**LAB EXERCISE – 17**

**Multi Layer Perceptron**

1. **Aim of the Experiment:**

Implement and demonstrate Multi-Layer Perceptron (MLP) model with back propagation to solve the XOR Boolean function.

1. **Reference to Text book for Algorithms:**

Refer to Section 10.6 in Chapter 10 Artificial Neural Networks to understand the working of the algorithm.

**Listing 1: XOR Boolean function**

**Python Program with Explanation:**

1. Import numpy, array-processing package to work with the arrays.

import numpy as np

2. Define a function abs() that returns the absolute value of x.

def abs(x):

return x if x>0 else –x

3. Define sigmoid(x) to implement a logistic sigmoid activation function.

def sigmoid (x):

return 1/(1 + np.exp(-x))

4. Define sigmoid\_derivative(x) used in computing error in back propagation phase.

def sigmoid\_derivative(x):

return x \* (1 - x)

5. Define the error function/cost function which checks if there is error between the expected

output and predicted output and returns a Boolean value.

def checkError(predicted\_output):

expected\_output = [[0],[1],[1],[0]]

for i,j in zip(expected\_output , predicted\_output):

if abs(i[0]-j[0]) > 0.001:

return True

return False

6. Set input data and desired output of an XOR Boolean function.

#Input datasets

inputs = np.array([[0,0],[0,1],[1,0],[1,1]])

expected\_output = np.array([[0],[1],[1],[0]])

7. Initialialize epoch to 0 and learning rate to 0.1.

epoch = 0

lr = 0.1

8. Define the MLP network which consists of 2 neurons in Input layer, 2 neurons in Hidden

layer and 1 neuron in the Output layer. Accept the number of neurons in each layer.

# inputLayerNeurons = 2

# hiddenLayerNeurons = 2

# outputLayerNeurons = 1

inputLayerNeurons = int(input("enter no of inputLayer"))

hiddenLayerNeurons = int(input("enter no of hiddenlayer "))

outputLayerNeurons = int(input("enter no of outputlayer "))

9. Define a list called hidden\_weights to store the weights of the hidden layer neurons in the

network.

hidden\_weights = []

10. Receive the weights of the link from the input layer neurons to hidden layer neurons.

for i in range(1,inputLayerNeurons+1):

hidden\_weights\_ind = []

for j in range(inputLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+1):

hidden\_weights\_ind.append(float(input('w'+str(i)+str(j))))

hidden\_weights.append(hidden\_weights\_ind)

11. Define list called output\_weights to store the weights of the output layer neurons in the network.

output\_weights = []

12. Receive the weights of the link from the Hidden layer neurons to Output layer neurons.

for i in range(inputLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+1):

output\_weights\_ind = []

for j in range(inputLayerNeurons+hiddenLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+outputLayerNeurons+1):

output\_weights\_ind.append(float(input('w'+str(i)+str(j))))

output\_weights.append(output\_weights\_ind)

13. Define lists called hidden\_bias and output\_bias to store the bias of the Hidden layer

neurons and Output layer neurons in the network.

hidden\_bias = []

output\_bias = []

14. Receive the Hidden layer biases and Output layer biases.

for i in range(inputLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+outputLayerNeurons+1):

if i > inputLayerNeurons+hiddenLayerNeurons:

output\_bias.append(float(input("o"+str(i))))

else:

hidden\_bias.append(float(input("o"+str(i))))

15. Convert the weight lists and bias lists to array.

hidden\_weights = np.asarray(hidden\_weights)

hidden\_bias = np.asarray([hidden\_bias])

output\_weights = np.asarray(output\_weights)

output\_bias = np.asarray([output\_bias])

16. Print the initial hidden weights, hidden biases, output weights and output biases.

print("Initial hidden weights: ",end='')

print(\*hidden\_weights)

print("Initial hidden biases: ",end='')

print(\*hidden\_bias)

print("Initial output weights: ",end='')

print(\*output\_weights)

print("Initial output biases: ",end='')

print(\*output\_bias)

17. Initialize the predicted\_output.

predicted\_output = [[0],[0],[0],[0]]

18. Train MLP until the predicted output converges to the desired output.

while checkError(predicted\_output):

epoch += 1

**Step 1: Forward Propagation.**

Calculate Net Input and Output in the Hidden Layer and Output Layer.

hidden\_layer\_activation = np.dot(inputs,hidden\_weights)

hidden\_layer\_activation += hidden\_bias

hidden\_layer\_output = sigmoid(hidden\_layer\_activation)

output\_layer\_activation = np.dot(hidden\_layer\_output,output\_weights)

output\_layer\_activation += output\_bias

predicted\_output = sigmoid(output\_layer\_activation)

