Lab 2

Daniel Mehta

Write a code to train the following architecture to learn XOR using Backpropagation.

- That has 2 inputs
- One hidden layer that consist of 2 or more neurons
- One output layer

In [4]:
```
import numpy as np
```

In [5]:
```
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [0]])
```

In [6]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def sigmoid_deriv(x):
    return x * (1 - x)
```

In [7]:
```
np.random.seed(5500)

#2 input neurons, 2 hidden neurons, 1 output neuron
input_size, hidden_size, output_size = 2, 2, 1

#weights and bias for input to hidden layer
W1 = np.random.randn(input_size, hidden_size)
b1 = np.zeros((1, hidden_size))

# wights for hidden to output
W2 = np.random.randn(hidden_size, output_size)
b2 = np.zeros((1, output_size))
```

In [8]:
```
epochs = 10000
lr = 0.1
```

In [9]:
```
for epoch in range(epochs):

    #forward pass
    z1 = np.dot(X, W1) + b1
    a1 = sigmoid(z1)
```

```
    z2 = np.dot(a1, W2) + b2
    a2 = sigmoid(z2)

    #backward pass
    error = y - a2
    d_output = error * sigmoid_deriv(a2)
    d_hidden = np.dot(d_output, W2.T) * sigmoid_deriv(a1)

    #updated weights and biases
    W2 += lr * np.dot(a1.T, d_output)
    b2 += lr * np.sum(d_output, axis=0, keepdims=True)
    W1 += lr * np.dot(X.T, d_hidden)
    b1 += lr * np.sum(d_hidden, axis=0, keepdims=True)
```

In [10]:
```
print("Predictions after training:")
print(np.round(a2, 3))
```

```
Predictions after training:
[[0.062]
 [0.946]
 [0.947]
 [0.055]]
```

In [ ]: