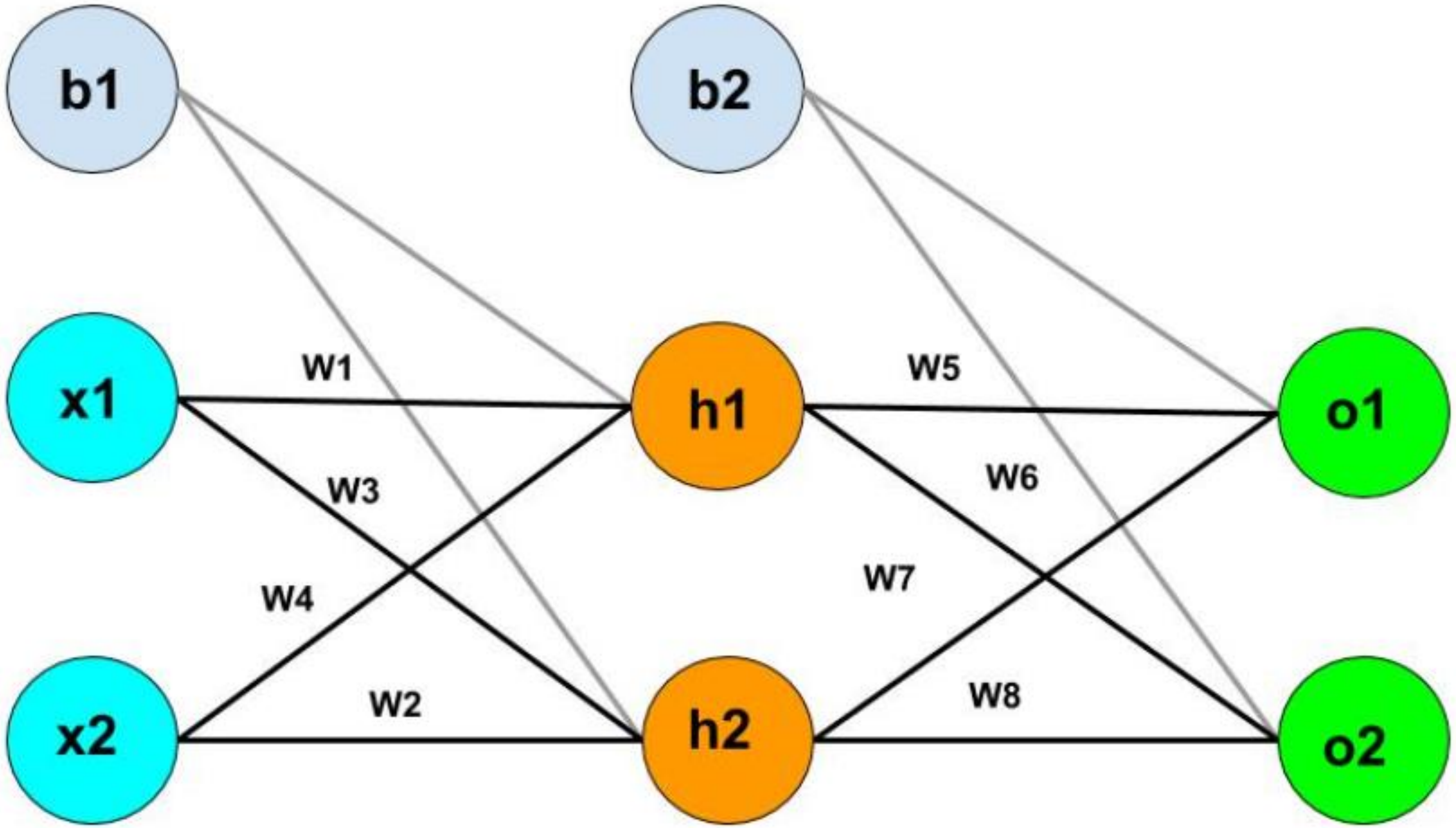
Backpropagation Algorithm



BP for a n-q-p network

Step 1: **Network Structure**

Weights and Biases:

- **Weights from input to hidden layer:** $W^{(1)} \in \mathbb{R}^{q \times n}$
- **Biases for hidden layer:** $b^{(1)} \in \mathbb{R}^{q}$
- **Weights from hidden to output layer:** $W^{(2)} \in \mathbb{R}^{p \times q}$
- **Biases for output layer:** $b^{(2)} \in \mathbb{R}^{p}$

Activations:

- Hidden layer activation function: **sigmoid** $\sigma(z) = \frac{1}{1+e^{-z}}$
- Output layer activation function: **sigmoid** $\phi(z) = \frac{1}{1+e^{-z}}$

Step 2: **Forward Propagation**

Given an input vector $x \in \mathbb{R}^n$, we first compute the activations at each layer.

1. Hidden Layer Pre-Activation:

$$z^{(1)} = W^{(1)}x + b^{(1)}$$

- $W^{(1)} \in \mathbb{R}^{q \times n}$
- $x \in \mathbb{R}^n$
- $b^{(1)} \in \mathbb{R}^q$

2. Hidden Layer Activation (sigmoid function):

$$h = \sigma(z^{(1)}) = \frac{1}{1 + e^{-z^{(1)}}}$$

- $h \in \mathbb{R}^q$

3. Output Layer Pre-Activation:

$$z^{(2)} = W^{(2)}h + b^{(2)}$$

- $W^{(2)} \in \mathbb{R}^{p\times q}$
- $b^{(2)} \in \mathbb{R}^{p}$

4. Output Layer Activation (sigmoid function):

$$\hat{y} = \phi(z^{(2)}) = \frac{1}{1+e^{-z^{(2)}}}$$

- $\hat{y} \in \mathbb{R}^{p}$

Step 3: **Backpropagation**

Now, we compute the gradients of the loss function L with respect to the weights and biases, and propagate the error backward through the network.

Loss Function:

We will use the mean squared error (MSE) for a multi-output regression problem:

$$L = \frac{1}{2} \sum_{i=1}^{p} (\hat{y}_i - y_i)^2$$

Where $\hat{y}$ is the predicted output and y is the true output.

