Backpropagation Mathematical Solution

Step 1: Forward Propagation

Given a neural network with:
- Inputs: $x_1 = 0.05$, $x_2 = 0.10$
- Target Outputs: $y_1 = 0.01$, $y_2 = 0.99$
- Initial Weights:
  - Input to Hidden: $w_1 = 0.15$, $w_2 = 0.20$, $w_3 = 0.25$, $w_4 = 0.30$
  - Hidden to Output: $w_5 = 0.40$, $w_6 = 0.45$, $w_7 = 0.50$, $w_8 = 0.55$
- Biases:
  - Hidden Layer: $b_h = 0.35$
  - Output Layer: $b_o = 0.60$

Compute hidden layer activation:
$h_1 = \sigma(x_1w_1 + x_2w_2 + b_h)$
$h_2 = \sigma(x_1w_3 + x_2w_4 + b_h)$

Where $\sigma(z) = 1 / (1 + e^{-z})$ (Sigmoid function).

Compute output layer activation:
$o_1 = \sigma(h_1w_5 + h_2w_6 + b_o)$
$o_2 = \sigma(h_1w_7 + h_2w_8 + b_o)$

Step 2: Compute Error

The error is computed using Mean Squared Error (MSE):
$E = ½ \Sigma (y_i - o_i)^2$

Step 3: Backpropagation (Compute Gradients)

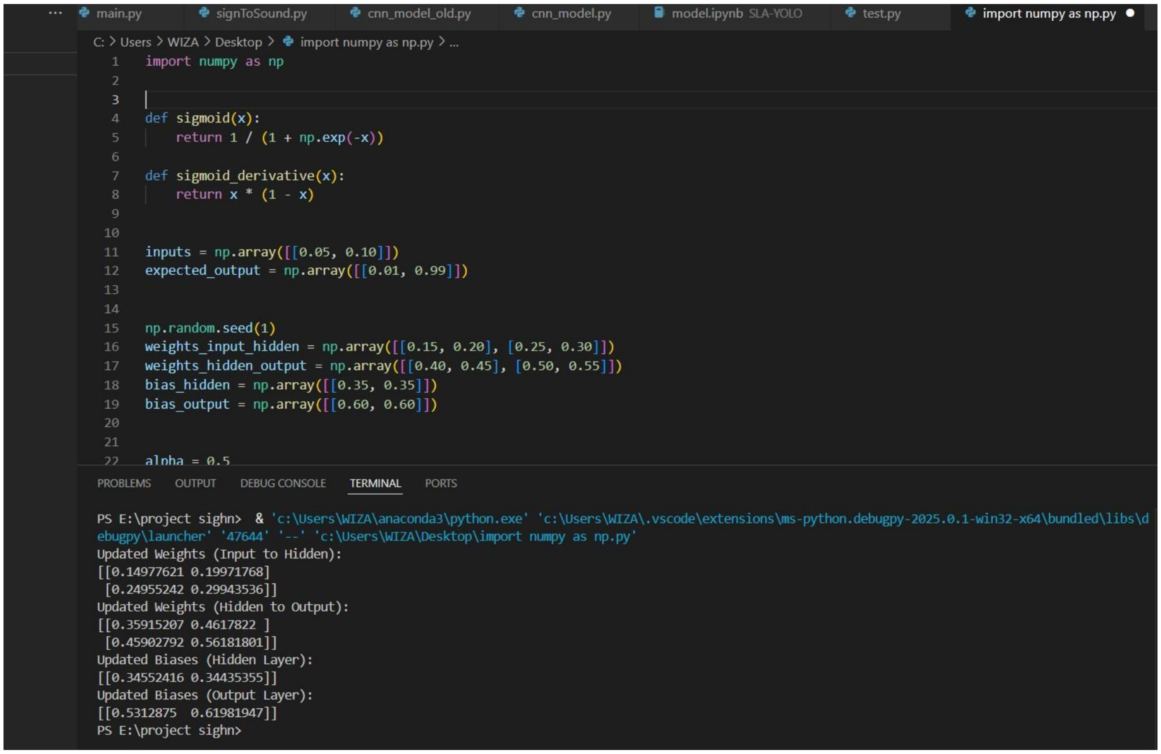
Compute the derivative of error w.r.t. output neuron weights using the chain rule:
$\partial E/\partial w_5 = \partial E/\partial o_1 * \partial o_1/\partial net_1 * \partial net_1/\partial w_5$
Similarly, for w_6, w_7, w_8.

Compute the derivative of error w.r.t. hidden neuron weights:
$\partial E/\partial w_1 = \partial E/\partial h_1 * \partial h_1/\partial net_{h1} * \partial net_{h1}/\partial w_1$
Similarly, for w_2, w_3, w_4.

Step 4: Update Weights

Using Gradient Descent:
$w\_new = w\_old - \eta * \partial E/\partial w$

Where η (eta) is the learning rate.




```
import numpy as np


def sigmoid(x):

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


def sigmoid_derivative(x):

    return x * (1 - x)


inputs = np.array([[0.05, 0.10]])

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


np.random.seed(1)

weights_input_hidden = np.array([[0.15, 0.20], [0.25, 0.30]])
```

```
weights_hidden_output = np.array([[0.40, 0.45], [0.50, 0.55]])
bias_hidden = np.array([[0.35, 0.35]])
bias_output = np.array([[0.60, 0.60]])

alpha = 0.5

hidden_layer_input = np.dot(inputs, weights_input_hidden) + bias_hidden
hidden_layer_output = sigmoid(hidden_layer_input)

output_layer_input = np.dot(hidden_layer_output, weights_hidden_output) +
bias_output
output_layer_output = sigmoid(output_layer_input)

delta_output = (expected_output - output_layer_output) *
sigmoid_derivative(output_layer_output)

delta_hidden = np.dot(delta_output, weights_hidden_output.T) *
sigmoid_derivative(hidden_layer_output)

weights_hidden_output += alpha * np.dot(hidden_layer_output.T, delta_output)
weights_input_hidden += alpha * np.dot(inputs.T, delta_hidden)
bias_output += alpha * delta_output
bias_hidden += alpha * delta_hidden

print("Updated Weights (Input to Hidden):")
print(weights_input_hidden)
print("Updated Weights (Hidden to Output):")
```

```
print(weights_hidden_output)

print("Updated Biases (Hidden Layer):")

print(bias_hidden)

print("Updated Biases (Output Layer):")

print(bias_output)
```