)

0.9153674832348021

**Fish Data**

from sklearn import preprocessing

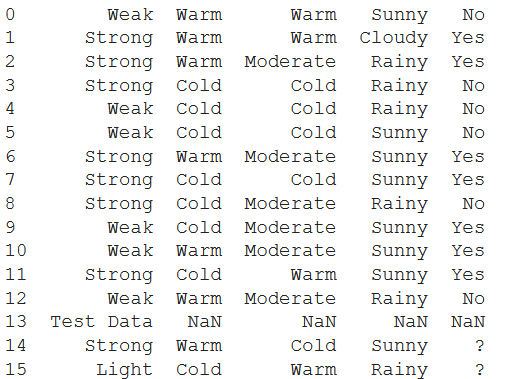
import pandas as pd

from keras.utils import np\_utils

path='/content/drive/MyDrive/fishing1.data'

data1=pd.read\_csv(path, sep=',')

print(data1)



As the data is in categorical form we converted it to numeric form using label encoder.

label\_encoder = preprocessing.LabelEncoder()

data=data1

y\_train=data.iloc[0:13,4]

X\_train1=data.iloc[0:13,0:4]

X\_train1['Strong'] = label\_encoder.fit\_transform(X\_train1['Strong'])

X\_train1['Warm'] = label\_encoder.fit\_transform(X\_train1['Warm'])

X\_train1['Warm.1'] = label\_encoder.fit\_transform(X\_train1['Warm.1'])

X\_train1['Sunny'] = label\_encoder.fit\_transform(X\_train1['Sunny'])

y\_train = label\_encoder.fit\_transform(y\_train)

X\_test1=data.iloc[14:17,0:4]

X\_test1['Strong'] = label\_encoder.fit\_transform(X\_test1['Strong'])

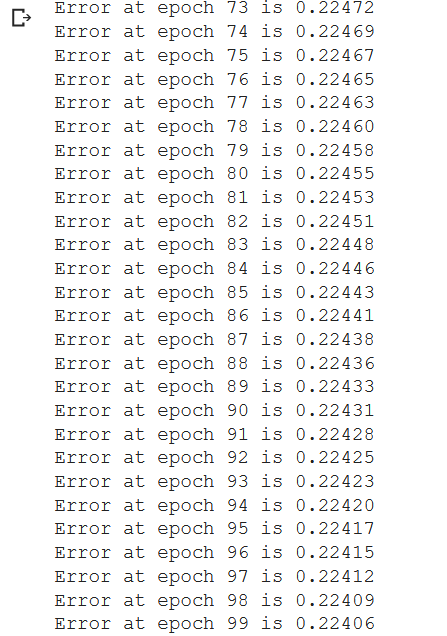
X\_test1['Warm'] = label\_encoder.fit\_transform(X\_test1['Warm'])

X\_test1['Warm.1'] = label\_encoder.fit\_transform(X\_test1['Warm.1'])

X\_test1['Sunny'] = label\_encoder.fit\_transform(X\_test1['Sunny'])

So with the help of label encoder we accessed each column one by one and converted it to numeric value for both training and testing data.

**TRAINING RESULT**



**RESHAPING TRAINING DATA**

X=X\_train1.T

y1=y\_train

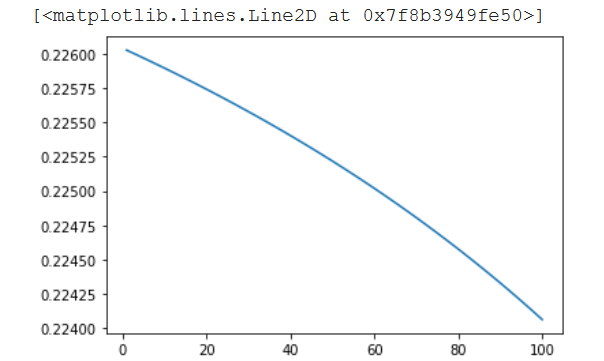
y=y1.reshape((1,13))

np.shape(y)

(1, 13)

import matplotlib.pyplot as plt

plt.plot(np.arange(1, Epochs + 1), np.array(losses))



**TESTING**

X = X\_test1.T

hlinearTransform = np.dot(wihidden.T, X)

hactivations = sigmoid(hlinearTransform)

olinearTransform = np.dot(whoutput.T, hactivations)

output = sigmoid(olinearTransform)

# err = np.average(np.square(y - output))

print(output)

[[0.99020522 0.97286386]]

Finally the predicted result for last two feature set has been shown. As the output is in categorical form so last two results will be ‘Yes’ and ‘Yes’.