Estimate error at the node in the Output Layer.

error = expected\_output - predicted\_output

**Step 2: Backward Propagation**

Calculate Error at each node in the Output layer and Hidden layer.

d\_predicted\_output = error \* sigmoid\_derivative(predicted\_output)

error\_hidden\_layer = d\_predicted\_output.dot(output\_weights.T)

d\_hidden\_layer = error\_hidden\_layer \* sigmoid\_derivative(hidden\_layer\_output)

Update all Weights and Biases.

output\_weights += hidden\_layer\_output.T.dot(d\_predicted\_output) \* lr

output\_bias += np.sum(d\_predicted\_output,axis=0,keepdims=True) \* lr

hidden\_weights += inputs.T.dot(d\_hidden\_layer) \* lr

hidden\_bias += np.sum(d\_hidden\_layer,axis=0,keepdims=True) \* lr

19. Print the final learned weights and biases of the Hidden layer and Output layer.

print("Final hidden weights: ",end='')

print(\*hidden\_weights)

print("Final hidden bias: ",end='')

print(\*hidden\_bias)

print("Final output weights: ",end='')

print(\*output\_weights)

print("Final output bias: ",end='')

print(\*output\_bias)

20. Print the final output obtained for the input data set (i.e., for the XOR function)

print("\nOutput from neural network: ",end='')

print(\*predicted\_output)

21. Print the number of epochs executed to learn the weights and biases for the model to get

the desired output.

print("\nNo of epochs")

print(epoch)

**Complete Program:**

import numpy as np

def abs(x):

return x if x>0 else -x

def sigmoid (x):

return 1/(1 + np.exp(-x))

def sigmoid\_derivative(x):

return x \* (1 - x)

def checkError(predicted\_output):

expected\_output = [[0],[1],[1],[0]]

for i,j in zip(expected\_output , predicted\_output):

if abs(i[0]-j[0]) > 0.001:

return True

return False

#Input datasets

inputs = np.array([[0,0],[0,1],[1,0],[1,1]])

expected\_output = np.array([[0],[1],[1],[0]])

epoch = 0

lr = 0.1

# inputLayerNeurons = 2

# hiddenLayerNeurons = 2

# outputLayerNeurons = 1

inputLayerNeurons = int(input("enter no of inputLayer"))

hiddenLayerNeurons = int(input("enter no of hiddenlayer "))

outputLayerNeurons = int(input("enter no of outputlayer "))

hidden\_weights = []

for i in range(1,inputLayerNeurons+1):

hidden\_weights\_ind = []

for j in range(inputLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+1):

hidden\_weights\_ind.append(float(input('w'+str(i)+str(j))))

hidden\_weights.append(hidden\_weights\_ind)

output\_weights = []

for i in range(inputLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+1):

output\_weights\_ind = []

for j in range(inputLayerNeurons+hiddenLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+outputLayerNeurons+1):

output\_weights\_ind.append(float(input('w'+str(i)+str(j))))

output\_weights.append(output\_weights\_ind)

hidden\_bias = []

output\_bias = []

for i in range(inputLayerNeurons+1,inputLayerNeurons+hiddenLayerNeurons+outputLayerNeurons+1):

if i > inputLayerNeurons+hiddenLayerNeurons:

output\_bias.append(float(input("o"+str(i))))

else:

hidden\_bias.append(float(input("o"+str(i))))

hidden\_weights = np.asarray(hidden\_weights)

hidden\_bias = np.asarray([hidden\_bias])

output\_weights = np.asarray(output\_weights)

output\_bias = np.asarray([output\_bias])

print("Initial hidden weights: ",end='')

print(\*hidden\_weights)

print("Initial hidden biases: ",end='')

print(\*hidden\_bias)

print("Initial output weights: ",end='')

print(\*output\_weights)

print("Initial output biases: ",end='')

print(\*output\_bias)

predicted\_output = [[0],[0],[0],[0]]

#Training algorithm

while checkError(predicted\_output):

epoch += 1

#Forward Propagation

hidden\_layer\_activation = np.dot(inputs,hidden\_weights)

hidden\_layer\_activation += hidden\_bias

hidden\_layer\_output = sigmoid(hidden\_layer\_activation)

output\_layer\_activation = np.dot(hidden\_layer\_output,output\_weights)

output\_layer\_activation += output\_bias

predicted\_output = sigmoid(output\_layer\_activation)

#Backpropagation

error = expected\_output - predicted\_output

d\_predicted\_output = error \* sigmoid\_derivative(predicted\_output)

error\_hidden\_layer = d\_predicted\_output.dot(output\_weights.T)

d\_hidden\_layer = error\_hidden\_layer \* sigmoid\_derivative(hidden\_layer\_output)

