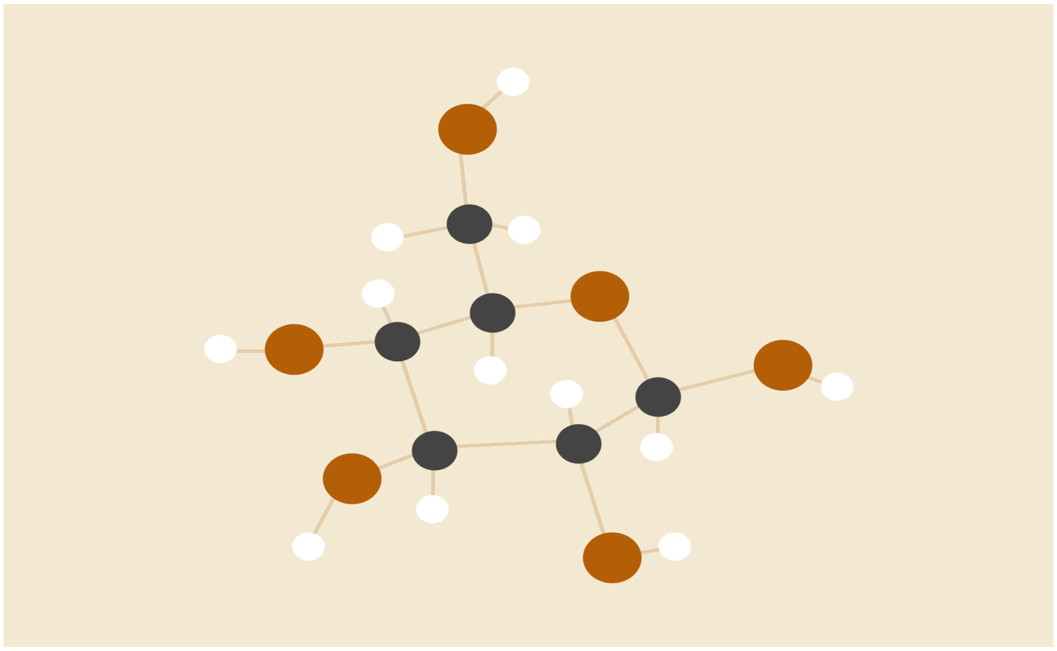
SOFT COMPUTING

CSE2009

Dr.Shruti Mishra

SCHOOL OF COMPUTER ENGINEERING



PROJECT

PREDICTION OF FLORA TYPE USING BACK PROPAGATION

BY

TERALA SRUJAN-18MIS7202
D.NISHANKH-18MIS7281

TABLE OF CONTENTS

PROBLEM STATEMENT

Plant Species knowledge is essential for protecting biodiversity. The identification of plants by traditional methods is hard, time consuming, and due to the use of specific terms identification of plant species is frustrating for non-experts. This creates a hard-to overcome hurdle for a new person interested in acquiring species knowledge. "
Biologists are facing difficulties to differentiate the flora type of plants based on their characteristics though there are several machine learning algorithms to predict their type there exists a minimal amount of error in prediction so we use back propagation to reduce the error rate and we have used the iris data set as our data for prediction considering the following attributes :

1)sepal length
2)sepal width
3)petal length
4)petal width
5)class: -   IrisSetosa
IrisVersicolour
IrisVirginica

OBJECTIVES

- Understand the dataset .
- Perform machine learning concepts on the iris data set and identify the accuracy score.
- Perform back propagation on the dataset and reduce error rate to predict accurate output.
- Compare accuracy score before and after performing backpropagation to identify the variation.
- We aim to identify the type of iris flowers by using the dataset that was prepared in advance by the expert biologists to study the flower types through some measurements and statistics for each type using data mining techniques and neural network classifiers.

SURVEY ANALYSIS

IRIS Data Analysis Using BackPropagation Neural Networks

- Sean Van Osselaer Murdoch
- Published 2011

This project paper refers to experiments towards the classification of Iris plants with back propagation neural networks (BPNN). The problem concerns the identification of Iris plant species on the basis of plant attribute measurements. The paper outlines background information concerning the problem, making reference to statistics and value constraints identified in the course of the project. There is an outline of the algorithm of techniques used within the project, with descriptions of these techniques and their context.

IRIS Flowers Classification Using Neural Network

- October 2019
- Project: Neural Network

Classification is a machine learning technique used to predict group membership for data instances. To simplify the problem of classification neural networks are being introduced. This report focuses on IRIS plant classification using Neural Network. The problem concerns the identification of IRIS plant species on the basis of plant attribute measurements. Classification of IRIS data set would be discovering patterns from examining petal and sepal size of the IRIS plant and how the prediction was made from analyzing the pattern to form the class of IRIS plant. By using this pattern and classification, in future upcoming years the unknown data can be predicted more precisely. Artificial neural networks have been successfully applied to problems in pattern classification, function approximations, optimization, and associative memories. In this work, Multilayer feed- forward networks are trained using back propagation learning algorithm. The experiential results show the minimum error rate was 0.01067 with training time of 0.691 millisecond, and the number of hidden neurons was 4.

Identification of Plant Types by Leaf Textures Based on the Backpropagation Neural Network

- December 2018
- International Journal of Electrical and Computer Engineering 8(6):5389

DOI:10.11591/ijece.v8i6.pp5389-5398

The segmentation of 1605 leaf images has been successfully done using the morphological operations. The extraction of the texture characteristics has also been successfully done based on the area value, the perimeter and such additional features of the leaf images as the roundness and the slimness of the leaf shape. The training of the extraction of leaf image characteristics was successfully performed using the backpropagation neural network. Based on the overall results of the classification testing trial on 32 classes of plants with 1605 plant leaf images there was a classification error of 48 species out of 1557 species which were successfully identified, resulting in an accuracy of 97%

PLAN OF ACTION

- Perform pre processing on the dataset
- Perform one hot encoding to the output.
- Perform sigmoid function
- Normalize the features.
- Splitting the data for training and testing.
- Perform the feedforward and backpropagation network
- Accuracy will be gathered and visualized by subtracting the error from the training data.
- Evaluating Model and testing

METHODOLOGIES USED

One Hot Encoding - integer encoded variable is removed and a new binary variable is added for each unique integer value(To convert categorical columns to numericals to predict data efficiently)

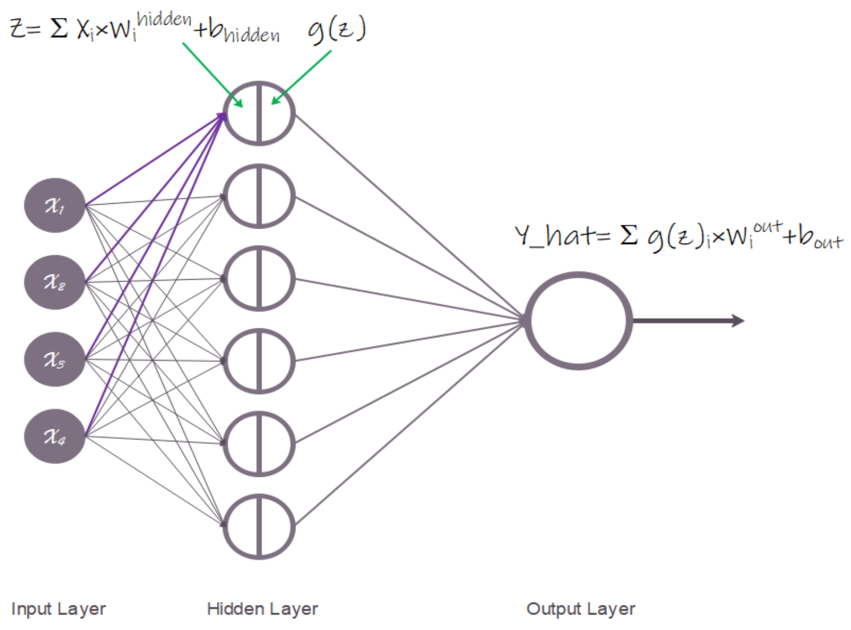
np.random seed   - function that sets the random seed of the NumPy pseudo   - random number generator.

np.zeros	- generates array with all zeros.
np.ones	- generates array with all one’s.
np.hstack	- Stack arrays in sequence horizontally (column wise).
np.dot	-dot product of arrays
np.concatenate	- function to joint arrays.