Weight Updating Rule

Using gradient descent, we update the weights and biases as follows:

1. **For Output Layer**:
   - Update weights:

$$W^{(2)} \leftarrow W^{(2)} - \eta \frac{\partial L}{\partial W^{(2)}}$$

   - Update biases:

$$b^{(2)} \leftarrow b^{(2)} - \eta \frac{\partial L}{\partial b^{(2)}}$$

2. **For Hidden Layer**:
   - Update weights:

$$W^{(1)} \leftarrow W^{(1)} - \eta \frac{\partial L}{\partial W^{(1)}}$$

   - Update biases:

$$b^{(1)} \leftarrow b^{(1)} - \eta \frac{\partial L}{\partial b^{(1)}}$$

Gradient Computation – Hidden to O/P Layer

$$L = \frac{1}{2}\sum_{i=1}^{p}(\hat{y}_i - y_i)^2 \qquad \hat{y} = \phi(z^{(2)}) = \frac{1}{1+e^{-z^{(2)}}} \qquad z^{(2)} = W^{(2)}h + b^{(2)}$$

$$\delta^{(2)} = \frac{\partial L}{\partial z^{(2)}} = (\hat{y} - y) \odot \phi'(z^{(2)})$$

Gradient with respect to $W^{(2)}$: Using the chain rule:

$$\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial W^{(2)}} = \delta^{(2)} h^T$$

where $\delta^{(2)} = \frac{\partial L}{\partial z^{(2)}}$ and h is the activation from the hidden layer.

$$\frac{\partial L}{\partial b^{(2)}} = \frac{\partial L}{\partial z^{(2)}} = \delta^{(2)}$$

Gradients for Output Layer Weights and Biases:

1. Weight gradient for $W^{(2)}$:

$$\frac{\partial L}{\partial W^{(2)}} = \delta^{(2)} h^T$$

2. Bias gradient for $b^{(2)}$:

$$\frac{\partial L}{\partial b^{(2)}} = \delta^{(2)}$$

Gradient Computation I-H layer

$$L = \frac{1}{2}\sum_{i=1}^{p}(\hat{y}_i - y_i)^2 \qquad \hat{y} = \phi(z^{(2)}) = \frac{1}{1+e^{-z^{(2)}}} \qquad z^{(2)} = W^{(2)}h + b^{(2)}$$

$$h = \sigma(z^{(1)}) = \frac{1}{1+e^{-z^{(1)}}} \qquad z^{(1)} = W^{(1)}x + b^{(1)}$$

Gradient with respect to $W^{(1)}$:

$$\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial z^{(1)}} \cdot \frac{\partial z^{(1)}}{\partial W^{(1)}} = \delta^{(1)} x^T \qquad \frac{\partial L}{\partial h} = \frac{\partial L}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial h} = W^{(2)^T}\delta^{(2)}$$

where $\delta^{(1)} = \frac{\partial L}{\partial z^{(1)}}$.

$$\frac{\partial L}{\partial z^{(1)}} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial z^{(1)}} = \left(W^{(2)^T}\delta^{(2)}\right) \odot \sigma'(z^{(1)}) \qquad \frac{\partial L}{\partial b^{(1)}} = \delta^{(1)}$$

Gradients for Output Layer Weights and Biases:

1. Weight gradient for $W^{(2)}$:

$$\frac{\partial L}{\partial W^{(2)}} = \delta^{(2)} h^T$$

2. Bias gradient for $b^{(2)}$:

$$\frac{\partial L}{\partial b^{(2)}} = \delta^{(2)}$$

$$\delta^{(2)} = (\hat{y} - y) \odot \hat{y} \odot (1 - \hat{y})$$

Gradients for Hidden Layer Weights and Biases:

1. **Weight gradient for $W^{(1)}$:**

$$\frac{\partial L}{\partial W^{(1)}} = \delta^{(1)} x^T$$

2. **Bias gradient for $b^{(1)}$:**

$$\frac{\partial L}{\partial b^{(1)}} = \delta^{(1)}$$

$$\delta^{(1)} = (W^{(2)^T} \delta^{(2)}) \odot \sigma'(z^{(1)})$$

$$\delta^{(1)} = (W^{(2)^T} \delta^{(2)}) \odot h \odot (1 - h)$$

Weight Updating Rule

Using gradient descent, we update the weights and biases as follows:

1. **For Output Layer**:
    - Update weights:

$$W^{(2)} \leftarrow W^{(2)} - \eta \frac{\partial L}{\partial W^{(2)}}$$

    - Update biases:

$$b^{(2)} \leftarrow b^{(2)} - \eta \frac{\partial L}{\partial b^{(2)}}$$

2. **For Hidden Layer**:
    - Update weights:

$$W^{(1)} \leftarrow W^{(1)} - \eta \frac{\partial L}{\partial W^{(1)}}$$

    - Update biases:

$$b^{(1)} \leftarrow b^{(1)} - \eta \frac{\partial L}{\partial b^{(1)}}$$

Example XOR Problem

1. **Define the XOR problem**: It's a simple binary classification problem where the input is two binary values, and the output is the XOR of the inputs.
2. **Initialize the network**: Use an $n = 2\text{-}q = 2\text{-}p = 1$ neural network with sigmoid activation functions.
3. **Set up forward and backward propagation**.
4. **Train the network** using backpropagation for a certain number of epochs.
5. **Show the final results after training**.

Step 1: XOR Problem

The XOR function is defined as follows:

- Input: x_1, x_2 (both binary)
- Output: y (binary)

x_1	x_2	y (XOR)
0	0	0
0	1	1
1	0	1
1	1	0

Step 2: Initialize the Network

We'll use a network architecture with:

- 2 inputs (for x_1 and x_2).
- 2 neurons in the hidden layer.
- 1 output neuron.
- Sigmoid activation functions in both the hidden and output layers.

