Ann2

**import** numpy **as** np

def sigmoid(x):

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

def sigmoid\_derivative(x):

return x \* (1 - x)

X = np.array([

[0, 0],

[0, 1],

[1, 0],

[1, 1]

])

# Expected output for XOR

Y = np.array([

[0],

[1],

[1],

[0]

])

np.random.seed(42)

input\_size = 2

hidden\_size = 2

output\_size = 1

learning\_rate = 0.5

epochs = 10000

w\_input\_hidden = np.random.uniform(-1, 1, (input\_size, hidden\_size))

w\_hidden\_output = np.random.uniform(-1, 1, (hidden\_size, output\_size))

bias\_hidden = np.zeros((1, hidden\_size))

bias\_output = np.zeros((1, output\_size))

for epoch in range(epochs):

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

hidden\_output = sigmoid(hidden\_input)

final\_input = np.dot(hidden\_output, w\_hidden\_output) + bias\_output

final\_output = sigmoid(final\_input)

error = Y - final\_output

d\_output = error \* sigmoid\_derivative(final\_output)

d\_hidden = d\_output.dot(w\_hidden\_output.T) \* sigmoid\_derivative(hidden\_output)

w\_hidden\_output += hidden\_output.T.dot(d\_output) \* learning\_rate

bias\_output += np.sum(d\_output, axis=0, keepdims=True) \* learning\_rate

w\_input\_hidden += X.T.dot(d\_hidden) \* learning\_rate

bias\_hidden += np.sum(d\_hidden, axis=0, keepdims=True) \* learning\_rate

if epoch % 1000 == 0:

loss = np.mean(np.square(error))

print(f"Epoch {epoch}, Loss: {loss:.4f}")

print("\nTrained XOR results:")

for i in range(len(X)):

print(f"Input: {X[i]} => Output: {np.round(final\_output[i][0])} (raw: {final\_output[i][0]:.4f})")