We have defined forwardpropagation( ) to calculate input ,hidden and output layer.

We have defined Backpropagation() to calculate errors and weights in hidden and output layer

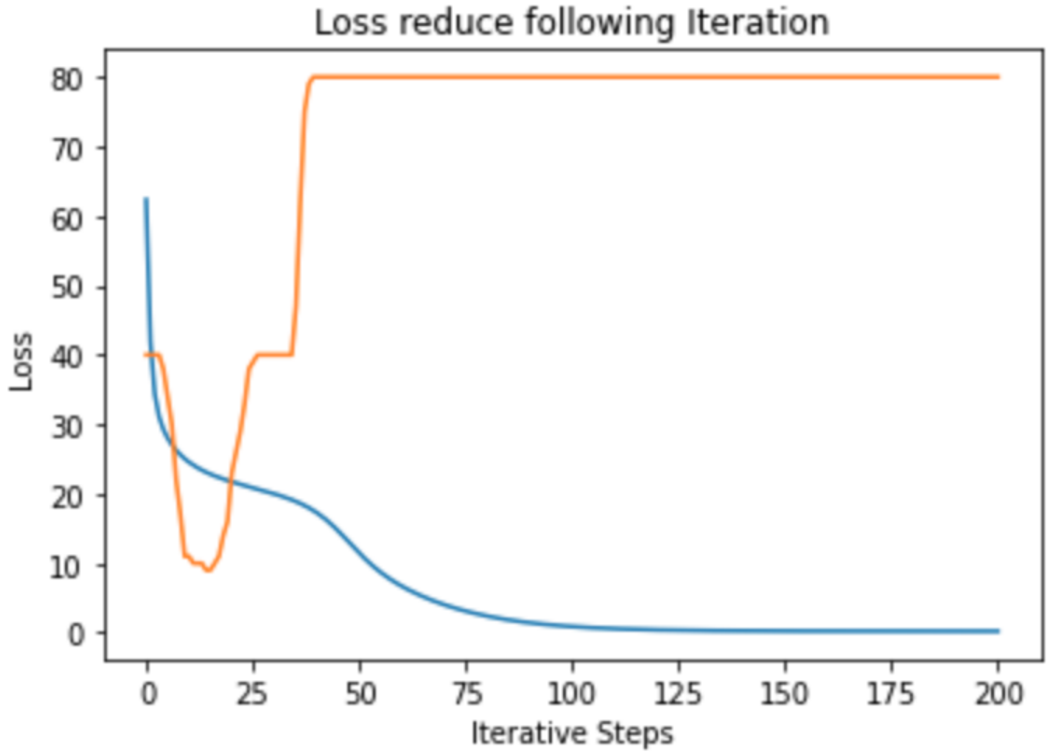
Cost function is defined to generate outputs.


$Z = \Sigma X_i \times W_i^{hidden} + b_{hidden}$
$g(z)$
x_1
x_2
x_3
x_4
$Y\_hat = \Sigma g(z)_i \times W_i^{out} + b_{out}$
Input Layer
Hidden Layer
Output Layer

RESULT ANALYSIS

After Performing the one hot encoding and defining functions we observed that there is a increase in accuracy and decrease in error rate as iterations are being performed and we have considered 200 iterations and we were able to predict 100 % accuracy at 39th iteration and most of the lost is eliminated by 97th iteration so,we could say that back propagation showed us best results .

```
------------ TEST Result ------
Loss:0.07708305810292525
Accuracy:100.0%
------------ TEST Result ------
```



CONCLUSION

Neural network is a collection of connected units with input and output mechanism, each of the connections has an associated weights.Backpropagation is the “backward propagation of errors” and is useful to train neural networks. It is fast, easy to implement and simple. Backpropagation is very beneficial for deep neural networks working over error prone projects like speech or image recognition as we have seen that our training predicted 100% accuracy.

REFERENCES

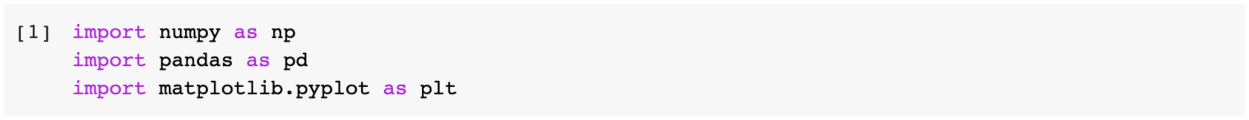
https://www.researchgate.net/publication/343149589_Iris_Flowers_Classification_Using_Neural_Network

https://www.researchgate.net/publication/238770565_Leaves_recognition_using_back_propagation_neural_network-advice_for_pest_and_disease_control_on_crops

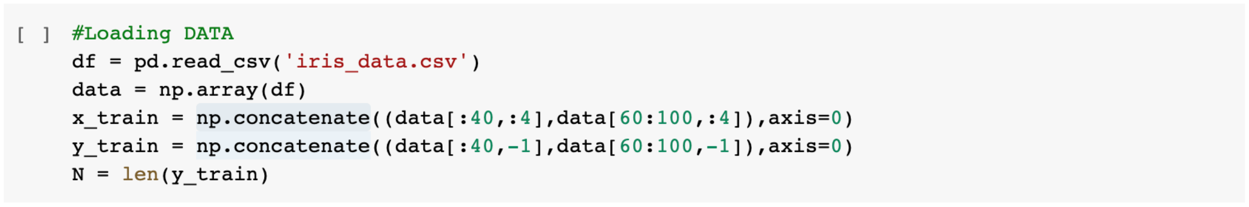
https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7

https://www.semanticscholar.org/paper/IRIS-DATA-ANALYSIS-U

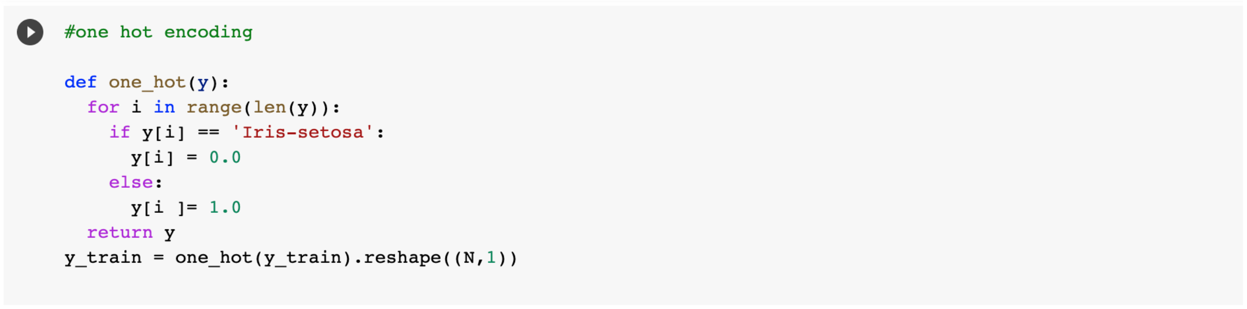
CODE



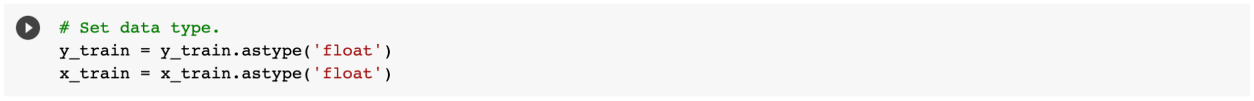
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```



