# Tiny 2-layer NN for XOR using numpy

import numpy as np

np.random.seed(0)

X = np.array([[0,0],[0,1],[1,0],[1,1]], dtype=float)

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

def sigmoid(z): return 1/(1+np.exp(-z))

def dsigmoid(a): return a\*(1-a)

# Architecture: 2 -> 4 -> 1

W1 = np.random.randn(2,4)\*0.5

b1 = np.zeros((1,4))

W2 = np.random.randn(4,1)\*0.5

b2 = np.zeros((1,1))

lr = 0.5

for epoch in range(10000):

# forward

z1 = X@W1 + b1

a1 = sigmoid(z1)

z2 = a1@W2 + b2

a2 = sigmoid(z2) # predictions

# loss (MSE)

loss = np.mean((a2 - y)\*\*2)

# backprop

dA2 = 2\*(a2 - y)/y.shape[0]

dZ2 = dA2 \* dsigmoid(a2)

dW2 = a1.T @ dZ2

dB2 = dZ2.sum(axis=0, keepdims=True)

dA1 = dZ2 @ W2.T

dZ1 = dA1 \* dsigmoid(a1)

dW1 = X.T @ dZ1

dB1 = dZ1.sum(axis=0, keepdims=True)

# gradient descent

W2 -= lr\*dW2; b2 -= lr\*dB2

W1 -= lr\*dW1; b1 -= lr\*dB1

if epoch % 2000 == 0:

print(f"epoch {epoch:5d} loss={loss:.6f}")

print("Predictions (rounded):", np.round(a2.T, 3))