#Updating Weights and Biases

output\_weights += hidden\_layer\_output.T.dot(d\_predicted\_output) \* lr

output\_bias += np.sum(d\_predicted\_output,axis=0,keepdims=True) \* lr

hidden\_weights += inputs.T.dot(d\_hidden\_layer) \* lr

hidden\_bias += np.sum(d\_hidden\_layer,axis=0,keepdims=True) \* lr

print("Final hidden weights: ",end='')

print(\*hidden\_weights)

print("Final hidden bias: ",end='')

print(\*hidden\_bias)

print("Final output weights: ",end='')

print(\*output\_weights)

print("Final output bias: ",end='')

print(\*output\_bias)

print("\nOutput from neural network: ",end='')

print(\*predicted\_output)

print("\nNo of epochs")

print(epoch)

**Output:**

Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license()" for more information.

>>>

= RESTART: C:\Users\ADMIN\pythonpgms\Assignment1\Venkatesh\multilayer\_perceptron\_xor.py

enter no of inputLayer2

enter no of hiddenlayer 2

enter no of outputlayer 1

w133

w146

w234

w245

w352

w454

o31

o4-6

o5-3.93

Initial hidden weights: [3. 6.] [4. 5.]

Initial hidden biases: [ 1. -6.]

Initial output weights: [2.] [4.]

Initial output biases: [-3.93]

Final hidden weights: [ 6.12370882 10.03281141] [ 6.12342151 10.0272853 ]

Final hidden bias: [-9.34571401 -4.50662615]

Final output weights: [-15.62277004] [14.81103743]

Final output bias: [-7.06705554]

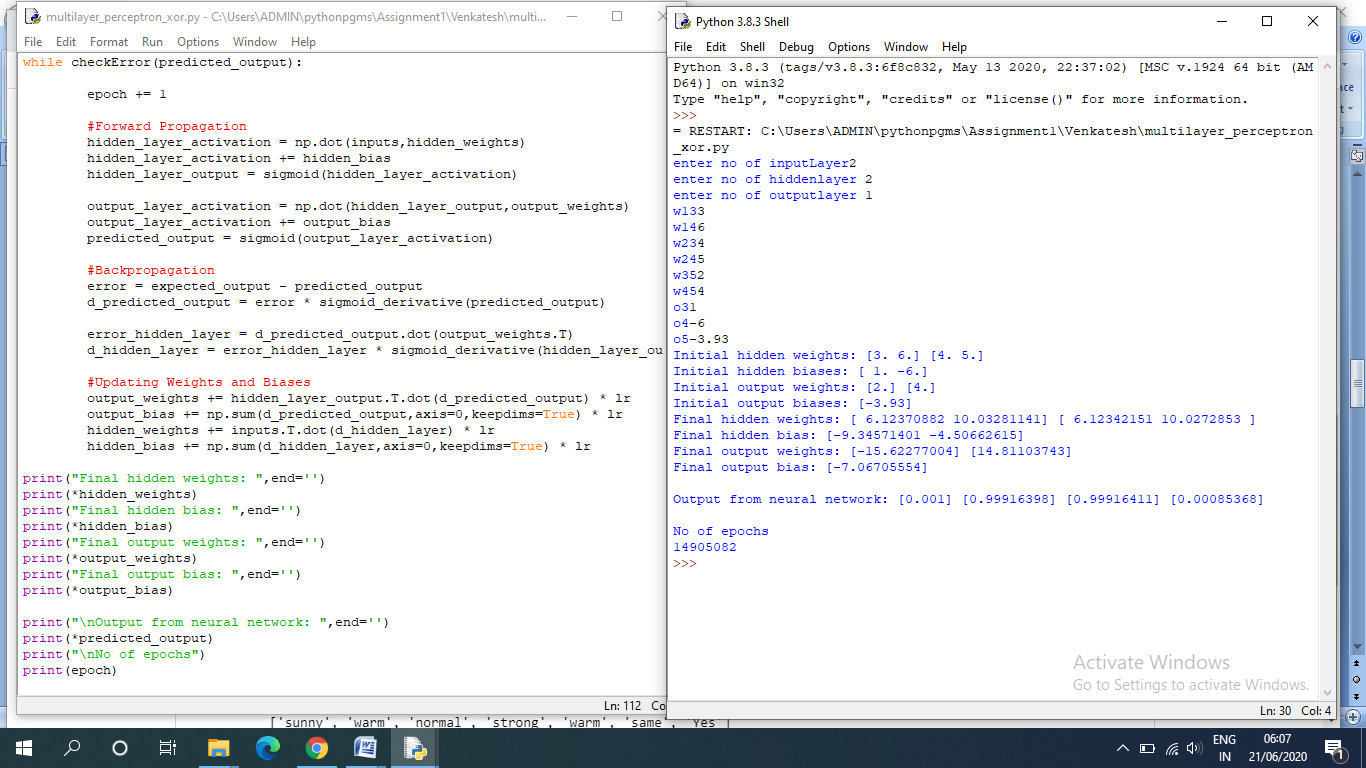
Output from neural network: [0.001] [0.99916398] [0.99916411] [0.00085368]

No of epochs

14905082

>>>

**Screenshot of the Output:**



**Listing 2:** MLP model using Keras with Iris dataset. Validating the model on the test data and then plotting the learning curve.