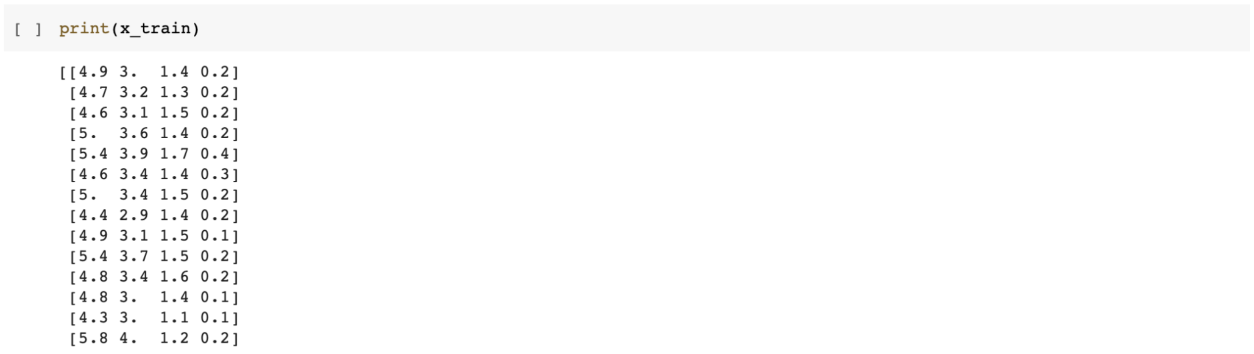
```
#Loading DATA
df = pd.read_csv('iris_data.csv')
data = np.array(df)
x_train = np.concatenate((data[:40,:4],data[60:100,:4]),axis=0)
y_train = np.concatenate((data[:40,-1],data[60:100,-1]),axis=0)
N = len(y_train)
```



```
#one hot encoding

def one_hot(y):
  for i in range(len(y)):
    if y[i] == 'Iris-setosa':
      y[i] = 0.0
    else:
      y[i ]= 1.0
  return y
y_train = one_hot(y_train).reshape((N,1))
```

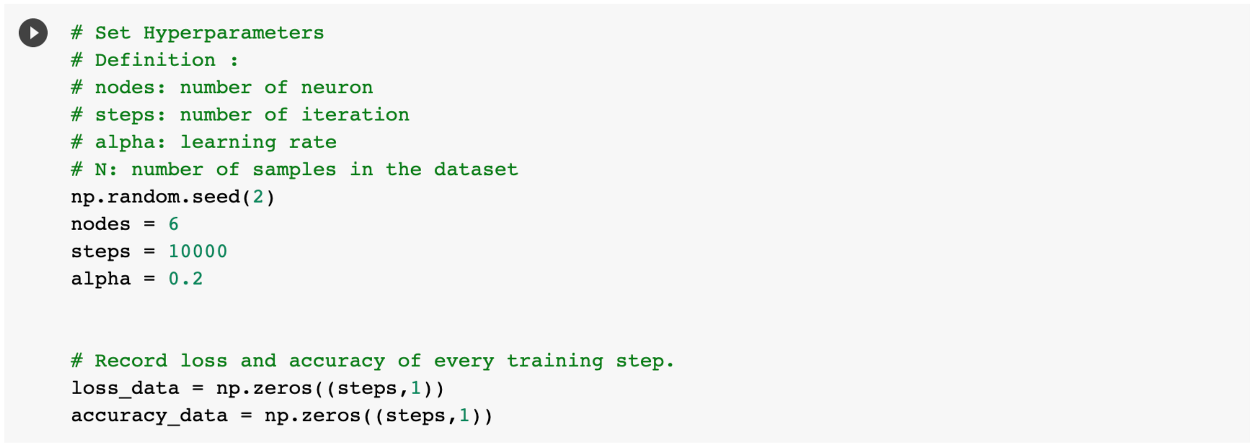


```
# Set data type.
y_train = y_train.astype('float')
x_train = x_train.astype('float')
```

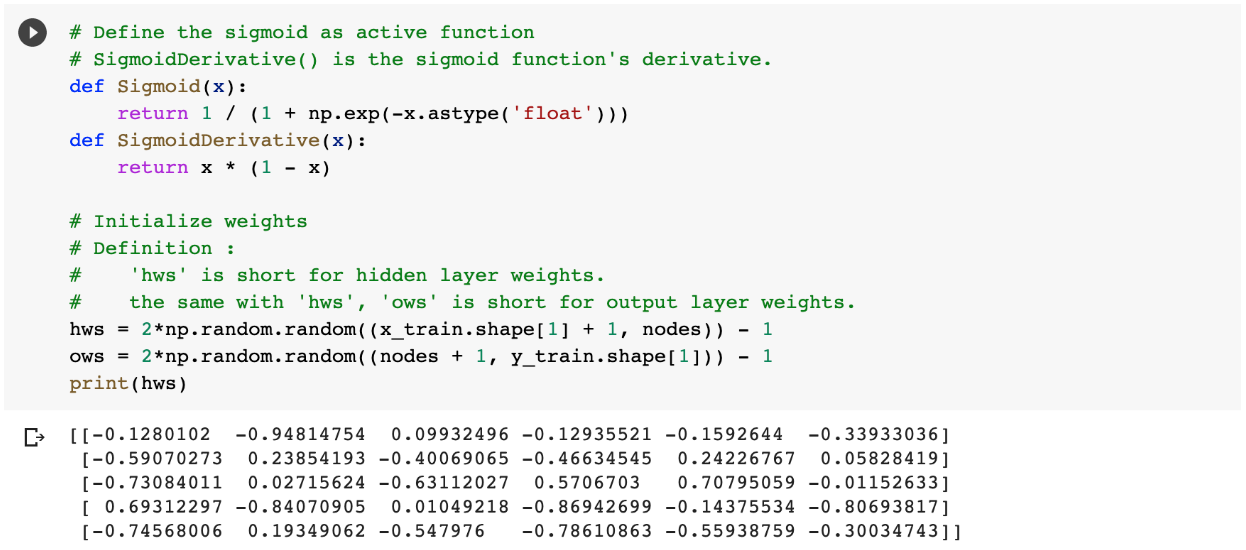


```
print(x_train)
```

```
[[4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
```

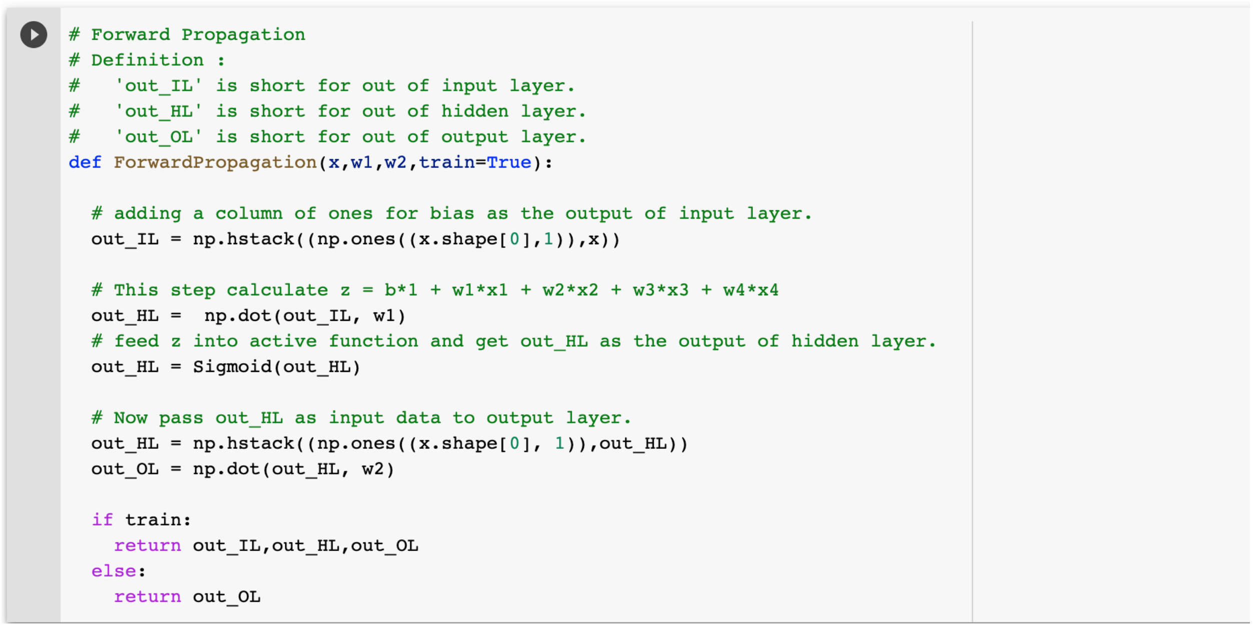


```
# Set Hyperparameters
# Definition :
# nodes: number of neuron
# steps: number of iteration
# alpha: learning rate
# N: number of samples in the dataset
np.random.seed(2)
nodes = 6
steps = 10000
alpha = 0.2


# Record loss and accuracy of every training step.
loss_data = np.zeros((steps,1))
accuracy_data = np.zeros((steps,1))
```

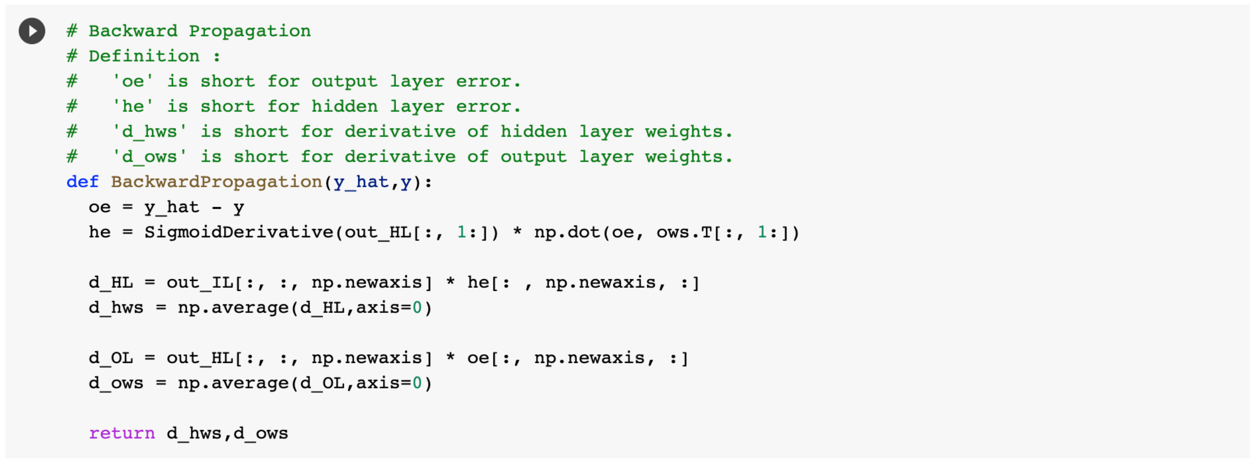


```python
# Define the sigmoid as active function
# SigmoidDerivative() is the sigmoid function's derivative.
def Sigmoid(x):
    return 1 / (1 + np.exp(-x.astype('float')))
def SigmoidDerivative(x):
    return x * (1 - x)

# Initialize weights
# Definition :
#    'hws' is short for hidden layer weights.
#    the same with 'hws', 'ows' is short for output layer weights.
hws = 2*np.random.random((x_train.shape[1] + 1, nodes)) - 1
ows = 2*np.random.random((nodes + 1, y_train.shape[1])) - 1
print(hws)
```

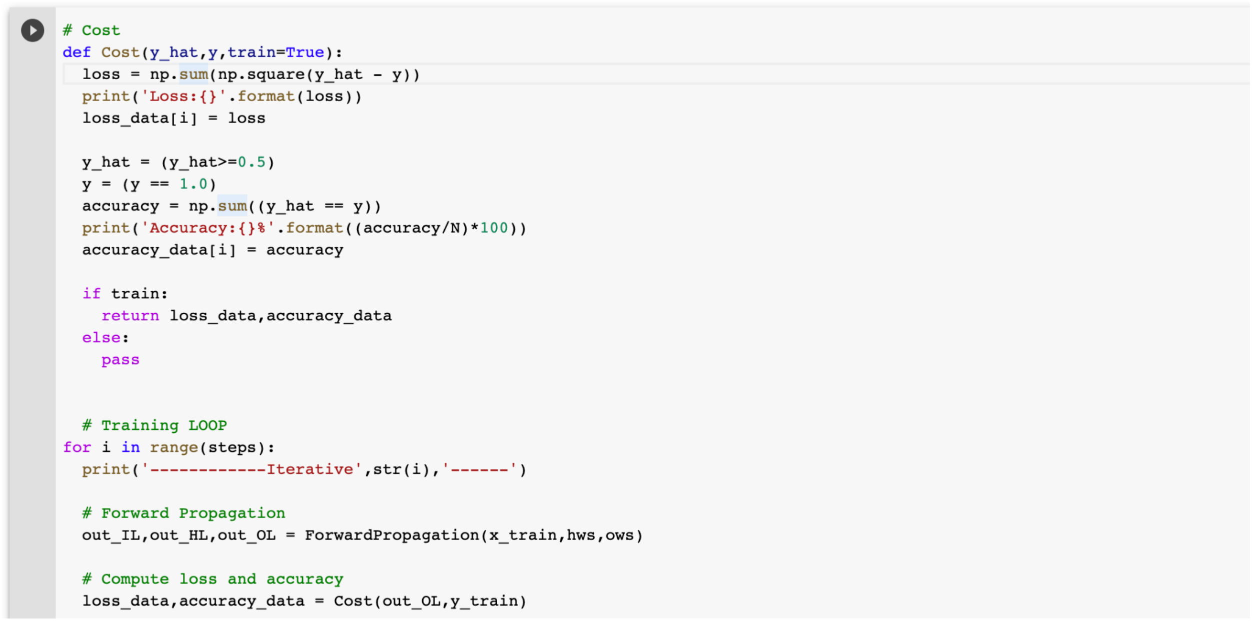
```
[[-0.1280102  -0.94814754  0.09932496 -0.12935521 -0.1592644  -0.33933036]
 [-0.59070273  0.23854193 -0.40069065 -0.46634545  0.24226767  0.05828419]
 [-0.73084011  0.02715624 -0.63112027  0.5706703   0.70795059 -0.01152633]
 [ 0.69312297 -0.84070905  0.01049218 -0.86942699 -0.14375534 -0.80693817]
 [-0.74568006  0.19349062 -0.547976   -0.78610863 -0.55938759 -0.30034743]]
```