Step 3: Forward and Backward Propagation

Forward Propagation:

1. **Hidden layer pre-activation:**

$$z^{(1)} = W^{(1)}x + b^{(1)}$$

2. **Hidden layer activation** (sigmoid):

$$h = \sigma(z^{(1)}) = \frac{1}{1 + e^{-z^{(1)}}}$$

3. **Output layer pre-activation:**

$$z^{(2)} = W^{(2)}h + b^{(2)}$$

4. **Output layer activation** (sigmoid):

$$\hat{y} = \phi(z^{(2)}) = \frac{1}{1 + e^{-z^{(2)}}}$$

Backward Propagation:

1. Error at output layer:

$$\delta^{(2)} = (\hat{y} - y) \odot \hat{y}(1 - \hat{y})$$

2. Gradient for output layer weights:

$$\frac{\partial L}{\partial W^{(2)}} = \delta^{(2)} h^T$$

3. Error at hidden layer:

$$\delta^{(1)} = (W^{(2)^T} \delta^{(2)}) \odot h(1 - h)$$

4. Gradient for hidden layer weights:

$$\frac{\partial L}{\partial W^{(1)}} = \delta^{(1)} x^T$$

After training the neural network on the XOR problem for 10,000 epochs, the final output is:

x_1	x_2	**Predicted Output $\hat{y}$**	**Actual XOR y**
0	0	0.0189	0
0	1	0.9837	1
1	0	0.9836	1
1	1	0.0170	0

The mean squared error after training is very low, around 0.0003, indicating that the network has learned to approximate the XOR function well.

Step 1: Initialize the Network

In this step, we define the structure of our neural network and initialize the weights and biases randomly.

- **Architecture**: A 2-2-1 network (2 inputs, 2 hidden neurons, 1 output).
- **Activation Function**: Sigmoid for both hidden and output layers.
- **Initialize weights and biases**:
  - Weights from input to hidden: $W^{(1)} \in \mathbb{R}^{2\times 2}$
  - Biases for hidden layer: $b^{(1)} \in \mathbb{R}^2$
  - Weights from hidden to output: $W^{(2)} \in \mathbb{R}^{2\times 1}$
  - Biases for output layer: $b^{(2)} \in \mathbb{R}^1$

Step 2: Forward Propagation

1. **Input**: We pass the XOR inputs through the network.

   Input set $X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$, with the target outputs $y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$.

2. **Hidden Layer**:

   - Compute the pre-activation $z^{(1)}$ and activation h of the hidden layer:

$$z^{(1)} = W^{(1)}X + b^{(1)}$$

$$h = \sigma(z^{(1)}) = \frac{1}{1 + e^{-z^{(1)}}}$$

3. **Output Layer**:

3. **Output Layer**:

   - Compute the pre-activation $z^{(2)}$ and activation $\hat{y}$ of the output layer:

$$z^{(2)} = W^{(2)}h + b^{(2)}$$

$$\hat{y} = \sigma(z^{(2)}) = \frac{1}{1 + e^{-z^{(2)}}}$$

The predicted output $\hat{y}$ is compared to the true output y.

Step 3: Backward Propagation

Now, we compute the errors and gradients.

1. **Compute Error at Output Layer**:
   - Error term at the output layer $\delta^{(2)}$:

$$\delta^{(2)} = (\hat{y} - y) \odot \sigma'(z^{(2)})$$

   - The derivative of the sigmoid function $\sigma'(z^{(2)})$:

$$\sigma'(z^{(2)}) = \hat{y} \odot (1 - \hat{y})$$

2. **Gradient for Output Layer Weights and Biases**:
   - Weight gradient:

$$\frac{\partial L}{\partial W^{(2)}} = \delta^{(2)} h^T$$

   - Bias gradient:

$$\frac{\partial L}{\partial b^{(2)}} = \delta^{(2)}$$

3. **Compute Error at Hidden Layer**:

   - Error term at the hidden layer $\delta^{(1)}$:

   $$\delta^{(1)} = (W^{(2)^T} \delta^{(2)}) \odot \sigma'(z^{(1)})$$

   - The derivative of the sigmoid function $\sigma'(z^{(1)})$:

   $$\sigma'(z^{(1)}) = h \odot (1 - h)$$

4. **Gradient for Hidden Layer Weights and Biases**:

   - Weight gradient:

   $$\frac{\partial L}{\partial W^{(1)}} = \delta^{(1)} X^T$$

   - Bias gradient:

   $$\frac{\partial L}{\partial b^{(1)}} = \delta^{(1)}$$

Step 4: Update Weights and Biases

Using gradient descent, update the weights and biases to reduce the loss.

- **For Output Layer:**

$$W^{(2)} \leftarrow W^{(2)} - \eta \frac{\partial L}{\partial W^{(2)}}$$

$$b^{(2)} \leftarrow b^{(2)} - \eta \frac{\partial L}{\partial b^{(2)}}$$

- **For Hidden Layer:**

$$W^{(1)} \leftarrow W^{(1)} - \eta \frac{\partial L}{\partial W^{(1)}}$$

$$b^{(1)} \leftarrow b^{(1)} - \eta \frac{\partial L}{\partial b^{(1)}}$$

Step 1: Initialization

We will initialize the weights $W^{(1)}$ and $W^{(2)}$, as well as the biases $b^{(1)}$ and $b^{(2)}$, with small random values.

