BUBT | BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY

Committed to Academic Excellence

Lab Test

Course Code: CSE 478

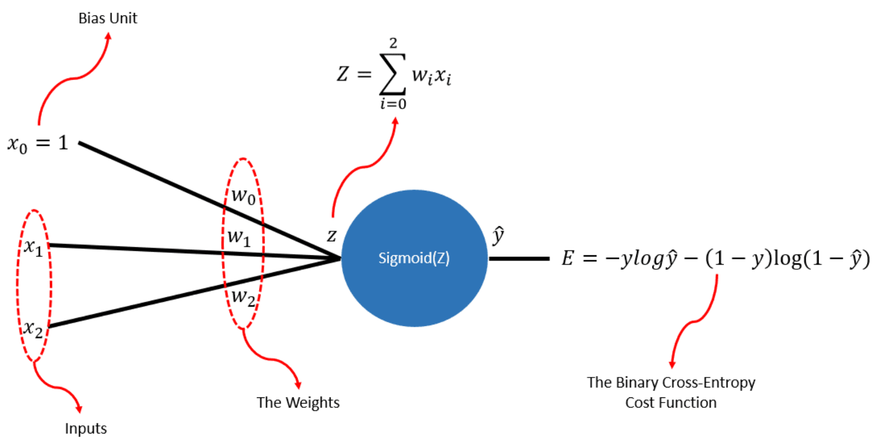
Course Title: Neural Network and Fuzzy Systems & Lab

Submitted to:	Submitted by:
Name: Mr.T.M. Amir - Ul - Haque Bhuiyan	Name: Syeda Nowshin Ibnat
Assistant Professor	ID: 17183103020
Department of Computer Science & Engineering	Intake: 39
at Bangladesh University of Business and Technology.	Section: 02
	Program: B.Sc. in CSE
	Semester: Fall 2021-2022

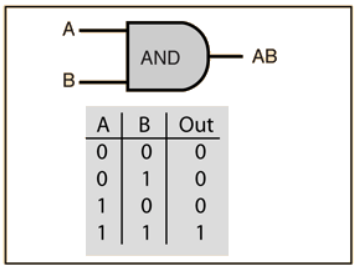
Date of Submission: 05-04-2022

Problem: Perceptron Learning (logic AND gate).

Solution: Of all the gates that are linearly separable, here we have chosen the AND gate. First foremost, let’s take a look at our perceptron:


Bias Unit
$Z = \sum_{i=0}^{2} w_i x_i$
$x_0 = 1$
w_0
x_1
w_1
z
Sigmoid(Z)
$\hat{y}$
$E = -y log\hat{y} - (1-y)\log(1-\hat{y})$
w_2
x_2
The Binary Cross-Entropy Cost Function
The Weights
Inputs

Here we have used the sigmoid() function, instead of a step function. The reason is that during back-propagation, we need all the functions that we have used during the forward pass to be differentiable! Now, the step function is not differentiable as it breaks at point 0 (when its input is 0)! So, I have used the sigmoid() function, which is differentiable all across and is continuous. From our knowledge of logic gates, we know that an AND logic table is given by the diagram below.



A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

Figure: AND Gate

Code:

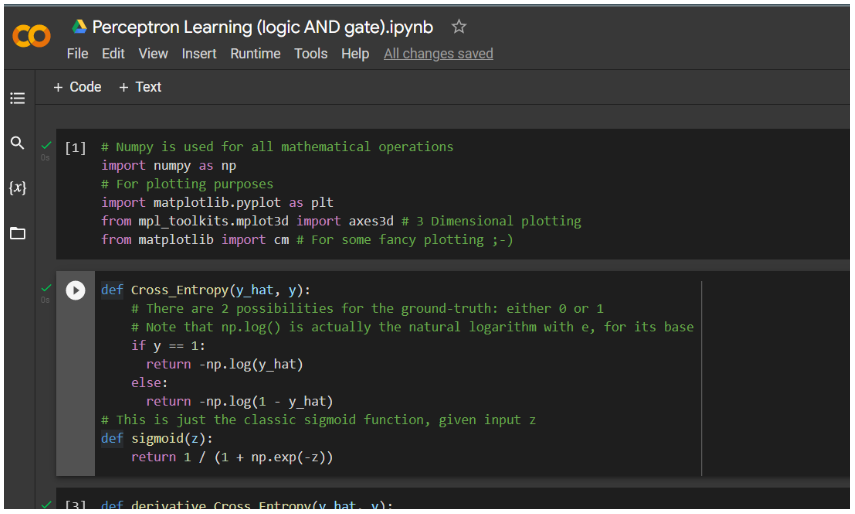
# Numpy is used for all mathematical operations

import numpy as np

```
# For plotting purposes
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d # 3 Dimensional plotting
from matplotlib import cm # For some fancy plotting ;-)
def Cross_Entropy(y_hat, y):
# There are 2 possibilities for the ground-truth: either 0 or 1
# Note that np.log() is actually the natural logarithm with e, for its base
if y == 1:
return -np.log(y_hat)
else:
return -np.log(1 - y_hat)
# This is just the classic sigmoid function, given input z
def sigmoid(z):
return 1 / (1 + np.exp(-z))
def derivative_Cross_Entropy(y_hat, y):
# Again we account for 2 possibilities of y=0/1
if y == 1:
return -1/y_hat
else:
return 1 / (1 - y_hat)
# The derivative of sigmoid is quite straight-forward
def derivative_sigmoid(x):
return x*(1-x)
# Our data
X = np.array([[0, 0], [0, 5], [5, 0], [5, 5]])
# The ground truth (i.e., what AND returns and our perceptron should learn to produce)
Y = np.array([0, 0, 0, 1])
area = 200
```

```
fig = plt.figure(figsize=(6, 6))
plt.title('The AND Gate', fontsize=20)
ax = fig.add_subplot(111)
# color red: is class 0 and color blue is class 1.
ax.scatter(0, 0, s=area, c='r', label="Class 0")
ax.scatter(0, 5, s=area, c='r', label="Class 0")
ax.scatter(5, 0, s=area, c='r', label="Class 0")
ax.scatter(5, 5, s=area, c='b', label="Class 1")
plt.grid()
plt.show()
```

Input Snapshot:

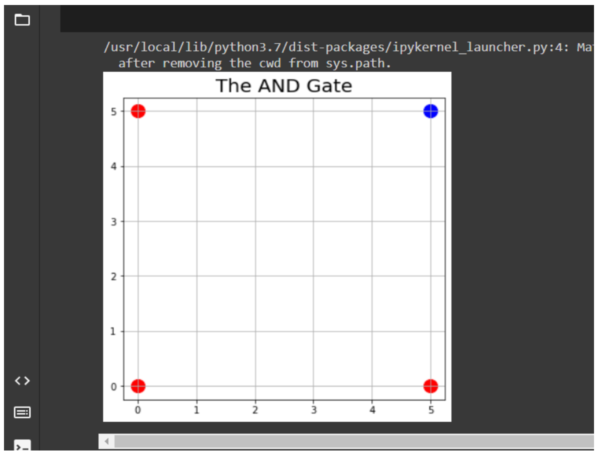

Perceptron Learning (logic AND gate).ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

```
[1]  # Numpy is used for all mathematical operations
     import numpy as np
     # For plotting purposes
     import matplotlib.pyplot as plt
     from mpl_toolkits.mplot3d import axes3d # 3 Dimensional plotting
     from matplotlib import cm # For some fancy plotting ;-)
```

```
def Cross_Entropy(y_hat, y):
    # There are 2 possibilities for the ground-truth: either 0 or 1
    # Note that np.log() is actually the natural logarithm with e, for its base
    if y == 1:
      return -np.log(y_hat)
    else:
      return -np.log(1 - y_hat)
# This is just the classic sigmoid function, given input z
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

```
[3]  def derivative_Cross_Entropy(y_hat, y):
```

Output Snapshot:



```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: Ma
  after removing the cwd from sys.path.
```