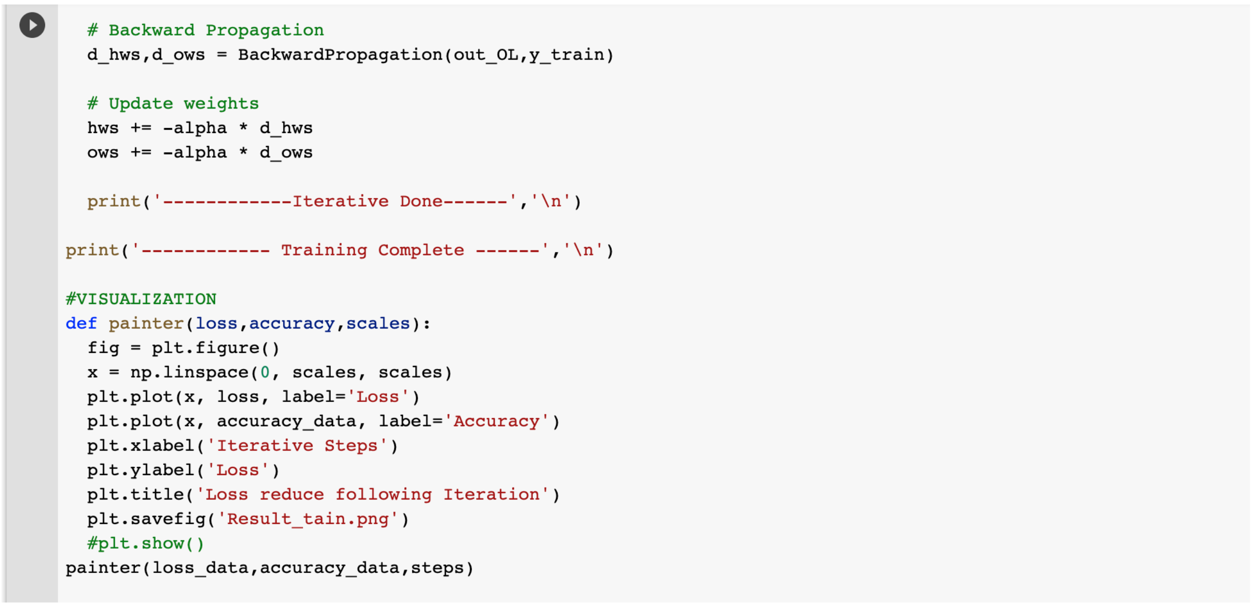
```python
# Forward Propagation
# Definition :
#   'out_IL' is short for out of input layer.
#   'out_HL' is short for out of hidden layer.
#   'out_OL' is short for out of output layer.
def ForwardPropagation(x,w1,w2,train=True):

  # adding a column of ones for bias as the output of input layer.
  out_IL = np.hstack((np.ones((x.shape[0],1)),x))

  # This step calculate z = b*1 + w1*x1 + w2*x2 + w3*x3 + w4*x4
  out_HL =  np.dot(out_IL, w1)
  # feed z into active function and get out_HL as the output of hidden layer.
  out_HL = Sigmoid(out_HL)

  # Now pass out_HL as input data to output layer.
  out_HL = np.hstack((np.ones((x.shape[0], 1)),out_HL))
  out_OL = np.dot(out_HL, w2)

  if train:
    return out_IL,out_HL,out_OL
  else:
    return out_OL
```



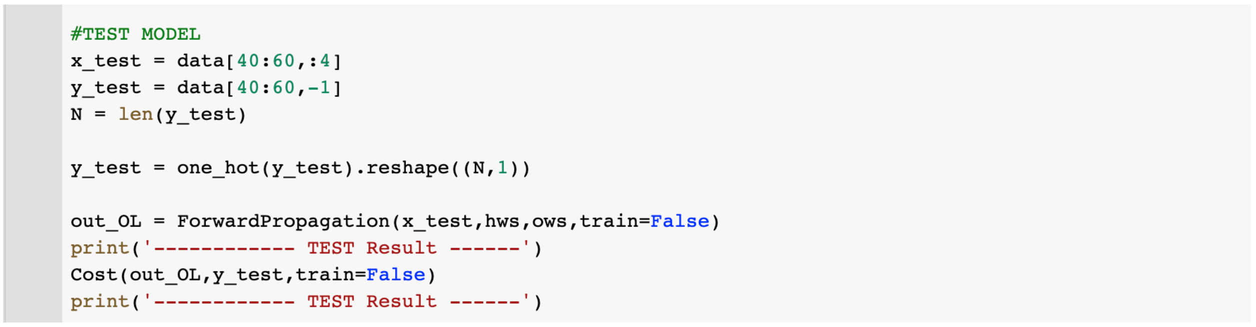
```python
# Backward Propagation
# Definition :
#   'oe' is short for output layer error.
#   'he' is short for hidden layer error.
#   'd_hws' is short for derivative of hidden layer weights.
#   'd_ows' is short for derivative of output layer weights.
def BackwardPropagation(y_hat,y):
  oe = y_hat - y
  he = SigmoidDerivative(out_HL[:, 1:]) * np.dot(oe, ows.T[:, 1:])

  d_HL = out_IL[:, :, np.newaxis] * he[: , np.newaxis, :]
  d_hws = np.average(d_HL,axis=0)

  d_OL = out_HL[:, :, np.newaxis] * oe[:, np.newaxis, :]
  d_ows = np.average(d_OL,axis=0)

  return d_hws,d_ows
```



```
# Cost
def Cost(y_hat,y,train=True):
  loss = np.sum(np.square(y_hat - y))
  print('Loss:{}'.format(loss))
  loss_data[i] = loss

  y_hat = (y_hat>=0.5)
  y = (y == 1.0)
  accuracy = np.sum((y_hat == y))
  print('Accuracy:{}%'.format((accuracy/N)*100))
  accuracy_data[i] = accuracy

  if train:
    return loss_data,accuracy_data
  else:
    pass


  # Training LOOP
for i in range(steps):
  print('------------Iterative',str(i),'------')

  # Forward Propagation
  out_IL,out_HL,out_OL = ForwardPropagation(x_train,hws,ows)

  # Compute loss and accuracy
  loss_data,accuracy_data = Cost(out_OL,y_train)
```