**Program Code:**

from pandas import read\_csv

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.optimizers import SGD

from matplotlib import pyplot

# Read the dataset ‘Iris.csv’

df = read\_csv('Iris.csv')

# Split the Iris features into input and output columns

X = df.values[:, :-1]

y = df.values[:, -1]

# Check all data are floating point values

X = X.astype('float32')

# Encode the strings of labels to integer values

y = LabelEncoder().fit\_transform(y)

# Split the data matrix into train and test dataset and Print the shape of train dataset and test dataset.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle = True, random\_state = 123)

print("X\_train shape: {}".format(X\_train.shape))

print("X\_test shape: {}".format(X\_test.shape))

print("y\_train shape: {}".format(y\_train.shape))

print("y\_test shape: {}".format(y\_test.shape))

# split randomly the data matrix into training dataset and validation dataset

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.2, random\_state=1)

# Determine the number of input features

n\_features = X\_train.shape[1]

# Define the model

model = Sequential()

model.add(Dense(10, activation='relu', kernel\_initializer='he\_normal', input\_shape=(n\_features,)))

model.add(Dense(8, activation='relu', kernel\_initializer='he\_normal'))

model.add(Dense(3, activation='softmax'))

# Compile the model

sgd = SGD(learning\_rate=0.001, momentum=0.8)

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Fit the model

history = model.fit(X\_train, y\_train, epochs=150, batch\_size=32, verbose=0, validation\_split=0.3)

# Evaluate the model and print the accuracy

loss, acc = model.evaluate(X\_test, y\_test, verbose=0)

print('Test Accuracy: %.3f' % acc)

# Visualize by plotting the learning curves

pyplot.title('Learning Curves')

pyplot.xlabel('Epoch')

pyplot.ylabel('Cross Entropy')

pyplot.plot(history.history['loss'], label='train')

pyplot.plot(history.history['val\_loss'], label='val')

pyplot.legend()

pyplot.show()

**Output:**

Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license()" for more information.