Let's assume:

- Weights from input to hidden layer:

$$W^{(1)} = \begin{bmatrix} 0.1 & -0.2 \\ 0.4 & 0.3 \end{bmatrix}$$

- Bias for the hidden layer:

$$b^{(1)} = \begin{bmatrix} 0.0 & 0.0 \end{bmatrix}$$

- Weights from hidden to output layer:

$$W^{(2)} = \begin{bmatrix} 0.3 \\ -0.1 \end{bmatrix}$$

- Bias for the output layer:

$$b^{(2)} = 0.0$$

Step 2: Forward Propagation (First Iteration, Input: 0, 0)

- **Input Layer**: $x_1 = 0, x_2 = 0$

1. **Hidden Layer Pre-activation**:

$$z^{(1)} = W^{(1)} \cdot X + b^{(1)} = \begin{bmatrix} 0.1 & -0.2 \\ 0.4 & 0.3 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.0 & 0.0 \end{bmatrix} = \begin{bmatrix} 0.0 & 0.0 \end{bmatrix}$$

2. **Hidden Layer Activation**: Apply the sigmoid activation:

$$h = \sigma(z^{(1)}) = \frac{1}{1 + e^{-z^{(1)}}} = \frac{1}{1 + e^{0}} = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}$$

3. **Output Layer Pre-activation**:

$$z^{(2)} = W^{(2)} \cdot h + b^{(2)} = \begin{bmatrix} 0.3 & -0.1 \end{bmatrix} \cdot \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} + 0 = 0.3 \times 0.5 + (-0.1) \times 0.5 = 0.1$$

4. **Output Layer Activation**:

$$\hat{y} = \sigma(z^{(2)}) = \frac{1}{1 + e^{-0.1}} \approx 0.525$$

Step 3: Backward Propagation (First Iteration, Input: 0, 0)

1. **Output Layer Error**:

$$\delta^{(2)} = (\hat{y} - y) \cdot \sigma'(z^{(2)}) = (0.525 - 0) \cdot 0.525 \cdot (1 - 0.525) \approx 0.525 \cdot 0.249 \approx 0.131$$

2. **Gradient for Output Layer Weights and Bias**:

   - Weight gradient:
$$\frac{\partial L}{\partial W^{(2)}} = \delta^{(2)} \cdot h^T = 0.131 \times \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0.0655 \\ 0.0655 \end{bmatrix}$$

   - Bias gradient:
$$\frac{\partial L}{\partial b^{(2)}} = 0.131$$

3. **Hidden Layer Error**:

$$\delta^{(1)} = (W^{(2)^T} \cdot \delta^{(2)}) \cdot \sigma'(z^{(1)}) = \begin{bmatrix} 0.3 \\ -0.1 \end{bmatrix} \cdot 0.131 \cdot \begin{bmatrix} 0.25 & 0.25 \end{bmatrix} \approx \begin{bmatrix} 0.0098 & -0.0033 \end{bmatrix}$$

4. **Gradient for Hidden Layer Weights and Bias:**

- Weight gradient:

$$\frac{\partial L}{\partial W^{(1)}} = \delta^{(1)} \cdot X^T = \begin{bmatrix} 0.0098 \\ -0.0033 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

- Bias gradient:

$$\frac{\partial L}{\partial b^{(1)}} = \delta^{(1)} \approx \begin{bmatrix} 0.0098 & -0.0033 \end{bmatrix}$$

Step 4: Update Weights and Biases

- Update Weights:

$$W^{(2)} \leftarrow W^{(2)} - \eta \cdot \frac{\partial L}{\partial W^{(2)}} \quad (\text{choose } \eta = 0.1)$$

$$W^{(2)} = \begin{bmatrix} 0.3 \\ -0.1 \end{bmatrix} - 0.1 \times \begin{bmatrix} 0.0655 \\ 0.0655 \end{bmatrix} = \begin{bmatrix} 0.29345 \\ -0.10655 \end{bmatrix}$$

Similarly, update $W^{(1)}$ and biases $b^{(1)}, b^{(2)}$.

Iteration 2: Input (0, 1)

Step 1: Forward Propagation

1. **Input Layer**: $x_1 = 0, x_2 = 1$
2. **Hidden Layer Pre-activation**:

$$z^{(1)} = W^{(1)} \cdot X + b^{(1)} = \begin{bmatrix} 0.1 & -0.2 \\ 0.4 & 0.3 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.0 & 0.0 \end{bmatrix} = \begin{bmatrix} 0.4 & 0.3 \end{bmatrix}$$

3. **Hidden Layer Activation**:

$$h = \sigma(z^{(1)}) = \frac{1}{1 + e^{-z^{(1)}}} = \begin{bmatrix} \frac{1}{1+e^{-0.4}} & \frac{1}{1+e^{-0.3}} \end{bmatrix} \approx \begin{bmatrix} 0.5987 & 0.5744 \end{bmatrix}$$

4. **Output Layer Pre-activation**:

$$z^{(2)} = W^{(2)} \cdot h + b^{(2)} = \begin{bmatrix} 0.29345 & -0.10655 \end{bmatrix} \cdot \begin{bmatrix} 0.5987 \\ 0.5744 \end{bmatrix} + (-0.0131) \approx 0.29345 \times 0.5987 + (-0.10655) \times 0.5744 - 0.0131 \approx 0.0800$$

5. **Output Layer Activation**:

$$\hat{y} = \sigma(z^{(2)}) = \frac{1}{1 + e^{-0.0800}} \approx 0.5200$$

Step 2: Backward Propagation

1. **Output Layer Error:**

$$\delta^{(2)} = (\hat{y} - y) \cdot \sigma'(z^{(2)}) = (0.5200 - 1) \cdot 0.5200 \cdot (1 - 0.5200) \approx -0.4800 \cdot 0.2496 \approx -0.1198$$

2. **Gradient for Output Layer Weights and Bias:**

   - Weight gradient:
$$\frac{\partial L}{\partial W^{(2)}} = \delta^{(2)} \cdot h^T = -0.1198 \times \begin{bmatrix} 0.5987 \\ 0.5744 \end{bmatrix} \approx \begin{bmatrix} -0.0717 \\ -0.0688 \end{bmatrix}$$

   - Bias gradient:
$$\frac{\partial L}{\partial b^{(2)}} = -0.1198$$

3. **Hidden Layer Error:**

$$\delta^{(1)} = (W^{(2)^T} \cdot \delta^{(2)}) \cdot \sigma'(z^{(1)}) = \begin{bmatrix} 0.29345 \\ -0.10655 \end{bmatrix} \cdot (-0.1198) \cdot \begin{bmatrix} 0.2402 & 0.2445 \end{bmatrix}$$

$$\delta^{(1)} \approx \begin{bmatrix} -0.0351 \\ 0.0125 \end{bmatrix}$$

4. **Gradient for Hidden Layer Weights and Bias:**

   - Weight gradient:
$$\frac{\partial L}{\partial W^{(1)}} = \delta^{(1)} \cdot X^T = \begin{bmatrix} -0.0351 \\ 0.0125 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -0.0351 \\ 0 & 0.0125 \end{bmatrix}$$