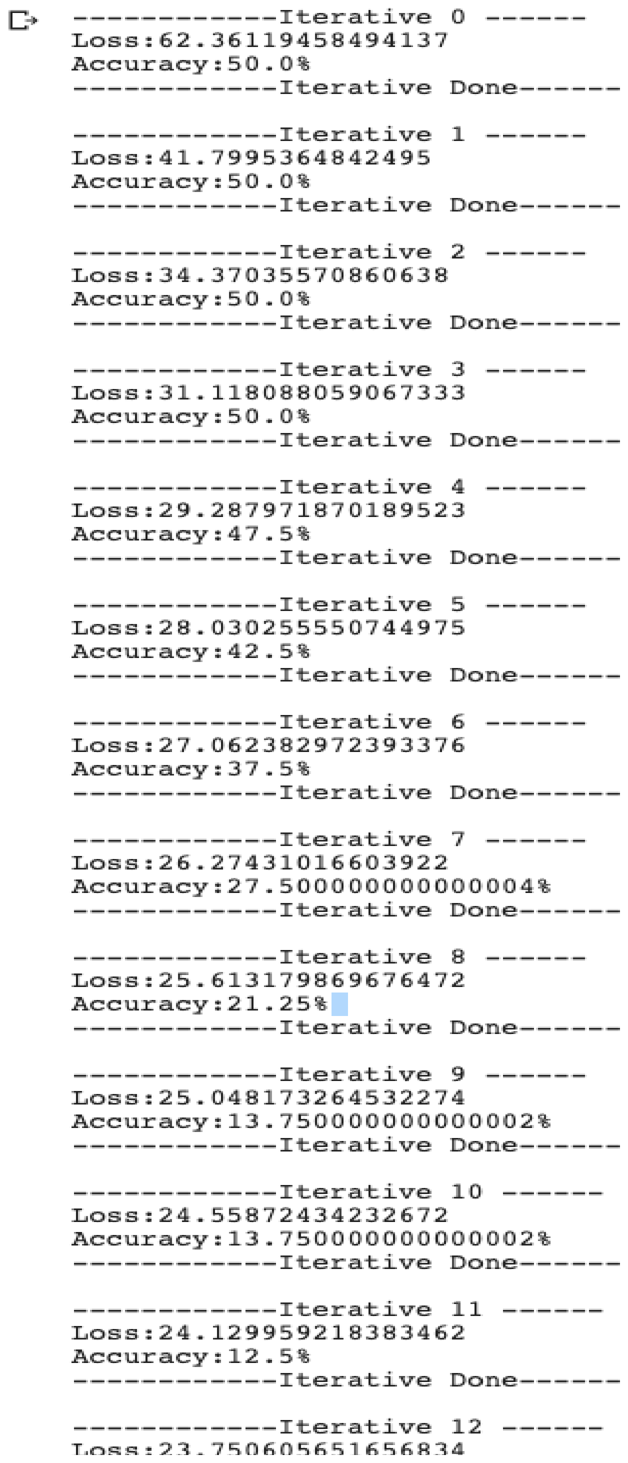


```
  # Backward Propagation
  d_hws,d_ows = BackwardPropagation(out_OL,y_train)

  # Update weights
  hws += -alpha * d_hws
  ows += -alpha * d_ows

  print('------------Iterative Done------','\n')

print('------------ Training Complete ------','\n')

#VISUALIZATION
def painter(loss,accuracy,scales):
  fig = plt.figure()
  x = np.linspace(0, scales, scales)
  plt.plot(x, loss, label='Loss')
  plt.plot(x, accuracy_data, label='Accuracy')
  plt.xlabel('Iterative Steps')
  plt.ylabel('Loss')
  plt.title('Loss reduce following Iteration')
  plt.savefig('Result_tain.png')
  #plt.show()
painter(loss_data,accuracy_data,steps)
```

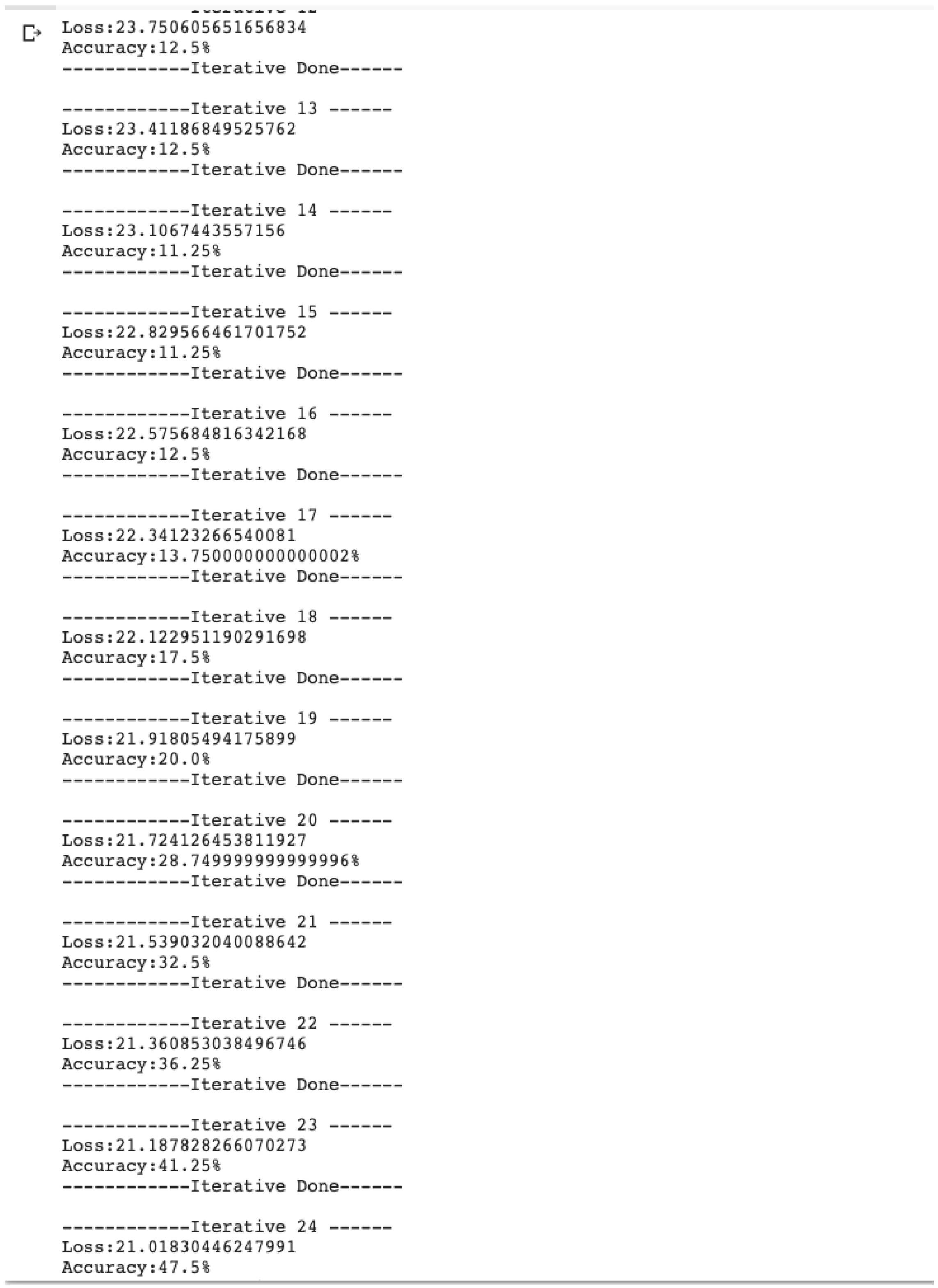


```
#TEST MODEL
x_test = data[40:60,:4]
y_test = data[40:60,-1]
N = len(y_test)

y_test = one_hot(y_test).reshape((N,1))

out_OL = ForwardPropagation(x_test,hws,ows,train=False)
print('------------ TEST Result ------')
Cost(out_OL,y_test,train=False)
print('------------ TEST Result ------')
```

