LAB EXERCISE – 8

(03-08-2022)

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.

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+hiddenLa
yerNeurons+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+outputLa
yerNeurons+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)
```

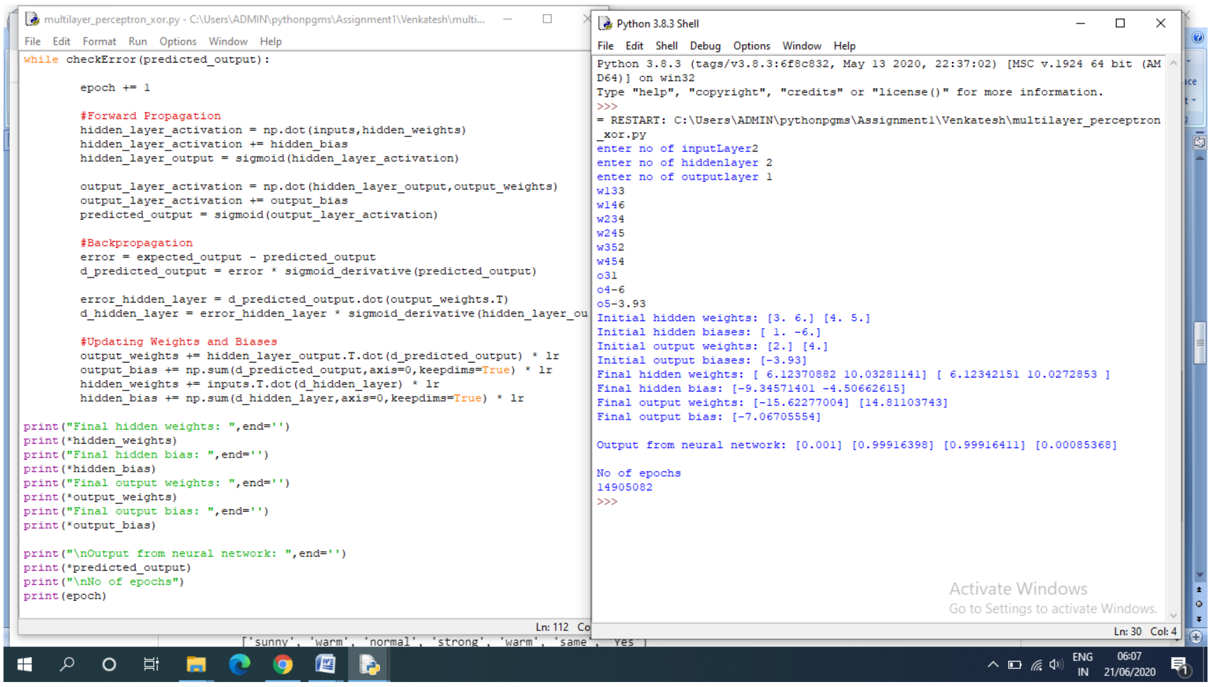
```
        #Updating Weights and Biases
        output_weights += hidden_layer_output.T.dot(d_predicted_output) * lr
        output_bias += np.sum(d_predicted_output,axis=0,keepdims=True) * lr
        hidden_weights += inputs.T.dot(d_hidden_layer) * lr
        hidden_bias += np.sum(d_hidden_layer,axis=0,keepdims=True) * 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\myself\
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:

