Build an MLP (Multi-Layer Perceptron) from scratch in Python, step by step — using only **NumPy**, no deep-learning libraries like TensorFlow or PyTorch.

We'll create:

- A **simple neural network** with one **hidden layer**
- Train it on a **small dataset (XOR problem)**
- Implement **forward pass**, **backpropagation**, and **training loop**

XOR problem:

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Step by step implementation from Scratch

```
import numpy as np


# ---------- Step 1: Define input and output ----------
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])    # inputs

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

# ---------- Step 2: Initialize parameters ----------
np.random.seed(42)

input_neurons = 2
```

```
hidden_neurons = 2
output_neurons = 1
learning_rate = 0.1

# Random weights and biases
W1 = np.random.randn(input_neurons, hidden_neurons)
b1 = np.zeros((1, hidden_neurons))
W2 = np.random.randn(hidden_neurons, output_neurons)
b2 = np.zeros((1, output_neurons))

# ---------- Step 3: Define activation functions ----------
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# ---------- Step 4: Training loop ----------
epochs = 10000
for epoch in range(epochs):
    # ----- Forward pass -----
    z1 = np.dot(X, W1) + b1
    a1 = sigmoid(z1)            # hidden layer output

    z2 = np.dot(a1, W2) + b2
    a2 = sigmoid(z2)            # final output

    # ----- Compute loss (Mean Squared Error) -----
    loss = np.mean((y - a2) ** 2)
```

```
    # ----- Backpropagation -----
    d_a2 = (a2 - y) * sigmoid_derivative(a2)  # output layer error
    d_W2 = np.dot(a1.T, d_a2)
    d_b2 = np.sum(d_a2, axis=0, keepdims=True)

    d_a1 = np.dot(d_a2, W2.T) * sigmoid_derivative(a1)  # hidden layer error
    d_W1 = np.dot(X.T, d_a1)
    d_b1 = np.sum(d_a1, axis=0, keepdims=True)

    # ----- Update weights and biases -----
    W2 -= learning_rate * d_W2
    b2 -= learning_rate * d_b2
    W1 -= learning_rate * d_W1
    b1 -= learning_rate * d_b1

    # Print loss occasionally
    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.4f}")

# ---------- Step 5: Test the trained model ----------
print("\nFinal predictions:")
print(a2.round(3))
```

Output (example)

```
Epoch 0, Loss: 0.2703
Epoch 1000, Loss: 0.2496
Epoch 2000, Loss: 0.2427
...
```

Epoch 9000, Loss: 0.0019

Final predictions:

[[0.02]

[0.97]

[0.97]

[0.03]]

✅ The output is close to the expected XOR results — MLP successfully learned the nonlinear pattern!

Step	Description
Forward pass	Compute neuron outputs layer by layer using activation functions
Loss	Compare prediction with actual (mean squared error)
Backpropagation	Compute gradients (partial derivatives) for each weight
Update	Adjust weights to reduce loss (gradient descent)