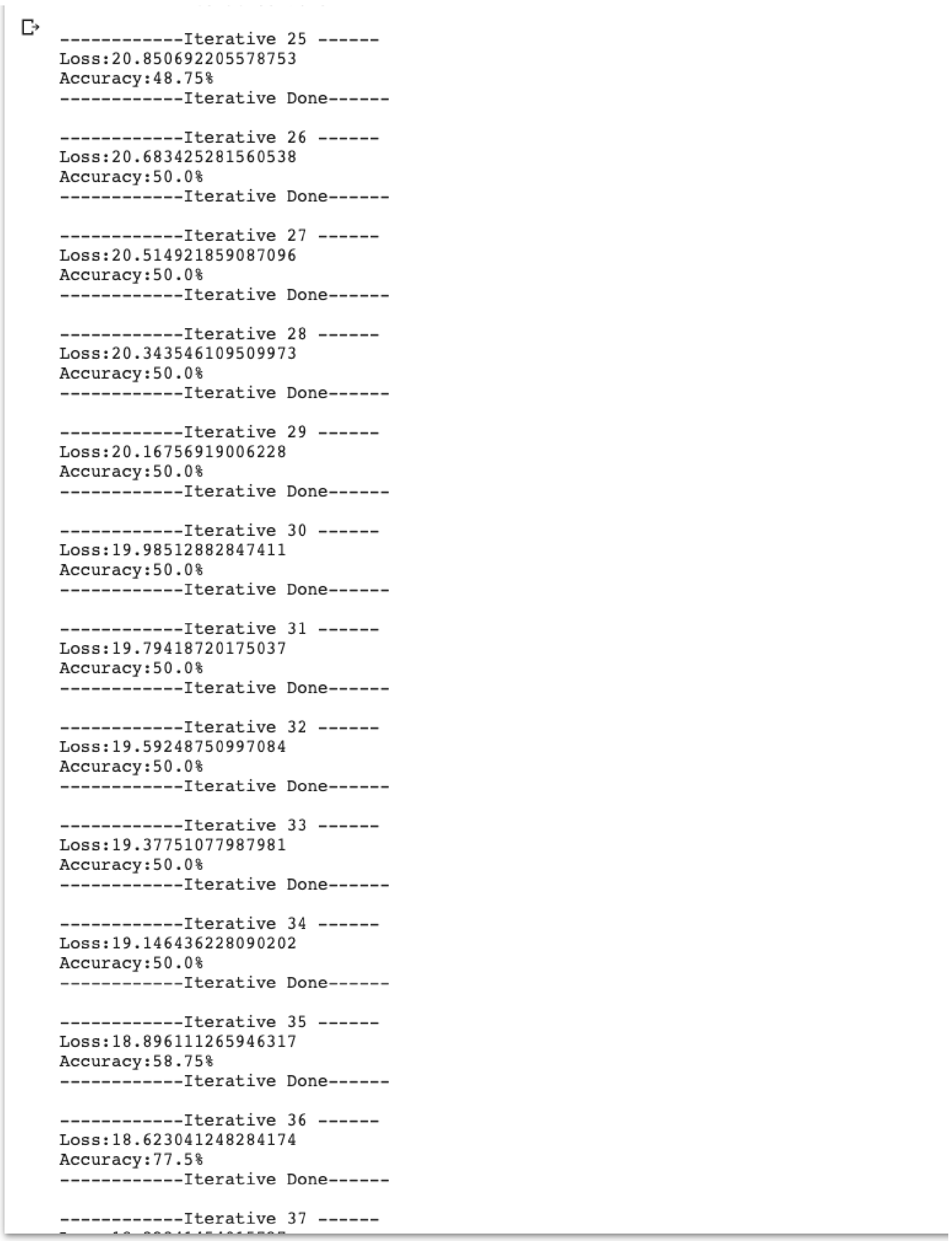
OUTPUT



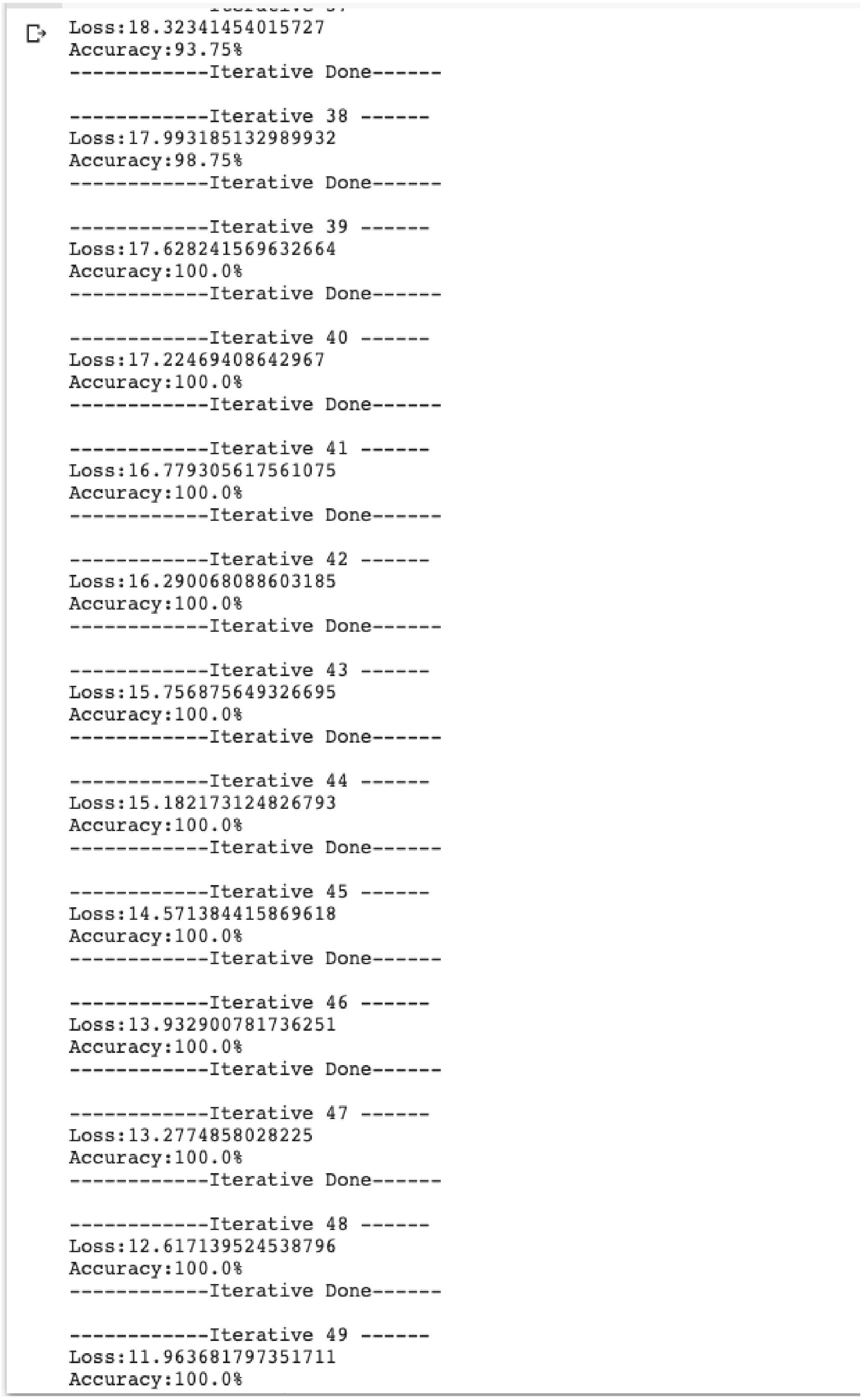
```
------------Iterative 0 ------
Loss:62.36119458494137
Accuracy:50.0%
------------Iterative Done------

------------Iterative 1 ------
Loss:41.7995364842495
Accuracy:50.0%
------------Iterative Done------

------------Iterative 2 ------
Loss:34.37035570860638
Accuracy:50.0%
------------Iterative Done------

------------Iterative 3 ------
Loss:31.118088059067333
Accuracy:50.0%
------------Iterative Done------

------------Iterative 4 ------
Loss:29.287971870189523
Accuracy:47.5%
------------Iterative Done------

------------Iterative 5 ------
Loss:28.030255550744975
Accuracy:42.5%
------------Iterative Done------

------------Iterative 6 ------
Loss:27.062382972393376
Accuracy:37.5%
------------Iterative Done------

------------Iterative 7 ------
Loss:26.27431016603922
Accuracy:27.500000000000004%
------------Iterative Done------

------------Iterative 8 ------
Loss:25.613179869676472
Accuracy:21.25%
------------Iterative Done------

------------Iterative 9 ------
Loss:25.048173264532274
Accuracy:13.750000000000002%
------------Iterative Done------

------------Iterative 10 ------
Loss:24.55872434232672
Accuracy:13.750000000000002%
------------Iterative Done------

------------Iterative 11 ------
Loss:24.129959218383462
Accuracy:12.5%
------------Iterative Done------

------------Iterative 12 ------
Loss:23.750605651656834
```



```
Loss:23.750605651656834
Accuracy:12.5%
------------Iterative Done------

------------Iterative 13 ------
Loss:23.41186849525762
Accuracy:12.5%
------------Iterative Done------

------------Iterative 14 ------
Loss:23.1067443557156
Accuracy:11.25%
------------Iterative Done------

------------Iterative 15 ------
Loss:22.829566461701752
Accuracy:11.25%
------------Iterative Done------

------------Iterative 16 ------
Loss:22.575684816342168
Accuracy:12.5%
------------Iterative Done------

------------Iterative 17 ------
Loss:22.34123266540081
Accuracy:13.750000000000002%
------------Iterative Done------

------------Iterative 18 ------
Loss:22.122951190291698
Accuracy:17.5%
------------Iterative Done------

------------Iterative 19 ------
Loss:21.91805494175899
Accuracy:20.0%
------------Iterative Done------

------------Iterative 20 ------
Loss:21.724126453811927
Accuracy:28.749999999999996%
------------Iterative Done------

------------Iterative 21 ------
Loss:21.539032040088642
Accuracy:32.5%
------------Iterative Done------

------------Iterative 22 ------
Loss:21.360853038496746
Accuracy:36.25%
------------Iterative Done------

------------Iterative 23 ------
Loss:21.187828266070273
Accuracy:41.25%
------------Iterative Done------

------------Iterative 24 ------
Loss:21.01830446247991
Accuracy:47.5%
```

