NAME: MOZEB AHMED KHAN

ROLL NO: 20F-0161

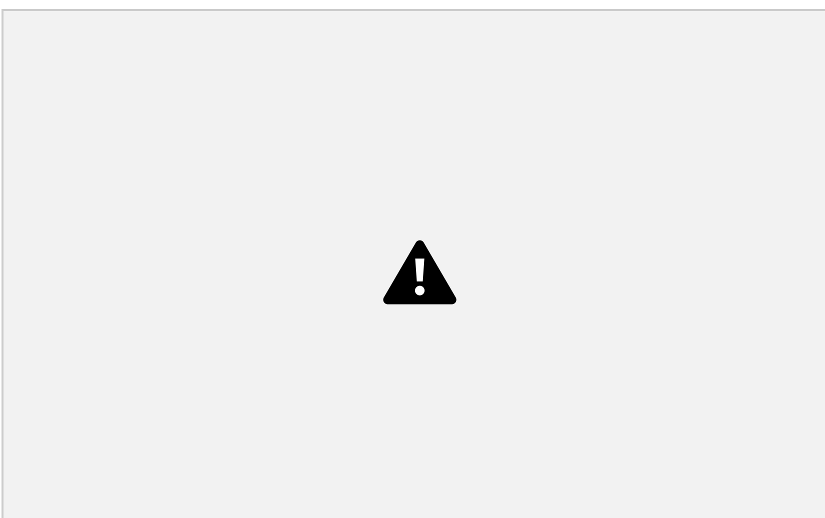
SECTION: BS(CS)-6A

COURSE: ARTIFICIAL INTELLIGENCE

ASSIGN NO: 06

<u>Question No: 01</u>

Consider the neural network architecture for XAND function given below:



1. Assume that the activation function is the sigmoid function. Initialize all the weights with 0.1. Write down the values of the weights after the first and the second iteration of Backpropagation algorithm run with the following examples:

X1 X2 Y

0	0	1
1	0	0
0	1	0
1	1	1

Solution:

2. Consider a neural network that uses a function tanh instead of the sigmoid function. What will be the values of the weights after the first and second iteration in such a case?

Question No: 02

Code:

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

#Initializing the weights randomly and biases as 0.1
np.random.seed(42)
Weights = {
    'W1': np.random.randn(),
    'W2': np.random.randn(),
    'W3': np.random.randn(),
    'W4': np.random.randn(),
    'W5': np.random.randn(),
    'W6': np.random.randn()
}
Bias_values = {
    'b1': 0.1,
    'b2': 0.1,
    'b3': 0.1
}


#Forward Propagating!
def forwardPropagating(x1, x2):
    n1 = x1 * Weights['W1'] + x2 * Weights['W3'] + Bias_values['b1']
    n2 = x1 * Weights['W2'] + x2 * Weights['W4'] + Bias_values['b2']
    yHat = n1 * Weights['W5'] + n2 * Weights['W6'] + Bias_values['b3']
    return yHat


#Back Propagating!
def backwardError(x1, x2, y, learning_rate):
    n1 = x1 * Weights['W1'] + x2 * Weights['W3'] + Bias_values['b1']
    n2 = x1 * Weights['W2'] + x2 * Weights['W4'] + Bias_values['b2']
    yhat = n1 * Weights['W5'] + n2 * Weights['W6'] + Bias_values['b3']

    #Computing gradients!
    gradientW5 = (y - yhat) * n1
    gradientW6 = (y - yhat) * n2
```

```python
    gradientW1 = (y - yhat) * Weights['W5'] * x1
    gradientW2 = (y - yhat) * Weights['W6'] * x1
    gradientW3 = (y - yhat) * Weights['W5'] * x2
    gradientW4 = (y - yhat) * Weights['W6'] * x2
    gradientB1 = (y - yhat) * Weights['W5']
    gradientB2 = (y - yhat) * Weights['W6']
    gradientB3 = (y - yhat)

    #Updating weights and biases!
    Weights['W5'] += learning_rate * gradientW5
    Weights['W6'] += learning_rate * gradientW6
    Weights['W1'] += learning_rate * gradientW1
    Weights['W2'] += learning_rate * gradientW2
    Weights['W3'] += learning_rate * gradientW3
    Weights['W4'] += learning_rate * gradientW4
    Bias_values['b1'] += learning_rate * gradientB1
    Bias_values['b2'] += learning_rate * gradientB2
    Bias_values['b3'] += learning_rate * gradientB3


#Training our Artificial Neural Network!
def trainingXANDAnn(X, y, epochs, learning_rate):
    errors = []
    for epoch in range(epochs):
        error = 0
        for i in range(len(X)):
            x1, x2 = X[i]
            y_true = y[i]

            #Forward Propagating!
            y_pred = forwardPropagating(x1, x2)

            #Back Propagating!
            backwardError(x1, x2, y_true, learning_rate)
            # Calculate the error
            error += abs(y_pred - y_true)

        #Appending the avg. error for epoch!
        errors.append(error / len(X))

    return errors

#Truth Table Data!
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([1, 1, 1, 0])

#Epoch and Learning rate provided!
NoOfEpochs = 100
```

```python
learning_rate = 0.1

#Training our Neural Network!
errors = trainingXANDAnn(X, y, NoOfEpochs, learning_rate)

#Plotting Graph for the errors by highlighting data points!
plt.plot(range(1, NoOfEpochs + 1), errors, marker='o', linestyle='-',
color='green', markersize=5)
plt.xlabel('Epoch#')
plt.ylabel('Error')
plt.title('Graph for 100-Epochs and their Errors!')
plt.grid(True)
plt.show()
```

Output: