```
import numpy as np
          # Sigmoid and derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
          def sigmoid_derivative(x):
    return x * (1 - x)
          # XOR dataset
X = np.array([[0,0],
                                         [0,1],
[1,0],
[1,1]])
           Y = np.array([[0],
          # Set random seed np.random.seed(42)
          input_size = 2
hidden_size = 2
          # Initialize weights and biases
W1 = np.random.randn(input_size, hidden_size)
b1 = np.zeros((1, hidden_size))
W2 = np.random.randn(hidden_size, output_size)
b2 = np.zeros((1, output_size))
          # Hyperparameters
lr = 0.1
epochs = 10000
          # Training
for epoch in range(epochs + 1):
    # Forward pass
Z1 = np.dot(X, W1) + b1
                   A1 = sigmoid(Z1)
                   Z2 = np.dot(A1, W2) + b2
A2 = sigmoid(Z2)
                   # Compute loss
loss = np.mean((Y - A2) ** 2)
                   # Print every 1000 epochs
if epoch % 1000 == 0:
                          print(f"Epoch {epoch}, Loss : {loss:.4f}")
                   # backpropagation

dA2 = (A2 - Y)

dZ2 = dA2 * sigmoid_derivative(A2)

dW2 = np.dot(A1.T, dZ2)

dB2 = np.sum(dZ2, axis=0, keepdims=True)
                   dA1 = np.dot(dZ2, WZ.T)
dZ1 = dA1 * sigmoid_derivative(A1)
dW1 = np.dot(X.T, dZ1)
dB1 = np.sum(dZ1, axis=0, keepdims=True)
                  W1 -= lr * dW1
b1 -= lr * dB1
W2 -= lr * dW2
b2 -= lr * dB2
           # Final predictions
print("\nFinal predictions:")
print(A2)
```

```
Epoch 0, Loss: 0.2558
Epoch 1000, Loss: 0.2494
Epoch 2000, Loss: 0.2454
Epoch 3000, Loss: 0.2047
Epoch 4000, Loss: 0.1532
Epoch 5000, Loss: 0.1387
Epoch 6000, Loss: 0.1336
Epoch 7000, Loss: 0.1312
Epoch 8000, Loss: 0.1297
Epoch 9000, Loss: 0.1288
Epoch 10000, Loss: 0.1282
Final predictions:
[[0.05300376]
 [0.49554286]
 [0.95091752]
 [0.50319846]]
```

To implement Gradient Descent and Backpropagation for training a simple feed forward neural network on the XOR problem.

Description:

* Gradient Descent & an optimization algorithm used to minimize the loss function by updating weights in the opposite direction of the gradient

* Backpropagation is the process of calculating the gradient of the loss function with respect to each weight using the chain rule, so we can apply gradient descent efficiently.

The steps are:

- 1.) Forward pass Compute output from inputs
- 2) Compute lors Différence between predicted and actual values.
- 3) Backward Pars Compute gradients of Lors wirt weights.
- 4) Update weights Use Gradient Descent rule:

m= m- 2. de

where of is the learning rate.

Procedure:
1.) Initialize weights randomly
Output:
-> Compute the hidden layer
3.) Compute lors: Mean Squared etror between predicted output and actual output
D. C. Linni
5) 1. Peight update.
update weights and spia Jood & 13093
Descent w= 25 + 25 36 1, 0000 A3093
6) Repeat steps 2-5 for several reports antil the glob loss converges. 1) Print trial predictions of the training Noop Noop
CODE:
smoot numpy as no [88800820.0]
(Simple Corp. of
def sigmoid - derivative [2) ? [1803.07]
return X*(1-X)
x = np. auray ([[0,0], [0,1], [1,0], [1,1]])
y = np. astrong CLES , 1 = 1
np. random. seed (42)
Input-size = 2

hidden - site = 2 output-size=1 w = np. random randa (input-size, hidden-size) bi = np. zeros ((1. hidden-size)) W2 = np. random. randn (hidden-size, output-size) ba = np. zeros (C1, output-size)) [r=0.1 epochs = 1000 for epoch in range (epochs): == np. dot (x, w1) + b1 al = sigmoid (21) 72= np.dot (a1, w2)+b2 a2 = sigmoid (22) lors = np. mean ((y-a2)**2) d-a2 = (d2-y) d-z2 = d-a2 * sigmoid - derivative (a2) d-w2 = np. dot (a1. T, d-22) d-62 = np. sum(d-22, axis=0, keep dims = True) d-a1 = np. dot (d-22, w2.T) d-Z1 = d-a1 * sigmoid-derivative(a1) dw, = np. dot (x.T, d-Z1) d-b1 = np. sum (d-z1, aris =0, keepdims = True) wz= Lr * dwz w1 = lx *d-101 Next line b2= Lr # d- b2 b2 == | +d-b1 if epoch % 1000 == 0; N/L print (f"Epoch Zepoch 3, loss (Sloss: %f 3") N/L print (" In final prediction) MP(a) Result: The Code has been successfully executed and shows the network learned the XOX Logic output near 0 for [0,0] and [1,1] and near 1 for 20,17 and [1,07

i orubani Jackie lie en eights randonly Frank Pars Output: Epoch o repair nobbid 2015.

Epoch o repair nobbid 2015.

Epoch o repair nobbid 2015. -> Compute -> Compute Epoch 1000, loss: 0.2494 mande soil stuges con stuges con 2454 parques con 2000 parques con 2454 parques con 2554 parques con Epoch 3000, loss: 0.2047 i Eukpropagation: Epoch 4000, loss: 0.1532 i weight update: update weights
Descript Epoch 5000, Loss: 0:13870 Epoch 6000, lossie 0, 1336 ... Epoch: 17000 al dors do 1312 Epoch 8000, losse 0:1297 represented Epoch 9000 inion lossil o 1288 ibang boint thing Final Predictions: : 190 [[0.02300898] de ca bedume produi [0.4955 4213] : (12) Biompie jik return 1/ (14 mp. c. [1818] 1902 p. 0.3 [0.5081988].] ovilavireb - biompie jeb return X (1-X) X = np-cosnay ([[0,0],[0],[0]]) [1,0] [1,1]) ([[0], [1], [1], [0]) pioneo .qn. = [p. random seed (42)