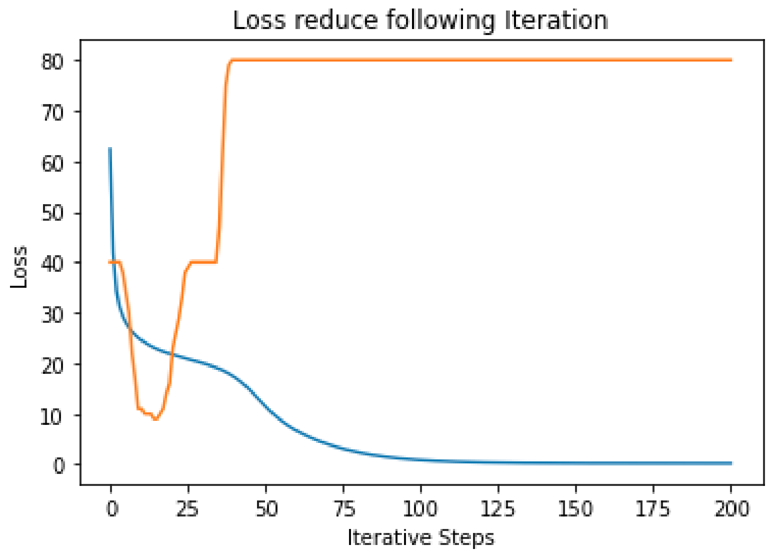


```
------------Iterative 25 ------
Loss:20.850692205578753
Accuracy:48.75%
------------Iterative Done------

------------Iterative 26 ------
Loss:20.683425281560538
Accuracy:50.0%
------------Iterative Done------

------------Iterative 27 ------
Loss:20.514921859087096
Accuracy:50.0%
------------Iterative Done------

------------Iterative 28 ------
Loss:20.343546109509973
Accuracy:50.0%
------------Iterative Done------

------------Iterative 29 ------
Loss:20.16756919006228
Accuracy:50.0%
------------Iterative Done------

------------Iterative 30 ------
Loss:19.98512882847411
Accuracy:50.0%
------------Iterative Done------

------------Iterative 31 ------
Loss:19.79418720175037
Accuracy:50.0%
------------Iterative Done------

------------Iterative 32 ------
Loss:19.59248750997084
Accuracy:50.0%
------------Iterative Done------

------------Iterative 33 ------
Loss:19.37751077987981
Accuracy:50.0%
------------Iterative Done------

------------Iterative 34 ------
Loss:19.146436228090202
Accuracy:50.0%
------------Iterative Done------

------------Iterative 35 ------
Loss:18.896111265946317
Accuracy:58.75%
------------Iterative Done------

------------Iterative 36 ------
Loss:18.623041248284174
Accuracy:77.5%
------------Iterative Done------

------------Iterative 37 ------
```



```
Loss:18.32341454015727
Accuracy:93.75%
------------Iterative Done------

------------Iterative 38 ------
Loss:17.993185132989932
Accuracy:98.75%
------------Iterative Done------

------------Iterative 39 ------
Loss:17.628241569632664
Accuracy:100.0%
------------Iterative Done------

------------Iterative 40 ------
Loss:17.22469408642967
Accuracy:100.0%
------------Iterative Done------

------------Iterative 41 ------
Loss:16.779305617561075
Accuracy:100.0%
------------Iterative Done------

------------Iterative 42 ------
Loss:16.290068088603185
Accuracy:100.0%
------------Iterative Done------

------------Iterative 43 ------
Loss:15.756875649326695
Accuracy:100.0%
------------Iterative Done------

------------Iterative 44 ------
Loss:15.182173124826793
Accuracy:100.0%
------------Iterative Done------

------------Iterative 45 ------
Loss:14.571384415869618
Accuracy:100.0%
------------Iterative Done------

------------Iterative 46 ------
Loss:13.932900781736251
Accuracy:100.0%
------------Iterative Done------

------------Iterative 47 ------
Loss:13.2774858028225
Accuracy:100.0%
------------Iterative Done------

------------Iterative 48 ------
Loss:12.617139524538796
Accuracy:100.0%
------------Iterative Done------

------------Iterative 49 ------
Loss:11.963681797351711
Accuracy:100.0%
```

```
------------Iterative 195 ------
Loss:0.21747952527583855
Accuracy:100.0%
------------Iterative Done------

------------Iterative 196 ------
Loss:0.2172978012071697
Accuracy:100.0%
------------Iterative Done------

------------Iterative 197 ------
Loss:0.2171210804589928
Accuracy:100.0%
------------Iterative Done------

------------Iterative 198 ------
Loss:0.21694904125492187
Accuracy:100.0%
------------Iterative Done------

------------Iterative 199 ------
Loss:0.21678138283219936
Accuracy:100.0%
------------Iterative Done------

------------ Training Complete ------

------------ TEST Result ------
Loss:0.07708305810292525
Accuracy:100.0%
------------ TEST Result ------
```


Loss reduce following Iteration
Loss
Iterative Steps