import numpy as np

# Sigmoid activation function and its derivative

def sigmoid(x):

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

def sigmoid\_derivative(x):

return x \* (1 - x)

# XOR Input and Output

X = np.array([[0, 0],

[0, 1],

[1, 0],

[1, 1]])

# Expected XOR Output

y = np.array([[0],

[1],

[1],

[0]])

# Set seed for reproducibility

np.random.seed(1)

# Initialize weights

input\_layer\_neurons = X.shape[1] # 2 input features

hidden\_neurons = 4 # Number of neurons in hidden layer

output\_neurons = 1 # 1 output

# Random weights and bias initialization

weights\_input\_hidden = np.random.uniform(size=(input\_layer\_neurons, hidden\_neurons))

bias\_hidden = np.random.uniform(size=(1, hidden\_neurons))

weights\_hidden\_output = np.random.uniform(size=(hidden\_neurons, output\_neurons))

bias\_output = np.random.uniform(size=(1, output\_neurons))

# Training the neural network

epochs = 10000

learning\_rate = 0.1

for epoch in range(epochs):

# Forward Propagation

hidden\_layer\_input = np.dot(X, weights\_input\_hidden) + bias\_hidden

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

output\_layer\_input = np.dot(hidden\_layer\_output, weights\_hidden\_output) + bias\_output

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

# Backpropagation

error = y - predicted\_output

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

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

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

# Updating Weights and Biases

weights\_hidden\_output += hidden\_layer\_output.T.dot(d\_predicted\_output) \* learning\_ra\_\*

OUTPUT:

