FORWARD AND BACKWARD PROPAGATION

✅ Improvements:

- Multiple training samples
- Multiple training epochs
- Loss printed each epoch
- Visualizing loss over time (optional)

🧠 Problem Setup:

We'll use a very simple dataset (X, y) and keep:

- 2 inputs
- 1 hidden layer (2 neurons)
- 1 output neuron

📌 Full Code with Multiple Epochs:

```
import numpy as np

import matplotlib.pyplot as plt


# Activation function: Sigmoid

def sigmoid(x):

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


# Derivative of sigmoid

def sigmoid_derivative(x):

  return x * (1 - x)


# Dataset: 4 training examples (XOR-like pattern)

X = np.array([

  [0.05, 0.10],

  [0.10, 0.20],

  [0.20, 0.05],

  [0.30, 0.25]
```

```
])

y_true = np.array([
    [0.01],
    [0.99],
    [0.01],
    [0.99]
])

# Initialize weights and biases with small random values
np.random.seed(42)  # for reproducibility
W1 = np.random.rand(2, 2)
b1 = np.random.rand(1, 2)

W2 = np.random.rand(2, 1)
b2 = np.random.rand(1, 1)

# Hyperparameters
lr = 0.5
epochs = 10000
loss_history = []

# Training loop
for epoch in range(epochs):
    # ---- FORWARD PROPAGATION ----
    hidden_input = np.dot(X, W1) + b1
    hidden_output = sigmoid(hidden_input)

    final_input = np.dot(hidden_output, W2) + b2
    final_output = sigmoid(final_input)
```

```
    # ---- LOSS ----
    loss = np.mean(0.5 * (y_true - final_output) ** 2)
    loss_history.append(loss)

    # ---- BACKPROPAGATION ----
    error_output = final_output - y_true
    delta_output = error_output * sigmoid_derivative(final_output)

    dW2 = np.dot(hidden_output.T, delta_output)
    db2 = np.sum(delta_output, axis=0, keepdims=True)

    error_hidden = np.dot(delta_output, W2.T)
    delta_hidden = error_hidden * sigmoid_derivative(hidden_output)

    dW1 = np.dot(X.T, delta_hidden)
    db1 = np.sum(delta_hidden, axis=0, keepdims=True)

    # ---- UPDATE WEIGHTS ----
    W2 -= lr * dW2
    b2 -= lr * db2
    W1 -= lr * dW1
    b1 -= lr * db1

    # Print loss occasionally
    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.6f}")

# Final predictions
print("\nFinal predictions after training:")
print(final_output)
```

```
# Plot loss curve
plt.plot(loss_history)
plt.title("Loss over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
```

📊 **Output:**

- Prints the loss every 1000 epochs
- Displays a loss curve showing how training improves
- Shows final output after training (should be close to targets)

✅ **What You've Learned:**

- How to implement forward and backward propagation from scratch
- How to train on multiple data samples
- How to observe loss decreasing over time
- How to make predictions from a trained model