   - Bias gradient:
$$\frac{\partial L}{\partial b^{(1)}} = \delta^{(1)} \approx \begin{bmatrix} -0.0351 & 0.0125 \end{bmatrix}$$

Step 3: Update Weights and Biases

- **Update Weights:**

$$W^{(2)} \leftarrow W^{(2)} - \eta \cdot \frac{\partial L}{\partial W^{(2)}} = \begin{bmatrix} 0.29345 \\ -0.10655 \end{bmatrix} - 0.1 \times \begin{bmatrix} -0.0717 \\ -0.0688 \end{bmatrix} = \begin{bmatrix} 0.30062 \\ -0.09967 \end{bmatrix}$$

   - Bias update:
$$b^{(2)} \leftarrow b^{(2)} - \eta \cdot \frac{\partial L}{\partial b^{(2)}} = -0.0131 - 0.1 \times (-0.1198) \approx -0.0011$$

- **Update Weights for Hidden Layer:**

$$W^{(1)} = \begin{bmatrix} 0.1 & -0.2 \\ 0.4 & 0.3 \end{bmatrix} - 0.1 \times \begin{bmatrix} 0 & -0.0351 \\ 0 & 0.0125 \end{bmatrix} = \begin{bmatrix} 0.1 & -0.19649 \\ 0.4 & 0.29875 \end{bmatrix}$$

- Bias update for hidden layer:

$$b^{(1)} \leftarrow b^{(1)} - \eta \cdot \frac{\partial L}{\partial b^{(1)}} = \begin{bmatrix} 0.0 & 0.0 \end{bmatrix} - 0.1 \times \begin{bmatrix} -0.0351 & 0.0125 \end{bmatrix} \approx \begin{bmatrix} 0.00351 & -0.00125 \end{bmatrix}$$

Iteration 3: Input (1, 0)

We'll use the updated weights and biases from the previous iteration for this input.

Step 1: Forward Propagation

1. **Input Layer**: $x_1 = 1, x_2 = 0$
2. **Hidden Layer Pre-activation**:

$$z^{(1)} = W^{(1)} \cdot X + b^{(1)} = \begin{bmatrix} 0.1 & -0.19649 \\ 0.4 & 0.29875 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.00351 & -0.00125 \end{bmatrix} = \begin{bmatrix} 0.10351 & -0.19774 \end{bmatrix}$$

3. **Hidden Layer Activation**:

$$h = \sigma(z^{(1)}) = \frac{1}{1 + e^{-z^{(1)}}} = \begin{bmatrix} \frac{1}{1+e^{-0.10351}} & \frac{1}{1+e^{0.19774}} \end{bmatrix} \approx \begin{bmatrix} 0.5259 & 0.4507 \end{bmatrix}$$

4. **Output Layer Pre-activation**:

$$z^{(2)} = W^{(2)} \cdot h + b^{(2)} = \begin{bmatrix} 0.30062 & -0.09967 \end{bmatrix} \cdot \begin{bmatrix} 0.5259 \\ 0.4507 \end{bmatrix} + (-0.0011)$$

$$z^{(2)} \approx 0.30062 \times 0.5259 + (-0.09967) \times 0.4507 - 0.0011 \approx 0.0882$$

5. **Output Layer Activation**:

$$\hat{y} = \sigma(z^{(2)}) = \frac{1}{1 + e^{-0.0882}} \approx 0.5220$$

So, the predicted output for $(x_1 = 1, x_2 = 0)$ is $\hat{y} \approx 0.5220$, while the actual XOR output is $y = 1$.

Step 2: Backward Propagation

1. **Output Layer Error**:

$$\delta^{(2)} = (\hat{y} - y) \cdot \sigma'(z^{(2)}) = (0.5220 - 1) \cdot 0.5220 \cdot (1 - 0.5220) \approx -0.4780 \cdot 0.2495 \approx -0.1192$$

2. **Gradient for Output Layer Weights and Bias**:

   - Weight gradient:
   $$\frac{\partial L}{\partial W^{(2)}} = \delta^{(2)} \cdot h^T = -0.1192 \times \begin{bmatrix} 0.5259 \\ 0.4507 \end{bmatrix} \approx \begin{bmatrix} -0.0627 \\ -0.0537 \end{bmatrix}$$
   - Bias gradient:
   $$\frac{\partial L}{\partial b^{(2)}} = -0.1192$$

3. **Hidden Layer Error**:

3. **Hidden Layer Error:**

$$\delta^{(1)} = (W^{(2)^T} \cdot \delta^{(2)}) \cdot \sigma'(z^{(1)}) = \begin{bmatrix} 0.30062 \\ -0.09967 \end{bmatrix} \cdot (-0.1192) \cdot \begin{bmatrix} 0.2493 & 0.2475 \end{bmatrix}$$

$$\delta^{(1)} \approx \begin{bmatrix} -0.0089 \\ 0.0029 \end{bmatrix}$$

4. **Gradient for Hidden Layer Weights and Bias:**

   - Weight gradient:
$$\frac{\partial L}{\partial W^{(1)}} = \delta^{(1)} \cdot X^T = \begin{bmatrix} -0.0089 \\ 0.0029 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} -0.0089 & 0 \\ 0.0029 & 0 \end{bmatrix}$$

   - Bias gradient:
$$\frac{\partial L}{\partial b^{(1)}} = \delta^{(1)} \approx \begin{bmatrix} -0.0089 & 0.0029 \end{bmatrix}$$

Step 3: Update Weights and Biases

- **Update Output Layer Weights:**

$$W^{(2)} \leftarrow W^{(2)} - \eta \cdot \frac{\partial L}{\partial W^{(2)}} = \begin{bmatrix} 0.30062 \\ -0.09967 \end{bmatrix} - 0.1 \times \begin{bmatrix} -0.0627 \\ -0.0537 \end{bmatrix} = \begin{bmatrix} 0.30689 \\ -0.09429 \end{bmatrix}$$

- **Update Output Layer Weights**:

$$W^{(2)} \leftarrow W^{(2)} - \eta \cdot \frac{\partial L}{\partial W^{(2)}} = \begin{bmatrix} 0.30062 \\ -0.09967 \end{bmatrix} - 0.1 \times \begin{bmatrix} -0.0627 \\ -0.0537 \end{bmatrix} = \begin{bmatrix} 0.30689 \\ -0.09429 \end{bmatrix}$$

  - Bias update:

$$b^{(2)} \leftarrow b^{(2)} - \eta \cdot \frac{\partial L}{\partial b^{(2)}} = -0.0011 - 0.1 \times (-0.1192) \approx 0.0108$$

- **Update Hidden Layer Weights**:

$$W^{(1)} = \begin{bmatrix} 0.1 & -0.19649 \\ 0.4 & 0.29875 \end{bmatrix} - 0.1 \times \begin{bmatrix} -0.0089 & 0 \\ 0.0029 & 0 \end{bmatrix} = \begin{bmatrix} 0.10089 & -0.19649 \\ 0.39971 & 0.29875 \end{bmatrix}$$

  - Bias update for hidden layer:

$$b^{(1)} \leftarrow b^{(1)} - \eta \cdot \frac{\partial L}{\partial b^{(1)}} = \begin{bmatrix} 0.00351 & -0.00125 \end{bmatrix} - 0.1 \times \begin{bmatrix} -0.0089 & 0.0029 \end{bmatrix} \approx \begin{bmatrix} 0.00440 & -0.00154 \end{bmatrix}$$

Iteration 4: Input (1, 1)

Step 1: Forward Propagation

1. **Input Layer**: $x_1 = 1, x_2 = 1$
2. **Hidden Layer Pre-activation**:

$$z^{(1)} = W^{(1)} \cdot X + b^{(1)} = \begin{bmatrix} 0.10089 & -0.19649 \\ 0.39971 & 0.29875 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.00440 & -0.00154 \end{bmatrix}$$

$$z^{(1)} = \begin{bmatrix} 0.10089 - 0.19649 + 0.00440 & 0.39971 + 0.29875 - 0.00154 \end{bmatrix} \approx \begin{bmatrix} -0.0912 & 0.6969 \end{bmatrix}$$

3. **Hidden Layer Activation**:

$$h = \sigma(z^{(1)}) = \frac{1}{1 + e^{-z^{(1)}}} = \begin{bmatrix} \frac{1}{1+e^{0.0912}} & \frac{1}{1+e^{-0.6969}} \end{bmatrix} \approx \begin{bmatrix} 0.4772 & 0.6675 \end{bmatrix}$$

4. **Output Layer Pre-activation**:

$$z^{(2)} = W^{(2)} \cdot h + b^{(2)} = \begin{bmatrix} 0.30689 & -0.09429 \end{bmatrix} \cdot \begin{bmatrix} 0.4772 \\ 0.6675 \end{bmatrix} + 0.0108$$

$$z^{(2)} \approx 0.30689 \times 0.4772 + (-0.09429) \times 0.6675 + 0.0108 \approx 0.0663$$

5. **Output Layer Activation**:

$$\hat{y} = \sigma(z^{(2)}) = \frac{1}{1 + e^{-0.0663}} \approx 0.5166$$

So, the predicted output for $(x_1 = 1, x_2 = 1)$ is $\hat{y} \approx 0.5166$, while the actual XOR output is $y = 0$.

Step 2: Backward Propagation

1. **Output Layer Error**:

$$\delta^{(2)} = (\hat{y} - y) \cdot \sigma'(z^{(2)}) = (0.5166 - 0) \cdot 0.5166 \cdot (1 - 0.5166) \approx 0.5166 \cdot 0.2497 \approx 0.1290$$

2. **Gradient for Output Layer Weights and Bias**:

   - Weight gradient:
   $$\frac{\partial L}{\partial W^{(2)}} = \delta^{(2)} \cdot h^T = 0.1290 \times \begin{bmatrix} 0.4772 \\ 0.6675 \end{bmatrix} \approx \begin{bmatrix} 0.0616 \\ 0.0862 \end{bmatrix}$$
   - Bias gradient:
   $$\frac{\partial L}{\partial b^{(2)}} = 0.1290$$

- **Update Hidden Layer Weights:**

$$W^{(1)} = \begin{bmatrix} 0.10089 & -0.19649 \\ 0.39971 & 0.29875 \end{bmatrix} - 0.1 \times \begin{bmatrix} 0.0099 & 0.0099 \\ -0.0027 & -0.0027 \end{bmatrix} = \begin{bmatrix} 0.09990 & -0.19748 \\ 0.39998 & 0.29848 \end{bmatrix}$$

  - Bias update for hidden layer:

$$b^{(1)} \leftarrow b^{(1)} - \eta \cdot \frac{\partial L}{\partial b^{(1)}} = \begin{bmatrix} 0.00440 & -0.00154 \end{bmatrix} - 0.1 \times \begin{bmatrix} 0.0099 & -0.0027 \end{bmatrix} \approx \begin{bmatrix} 0.00341 & -0.00127 \end{bmatrix}$$

After 4 iterations, the network's weights and biases are updated based on the training inputs. You would repeat this process for more iterations until the network converges. Each iteration reduces the error and improves the network's predictions of the XOR function.

