import numpy as np

def relu(x):

    return np.maximum(0, x)

def relu\_derivative(x):

    return np.where(x > 0, 1, 0)

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

    return x \* (1 - x)

# Example input (flattened image vectors)

X = np.array([[0.2, 0.4, 0.6, 0.8],  # Dog image example (simplified)

              [0.3, 0.5, 0.7, 0.9]]) # Cat image example (simplified)

# Example output (1 for dog, 0 for cat)

y = np.array([[1], [0]])

# Network architecture

input\_layer\_size = 4  # Number of features

hidden\_layer1\_size = 6  # First hidden layer neurons

hidden\_layer2\_size = 5  # Second hidden layer neurons

hidden\_layer3\_size = 4  # Third hidden layer neurons

output\_layer\_size = 1  # Binary classification

# Fixed weights and biases

W1 = np.array([[0.1, 0.2, 0.3, 0.4, 0.5, 0.6],

               [0.7, 0.8, 0.9, 1.0, 1.1, 1.2],

               [1.3, 1.4, 1.5, 1.6, 1.7, 1.8],

               [1.9, 2.0, 2.1, 2.2, 2.3, 2.4]])

b1 = np.array([[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]])

W2 = np.array([[0.2, 0.3, 0.4, 0.5, 0.6],

               [0.7, 0.8, 0.9, 1.0, 1.1],

               [1.2, 1.3, 1.4, 1.5, 1.6],

               [1.7, 1.8, 1.9, 2.0, 2.1],

               [2.2, 2.3, 2.4, 2.5, 2.6],

               [2.7, 2.8, 2.9, 3.0, 3.1]])

b2 = np.array([[0.1, 0.2, 0.3, 0.4, 0.5]])

W3 = np.array([[0.2, 0.3, 0.4, 0.5],

               [0.6, 0.7, 0.8, 0.9],

               [1.0, 1.1, 1.2, 1.3],

               [1.4, 1.5, 1.6, 1.7],

               [1.8, 1.9, 2.0, 2.1]])

b3 = np.array([[0.1, 0.2, 0.3, 0.4]])

W4 = np.array([[0.2], [0.3], [0.4], [0.5]])

b4 = np.array([[0.1]])

# Training parameters

learning\_rate = 0.1

epochs = 10000

# Training loop

for epoch in range(epochs):

    # Forward propagation

    z1 = np.dot(X, W1) + b1

    a1 = relu(z1)

    z2 = #YOUR CODE HERE

    a2 = #YOUR CODE HERE

    z3 = #YOUR CODE HERE

    a3 = #YOUR CODE HERE

    z4 = #YOUR CODE HERE

    a4 = #YOUR CODE HERE

    # Compute error

    error = y - a4

    # Backpropagation

    d\_a4 = error \* sigmoid\_derivative(a4)

    d\_W4 = np.dot(a3.T, d\_a4) \* learning\_rate

    d\_b4 = np.sum(d\_a4, axis=0, keepdims=True) \* learning\_rate

    d\_a3 = #YOUR CODE HERE

    d\_W3 = #YOUR CODE HERE

    d\_b3 = #YOUR CODE HERE

    d\_a2 = #YOUR CODE HERE

    d\_W2 = #YOUR CODE HERE

    d\_b2 = #YOUR CODE HERE

    d\_a1 = #YOUR CODE HERE

    d\_W1 = #YOUR CODE HERE

    d\_b1 = #YOUR CODE HERE

    # Update weights and biases

    W4 += d\_W4

    b4 += d\_b4

    W3 += d\_W3

    b3 += d\_b3

    W2 += d\_W2

    b2 += d\_b2

    W1 += d\_W1

    b1 += d\_b1

    # Print loss every 1000 epochs

    if epoch % 1000 == 0:

        loss = np.mean(np.abs(error))

        print(f"Epoch {epoch}, Loss: {loss}")

# Final predictions

y\_pred = a4