Lab Report: 11

Experiment Title: Forward and Backward Pass in a Simple Feedforward Neural Network

Objectives:

1. To implement a simple feedforward neural network with 2 input neurons, 2 hidden neurons, and 2 output neurons.
2. To perform a forward pass to calculate outputs.
3. To calculate the total error using the Mean Squared Error (MSE) function.
4. To perform a backward pass (backpropagation) for updating weights.

Theoretical Background:

1. A feedforward neural network is a layered structure where inputs pass through hidden layers to produce outputs.
2. Forward Pass: Computes outputs using current weights and biases.
3. Error Calculation: Quantifies the difference between predicted and target outputs using the MSE function.

$$\text{Error} = \frac{1}{2}(\text{Output-}Target)^2$$

4. Backward Pass: Adjusts weights using gradient descent to minimize error.

Sigmoid Function:

$$f(x) = \frac{1}{1+e^{-x}}$$

Algorithm:

Setp:1 Forward Pass: Compute hidden outputs and final outputs using sigmoid.

Step:2 Error: Calculate total error Etotal=$0.5*((t_1 - out_1)^2+(t_2 - out_2)^2)$

Step:3 Backward Pass:

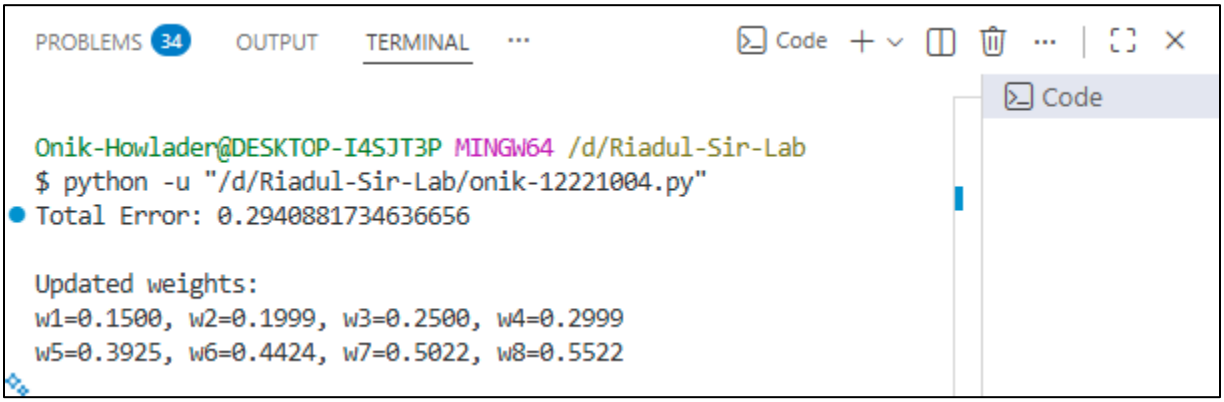
- Compute output layer deltas.
- Update hidden→output weights.
- Compute hidden layer deltas.
- Update input→hidden weights.

Step:4 Output: Print updated weights and error.

Source Code:

```python
import numpy as np

# Inputs
i1, i2 = .05, .10
t1, t2 = .01, .99
b1 = .35
b2 = .60
η = 0.1

# Weights
w1, w2, w3, w4 = .15, .20, .25, .30
w5, w6, w7, w8 = .40, .45, .50, .55

# ---------- Forward Pass ----------
neth1 = (i1*w1) + (i2*w2) + (b1*b1)
outh1 = 1 / (1 + np.exp(-neth1))

neth2 = (i1*w3) + (i2*w4) + (b1*b1)
outh2 = 1 / (1 + np.exp(-neth2))

y1 = (outh1*w5) + (outh2*w6) + b2
outy1 = 1 / (1 + np.exp(-y1))

y2 = (outh1*w7) + (outh2*w8) + b2
outy2 = 1 / (1 + np.exp(-y2))

# ---------- Error ----------
E1 = 0.5 * (t1 - outy1)**2
E2 = 0.5 * (t2 - outy2)**2
E_total = E1 + E2
print("Total Error:", E_total)

# ---------- Backward Pass ----------
delta_outy1 = -(t1 - outy1) * outy1 * (1 - outy1)
delta_outy2 = -(t2 - outy2) * outy2 * (1 - outy2)

# Hidden → Output weight updates
w5 = w5 - η * delta_outy1 * outh1
w6 = w6 - η * delta_outy1 * outh2
w7 = w7 - η * delta_outy2 * outh1
w8 = w8 - η * delta_outy2 * outh2

# Hidden layer deltas
delta_h1 = (delta_outy1*w5 + delta_outy2*w7) * outh1 * (1 - outh1)
delta_h2 = (delta_outy1*w6 + delta_outy2*w8) * outh2 * (1 - outh2)

# Input → Hidden weight updates
w1 = w1 - η * delta_h1 * i1
w2 = w2 - η * delta_h1 * i2
w3 = w3 - η * delta_h2 * i1
w4 = w4 - η * delta_h2 * i2

# ---------- Show new weights ----------
print("\nUpdated weights:")
print(f"w1={w1:.4f}, w2={w2:.4f}, w3={w3:.4f}, w4={w4:.4f}")
print(f"w5={w5:.4f}, w6={w6:.4f}, w7={w7:.4f}, w8={w8:.4f}")

```

Screenshot Of Execution:



Observation and Result:

Given the inputs i1=0.05 and i2=0. with target outputs t1=0.01 and t2=0.99, the network iteratively updated its weights, gradually reducing the total error. The outputs moved closer to the target values with each iteration. The hidden layer adjusted to propagate the error backward effectively, while the sigmoid activation function ensured smooth gradient flow. This demonstrates that the network successfully learned the required mapping through forward and backward passes.