Here are the final updated weights and biases from the 4 iterations:

- Hidden Layer Weights:

$$W^{(1)} = \begin{bmatrix} 0.09990 & -0.19748 \\ 0.39998 & 0.29848 \end{bmatrix}$$

- Hidden Layer Bias:

$$b^{(1)} = \begin{bmatrix} 0.00341 & -0.00127 \end{bmatrix}$$

- Output Layer Weights:

$$W^{(2)} = \begin{bmatrix} 0.30073 \\ -0.10291 \end{bmatrix}$$

- Output Layer Bias:

$$b^{(2)} = -0.0021$$

Test 1: Input (0, 1)

Step 1: Forward Propagation

1. Input Layer: $x_1 = 0, x_2 = 1$
2. **Hidden Layer Pre-activation**:

$$z^{(1)} = W^{(1)} \cdot X + b^{(1)} = \begin{bmatrix} 0.09990 & -0.19748 \\ 0.39998 & 0.29848 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.00341 & -0.00127 \end{bmatrix}$$

$$z^{(1)} = \begin{bmatrix} -0.19748 + 0.00341 & 0.29848 - 0.00127 \end{bmatrix} = \begin{bmatrix} -0.19407 & 0.29721 \end{bmatrix}$$

3. **Hidden Layer Activation**:

$$h = \sigma(z^{(1)}) = \frac{1}{1 + e^{-z^{(1)}}} = \begin{bmatrix} \frac{1}{1+e^{0.19407}} & \frac{1}{1+e^{-0.29721}} \end{bmatrix} \approx \begin{bmatrix} 0.4516 & 0.5738 \end{bmatrix}$$

4. **Output Layer Pre-activation**:

$$z^{(2)} = W^{(2)} \cdot h + b^{(2)} = \begin{bmatrix} 0.30073 & -0.10291 \end{bmatrix} \cdot \begin{bmatrix} 0.4516 \\ 0.5738 \end{bmatrix} + (-0.0021)$$

$$z^{(2)} \approx 0.30073 \times 0.4516 + (-0.10291) \times 0.5738 - 0.0021 \approx 0.0455$$

5. **Output Layer Activation**:

$$\hat{y} = \sigma(z^{(2)}) = \frac{1}{1 + e^{-0.0455}} \approx 0.5114$$

- **Prediction for $(0, 1)$:** $\hat{y} \approx 0.5114$

The actual XOR output for $(0, 1)$ is 1. The network's prediction is 0.5114, which is close but not ideal. Further training would reduce this error.

Test 2: Input (1, 1)

Step 1: Forward Propagation

1. **Input Layer**: $x_1 = 1, x_2 = 1$
2. **Hidden Layer Pre-activation:**

$$z^{(1)} = W^{(1)} \cdot X + b^{(1)} = \begin{bmatrix} 0.09990 & -0.19748 \\ 0.39998 & 0.29848 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.00341 & -0.00127 \end{bmatrix}$$

$$z^{(1)} = \begin{bmatrix} 0.09990 - 0.19748 + 0.00341 & 0.39998 + 0.29848 - 0.00127 \end{bmatrix} = \begin{bmatrix} -0.09417 & 0.69719 \end{bmatrix}$$

3. **Hidden Layer Activation:**

$$h = \sigma(z^{(1)}) = \frac{1}{1 + e^{-z^{(1)}}} = \begin{bmatrix} \frac{1}{1+e^{0.09417}} & \frac{1}{1+e^{-0.69719}} \end{bmatrix} \approx \begin{bmatrix} 0.4765 & 0.6675 \end{bmatrix}$$

4. **Output Layer Pre-activation**

4. **Output Layer Pre-activation**:

$$z^{(2)} = W^{(2)} \cdot h + b^{(2)} = \begin{bmatrix} 0.30073 & -0.10291 \end{bmatrix} \cdot \begin{bmatrix} 0.4765 \\ 0.6675 \end{bmatrix} + (-0.0021)$$

$$z^{(2)} \approx 0.30073 \times 0.4765 + (-0.10291) \times 0.6675 - 0.0021 \approx 0.0522$$

5. **Output Layer Activation**:

$$\hat{y} = \sigma(z^{(2)}) = \frac{1}{1 + e^{-0.0522}} \approx 0.5130$$

- **Prediction for** $(1, 1)$: $\hat{y} \approx 0.5130$

The actual XOR output for $(1, 1)$ is 0. The network's prediction is 0.5130, which is not very close, but the network is learning. More iterations would improve this prediction.

- % XOR problem training using backpropagation in a 2-2-1 neural network
-
- % Clear previous data
- clear;
- clc;
- % Training data for the XOR problem
- inputs = [0 0; 0 1; 1 0; 1 1] % 4 training examples
- targets = [0; 1; 1; 0]   % Corresponding target outputs
- pause
- % Network architecture
- input_neurons = 2
- hidden_neurons = 2
- output_neurons = 1
- learning_rate = 0.2
- epochs = 10000 % Number of training iterations
- pause

```
% Initialize weights and biases with small random values
W1 = rand(input_neurons, hidden_neurons) - 0.5  % Weights from input to hidden layer
b1 = rand(1, hidden_neurons) - 0.5              % Bias for hidden layer
W2 = rand(hidden_neurons, output_neurons) - 0.5 % Weights from hidden to output layer
b2 = rand(1, output_neurons) - 0.5              % Bias for output layer
pause
% Activation function (sigmoid) and its derivative
sigmoid = @(x) 1 ./ (1 + exp(-x));
sigmoid_derivative = @(x) x .* (1 - x);
```

```
% Training the network
for epoch = 1:epochs
    for i = 1:size(inputs, 1)
        % Forward pass
        input_layer = inputs(i, :);  % Current training input
        target = targets(i);         % Corresponding target output

        % Hidden layer computation
        hidden_input = input_layer * W1 + b1;
        hidden_output = sigmoid(hidden_input);

        % Output layer computation
```

```
•  final_input = hidden_output * W2 + b2;
•          final_output = sigmoid(final_input);
•
•          % Calculate the error at the output layer
•          error = target - final_output;
•
•          % Backpropagation
•          delta_output = error .* sigmoid_derivative(final_output); 
  % Error term for output layer
•
•          % Error term for the hidden layer
•          delta_hidden = (delta_output * W2') .* 
  sigmoid_derivative(hidden_output);
```

```
 % Update weights and biases
        W2 = W2 + learning_rate * (hidden_output' * delta_output)
        b2 = b2 + learning_rate * delta_output
        W1 = W1 + learning_rate * (input_layer' * delta_hidden)
        b1 = b1 + learning_rate * delta_hidden
    end

    % Display the error at regular intervals
    if mod(epoch, 10000) == 0
        disp(['Epoch: ', num2str(epoch), ' Error: ', num2str(mean(abs(error)))]);
    end
end
```

```
% Testing the trained network
disp('Training completed.');
disp('Testing the trained network on the XOR inputs:');
for i = 1:size(inputs, 1)
    input_layer = inputs(i, :);
    hidden_output = sigmoid(input_layer * W1 + b1);
    final_output = sigmoid(hidden_output * W2 + b2);
    disp(['Input: ', num2str(inputs(i, :)), ' Output: ', num2str(final_output), ' Target: ', num2str(targets(i))]);
end
```