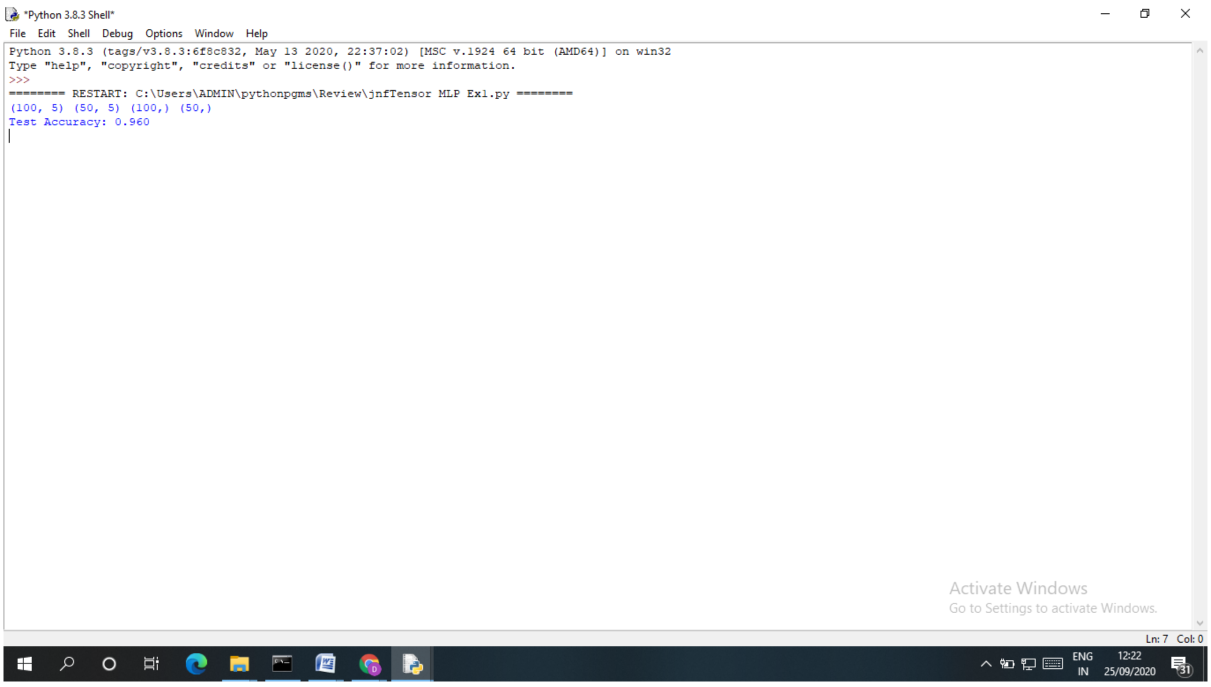


multilayer_perceptron_xor.py - C:\Users\ADMIN\pythonpgms\Assignment1\Venkatesh\multi...
File Edit Format Run Options Window Help
Ln: 112 Co
Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Activate Windows
Go to Settings to activate Windows.
Ln: 30 Col: 4
ENG IN
06:07 21/06/2020

```python
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_ou

        #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)
```

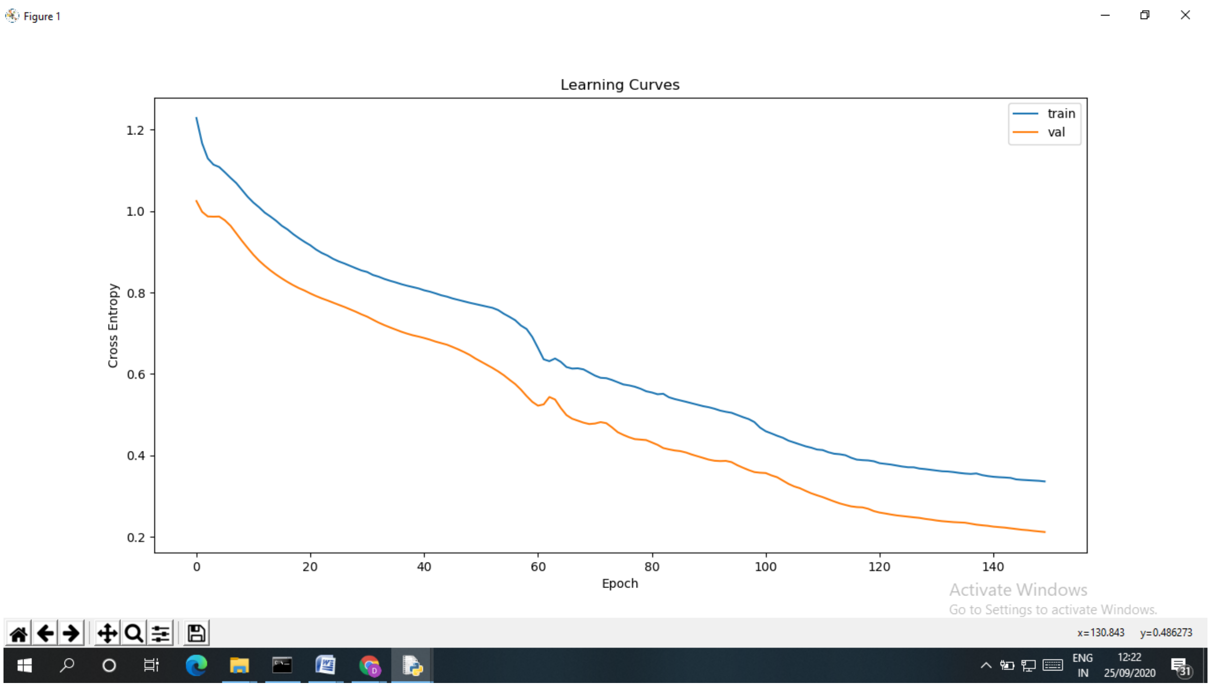
```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\ADMIN\pythonpgms\Assignment1\Venkatesh\multilayer_perceptron
_xor.py
enter no of inputLayer2
enter no of 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
>>>
```

```
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'Yes']
```



```
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
```

Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Activate Windows
Go to Settings to activate Windows.
Ln: 7 Col: 0
ENG IN
12:22
25/09/2020



Refer to the following URL on the internet for code examples from stephen marsland book

https://homepages.ecs.vuw.ac.nz/~marslast/MLbook.html