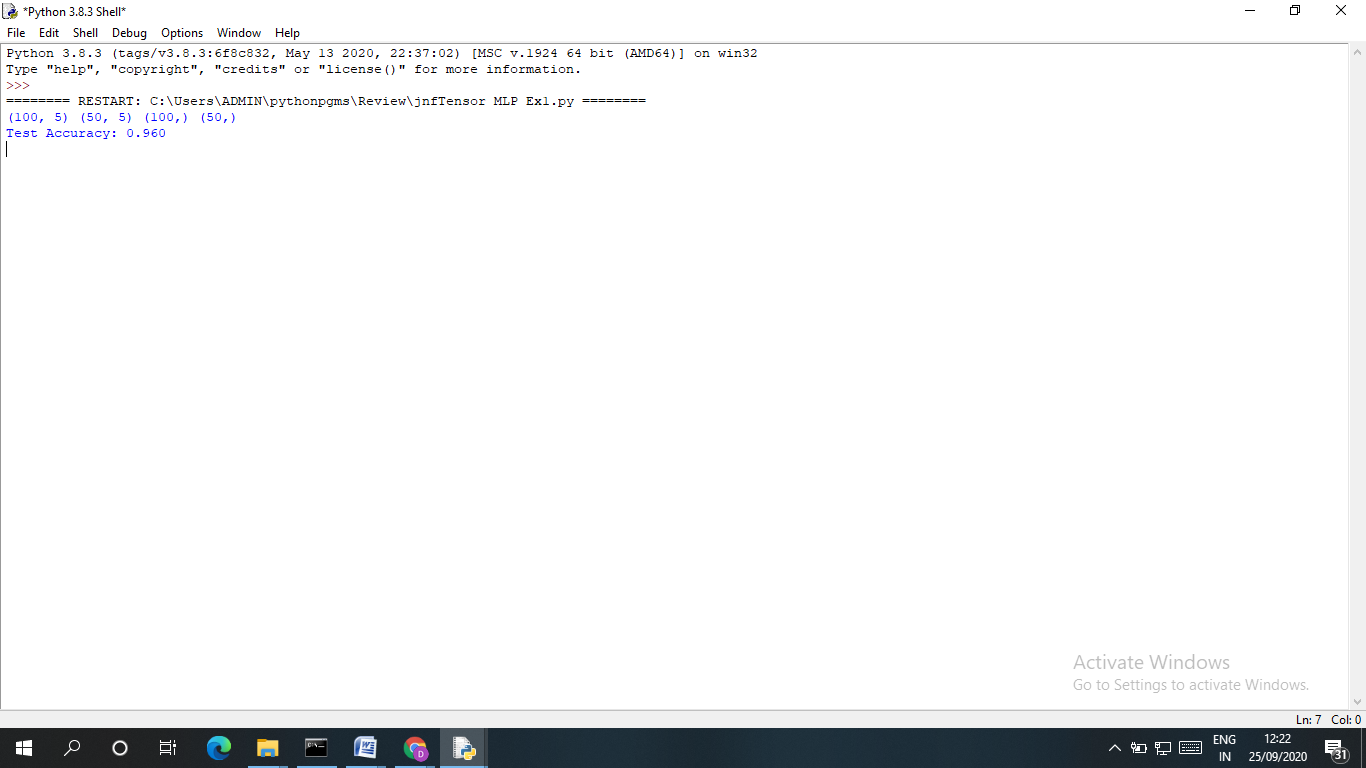
>>>

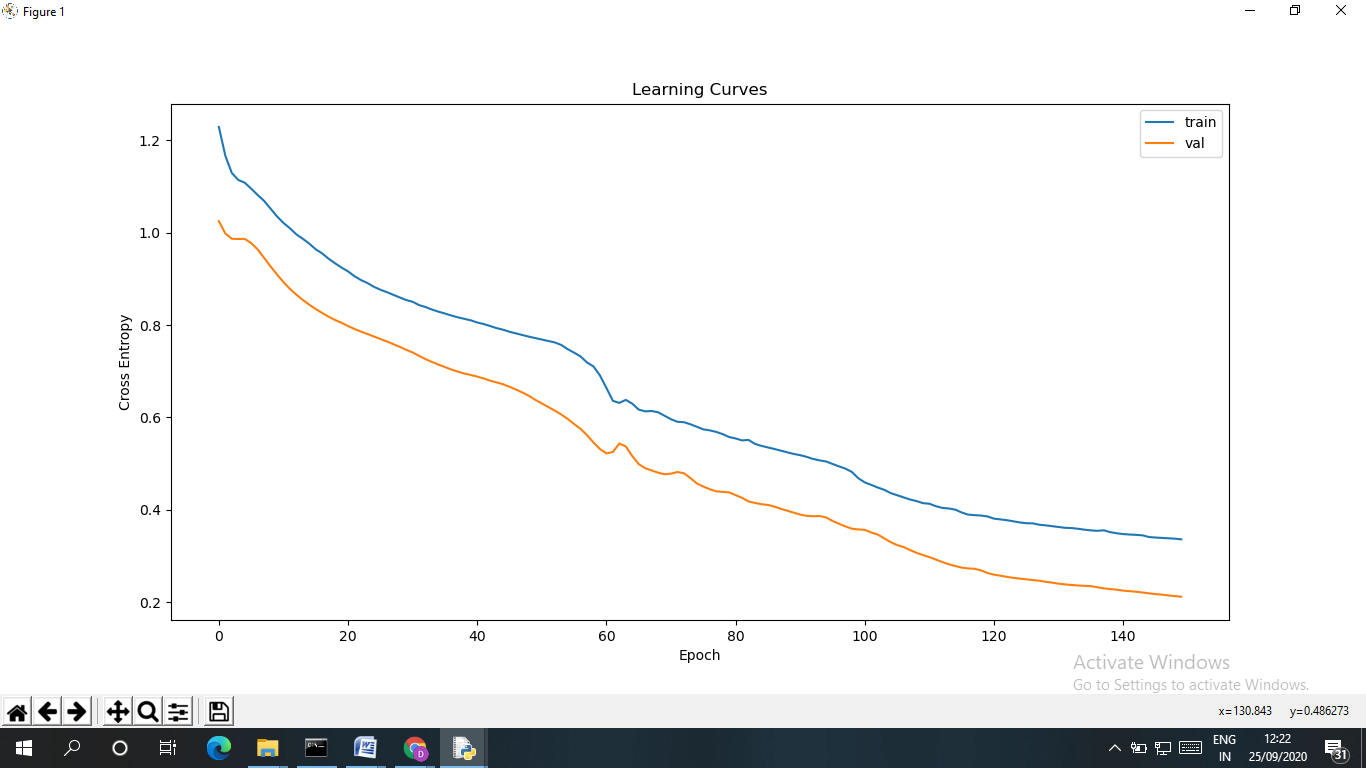
======== RESTART: C:\Users\ADMIN\pythonpgms\Review\jnfTensor MLP Ex1.py ========

(100, 5) (50, 5) (100,) (50,)

Test Accuracy: 0.960

>>>





**Listing 3:** MLP model using Keras with Iris dataset. Validating the model on the test data and then plotting the accuracy and loss.

**Program Code:**

from pandas import read\_csv

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Dense

from matplotlib import pyplot as plt

# Read the dataset ‘Iris.csv’

df = read\_csv('Iris.csv')

# Split the Iris features into input and output columns

X = df.values[:, :-1]

y = df.values[:, -1]

# Check all data are floating point values

X = X.astype('float32')

# Encode the strings of labels to integer values

y = LabelEncoder().fit\_transform(y)

# Split the data matrix into train and test dataset and Print the shape of train dataset and test dataset.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle = True, random\_state = 123)

print("X\_train shape: {}".format(X\_train.shape))

print("X\_test shape: {}".format(X\_test.shape))

print("y\_train shape: {}".format(y\_train.shape))

print("y\_test shape: {}".format(y\_test.shape))

# split randomly the data matrix into training dataset and validation dataset

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.2, random\_state=1)

# Determine the number of input features

n\_features = X\_train.shape[1]

# Define model

model = keras.Sequential([

keras.layers.Flatten(input\_shape=(n\_features,)),

keras.layers.Dense(4, activation=tf.nn.relu),

keras.layers.Dense(4, activation=tf.nn.relu),

keras.layers.Dense(1, activation=tf.nn.sigmoid),

])

# Compile the model

model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])

# Fit the model

history = model.fit(X\_train, y\_train, epochs=34, batch\_size=32, verbose=0, validation\_data=(X\_val, y\_val))

# Evaluate the model and print the accuracy

loss, acc = model.evaluate(X\_test, y\_test, verbose=0)

print('Test Accuracy: %.3f' % acc)

# Visualize 'Training vs. Validation loss'

loss\_train = history.history['loss']

loss\_val = history.history['val\_loss']

epochs = range(1,35)

plt.plot(epochs, loss\_train, 'g', label='Training loss')

plt.plot(epochs, loss\_val, 'b', label='validation loss')

plt.title('Training vs. Validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

# Visualize 'Training Accuracy vs. Validation accuracy'

loss\_train = history.history['accuracy']

loss\_val = history.history['val\_accuracy']

epochs = range(1,35)

plt.plot(epochs, loss\_train, 'g', label='Training accuracy')

plt.plot(epochs, loss\_val, 'b', label='validation accuracy')

plt.title('Training vs. Validation accuracy')

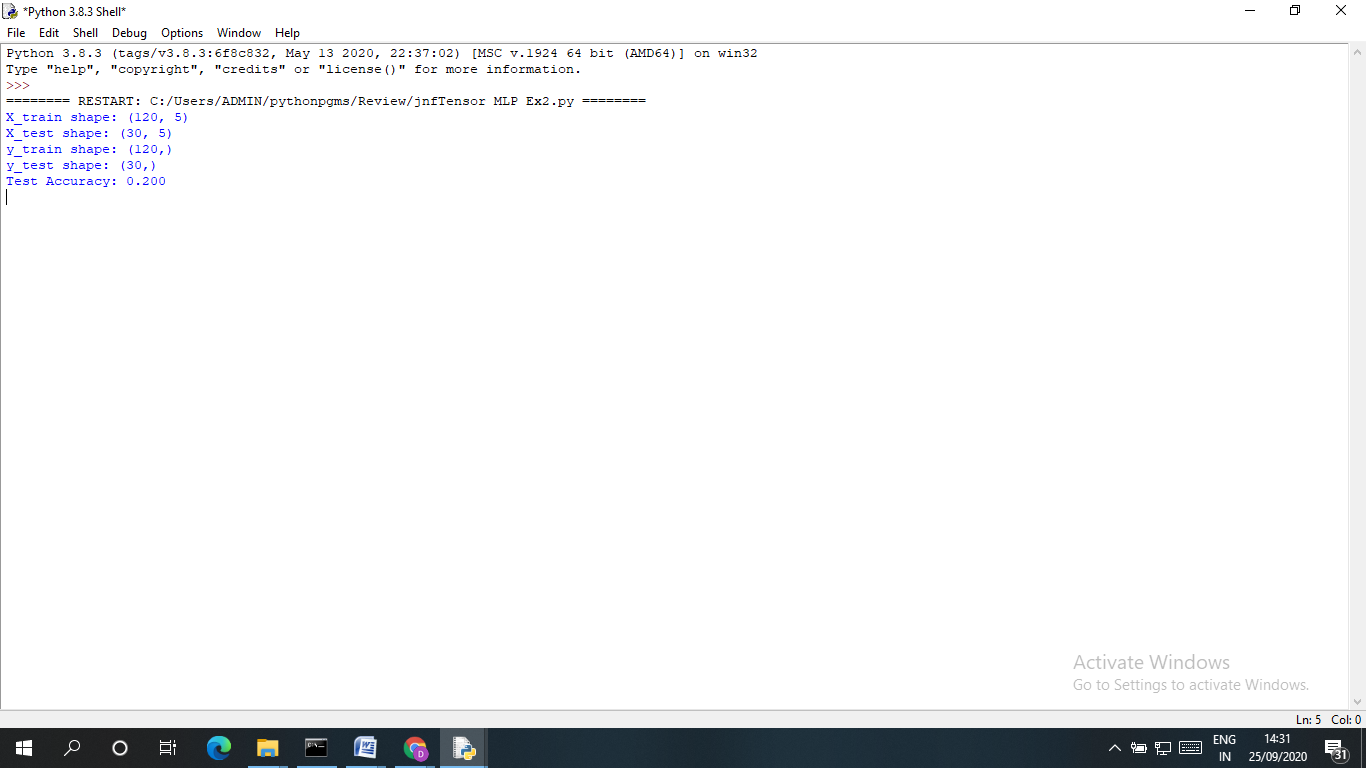
plt.xlabel('Epochs')

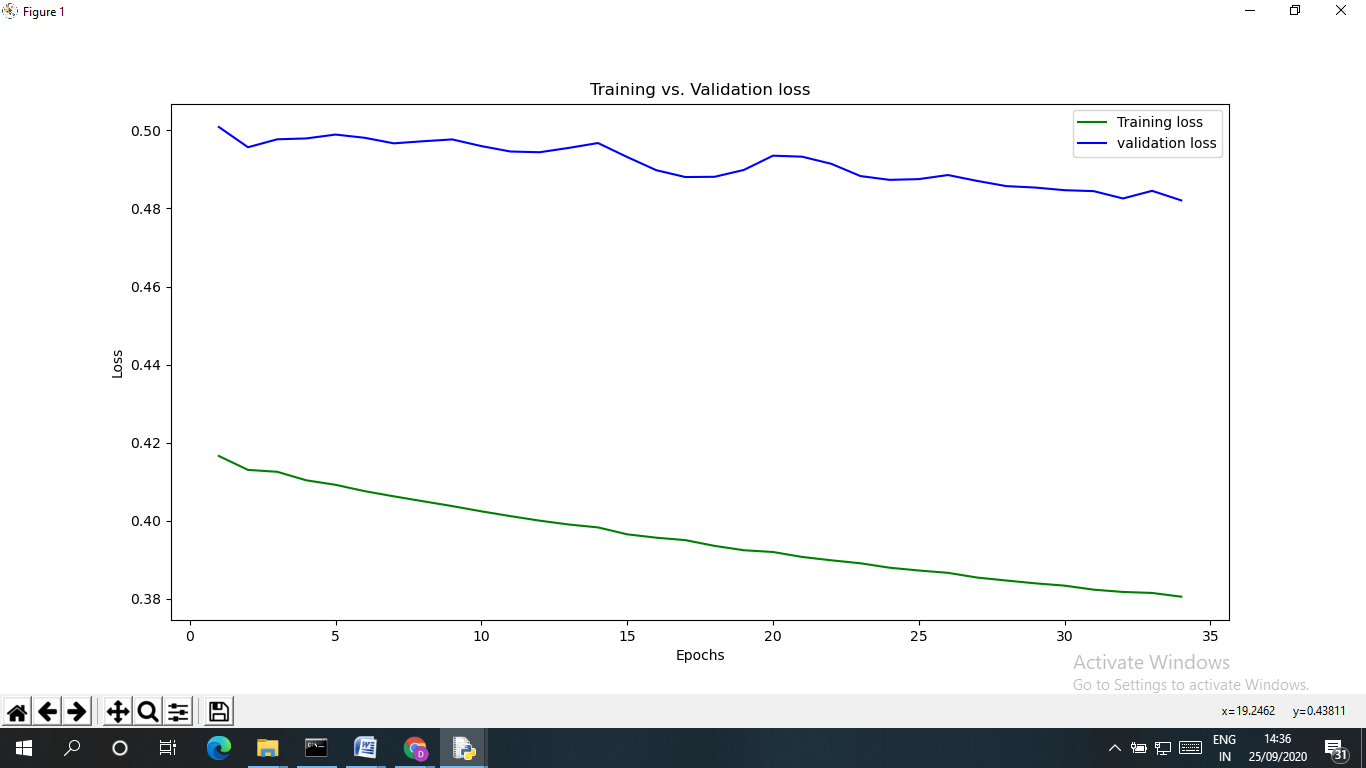
plt.ylabel('Accuracy')