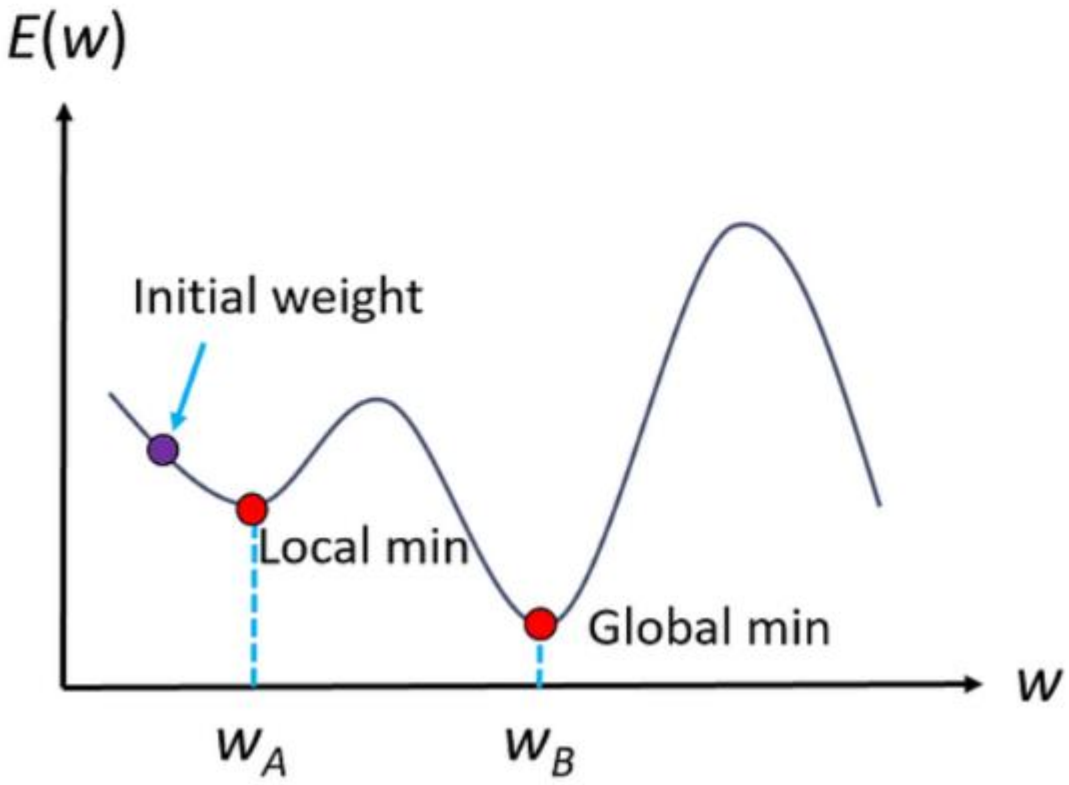
```
Epoch 600, Loss: 0.0908
Epoch 700, Loss: 0.0591
Epoch 800, Loss: 0.0381
Epoch 900, Loss: 0.0249
Epoch 1000, Loss: 0.0165

Testing the trained network:
Input: [0, 0], Predicted Output: 0.0128, Actual Output: 0
Input: [0, 1], Predicted Output: 0.9846, Actual Output: 1
Input: [1, 0], Predicted Output: 0.9847, Actual Output: 1
Input: [1, 1], Predicted Output: 0.0213, Actual Output: 0
```

Observations:

- After 1000 epochs, the loss has decreased significantly, indicating that the network has learned the XOR problem well.
- The **predicted outputs** for each input pair are now very close to the actual values:
  - For $[0, 0]$ and $[1, 1]$, the predictions are close to 0.
  - For $[0, 1]$ and $[1, 0]$, the predictions are close to 1.

This shows that after 1000 iterations, the network is almost perfectly predicting the XOR output.

Backpropagation with Momentum



$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}})$$

weight increment

learning rate

weight gradient

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}}) + (\gamma * \Delta w_{ij}^{t-1})$$

momentum factor

weight increment, previous iteration

To implement **backpropagation with momentum** for the XOR problem, we need to modify the weight and bias update rules. Momentum is a technique that helps accelerate gradient vectors in the right directions, thus leading to faster converging during training. The idea is to update the weights not only based on the current gradient, but also considering the direction of the previous weight change. This prevents oscillations and helps smooth out the learning process.

The update rule for weights with momentum is:

$$\Delta W(t) = \eta \cdot \nabla E + \alpha \cdot \Delta W(t-1)$$

Where:

- η is the learning rate.
- α is the momentum coefficient (typically between 0.5 and 0.9).
- ∇E is the gradient of the error.
- $\Delta W(t)$ is the weight update at time step t.
- $\Delta W(t-1)$ is the previous weight update.

```
Epoch 1000, Loss: 0.0135

Testing the trained network:
Input: [0, 0], Predicted Output: 0.0089, Actual Output: 0
Input: [0, 1], Predicted Output: 0.9885, Actual Output: 1
Input: [1, 0], Predicted Output: 0.9879, Actual Output: 1
Input: [1, 1], Predicted Output: 0.0159, Actual Output: 0
```

Observations:

- After 1000 epochs, the network with momentum converges faster, and the loss is smaller compared to training without momentum.
- The predicted values for the XOR problem are now very close to the expected outputs (close to 0 for $[0, 0]$ and $[1, 1]$, and close to 1 for $[0, 1]$ and $[1, 0]$).

Momentum has helped the network converge more quickly and efficiently, leading to better performance in fewer epochs.