import numpy as np

def leaky\_relu(x, alpha=0.01):

return np.where(x > 0, x, alpha \* x)

def leaky\_relu\_derivative(x, alpha=0.01):

return np.where(x > 0, 1, alpha)

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([[0.15, 0.3, 0.45, 0.6], # Dog image example (simplified)

[0.2, 0.4, 0.6, 0.8]]) # 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 = 8 # First hidden layer neurons

hidden\_layer2\_size = 6 # Second hidden layer neurons

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

hidden\_layer4\_size = 3 # Fourth 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],

[2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2]])

b1 = np.array([[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]])

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],

[3.2, 3.3, 3.4, 3.5, 3.6, 3.7],

[3.8, 3.9, 4.0, 4.1, 4.2, 4.3],

[4.4, 4.5, 4.6, 4.7, 4.8, 4.9]])

b2 = np.array([[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]])

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],

[2.2, 2.3, 2.4, 2.5]])

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

W4 = 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]])

b4 = np.array([[0.1, 0.2, 0.3]])

W5 = np.array([[0.2], [0.3], [0.4]])

b5 = np.array([[0.1]])

# Training parameters

learning\_rate = 0.1

epochs = 10000

# Training loop

for epoch in range(epochs):

# Forward propagation

z1 =

a1 = leaky\_relu(z1)

z2 =

a2 = leaky\_relu(z2)

z3 =

a3 = leaky\_relu(z3)

z4 =

a4 = leaky\_relu(z4)

z5 =

a5 = sigmoid(z5)

# Compute error

error = y - a5

# 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 = a5

print("Final Predictions:", y\_pred)