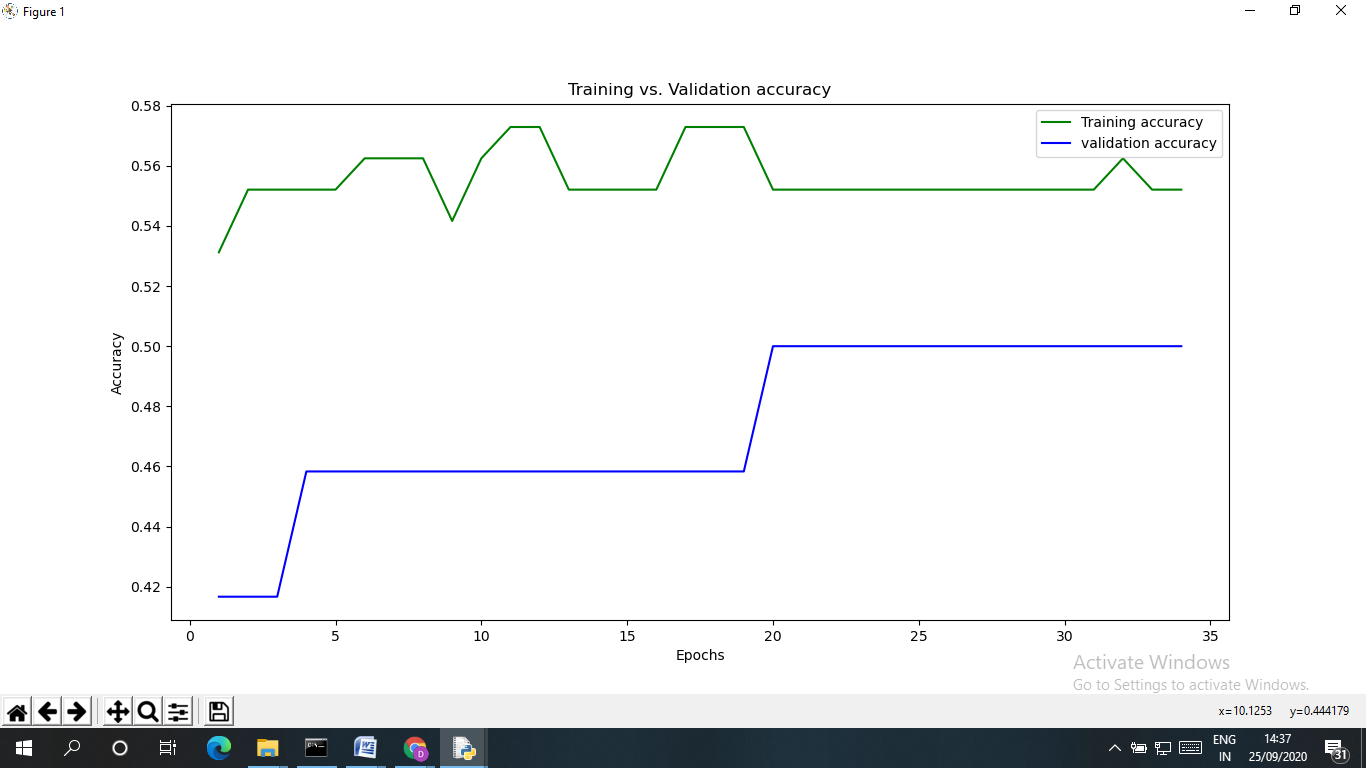
plt.legend()

plt.show()

**Output:**

****

****

****

**Programming Assignment:**

Let us now look at an example of learning in a Multi –Layer Perceptron. The given MLP consists of an Input layer, one Hidden layer and an Output layer. Input layer has 3neurons, Hidden layer has 2 neurons and a single neuron in the Output layer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X1 | X2 | X3 | ODesired |  |
| 1 | 1 | 0 | 1 |

Learning rate: =0.6

The weights and biases are tabulated in Table 17.1.

**Table 17.1: Weights and Biases**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X1 | X2 | X3 | W14 | W15 | W24 | W25 | W34 | W35 | W46 | W56 |  |  |  |
| 1 | 1 | 0 | 0.2 | 0.3 | -0.1 | 0.2 | 0.3 | -0.4 | -0.3 | 0.1 | 0.1 | 0.3 | -0.3 |