6CS012 - Artificial Intelligence and Machine Learning. Hint for Understanding and Computing Gradients.

Prepared By: Siman Giri {Module Leader - 6CS012}

March 16, 2025

Tutorial - 4.

1 Instructions

This sheet contains the Hint for solving the problem mentioned in Tutorial Slide.

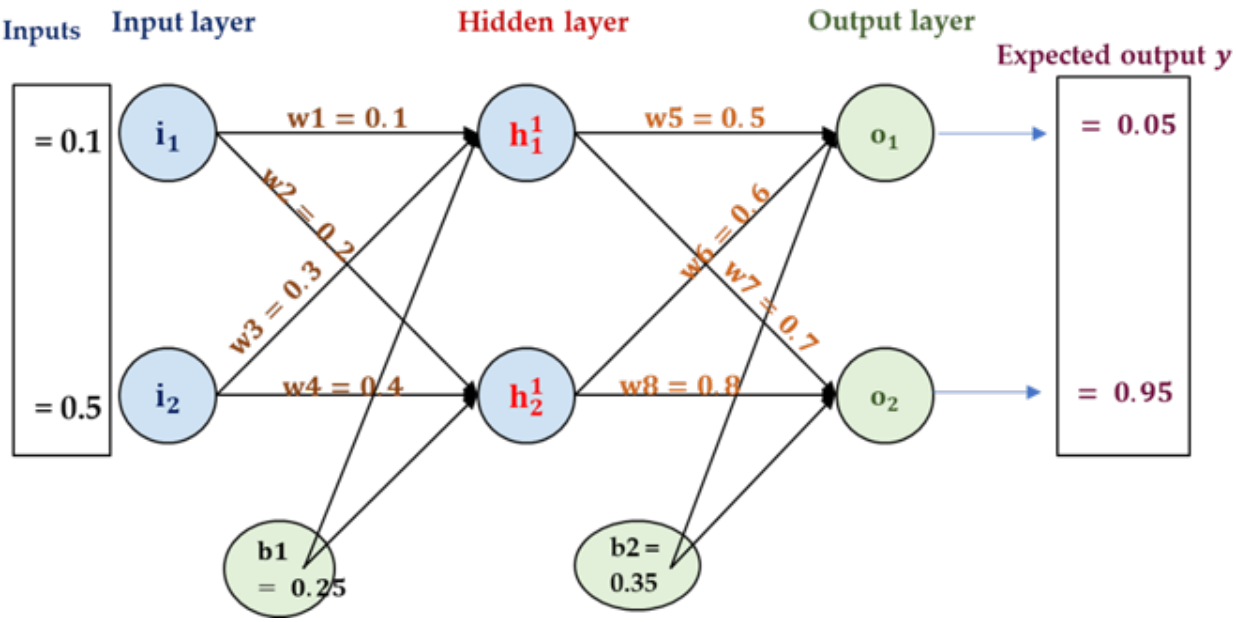

Figure 1: Network we are trying to Solve.

1. Computing Gradient against $w6$:

Hint: Computing Partial Derivative of Error Function

To compute the partial derivative of the error function E_{total} with respect to weight w_6, we use the chain rule:

$$\frac{\partial E_{\text{total}}}{\partial w_6} = \frac{\partial E_{\text{total}}}{\partial y_1} \cdot \frac{\partial y_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_6}$$

This formula breaks down as follows:

Step 1: Compute $\frac{\partial E_{\text{total}}}{\partial y_1}$

The total error function is the sum of squared errors for all outputs. For simplicity, we focus on a single output error:

$$E_{\text{total}} = \frac{1}{2} \sum_i (y_i - y_i^{\text{target}})^2$$

For the first output neuron, the partial derivative is:

$$\frac{\partial E_{\text{total}}}{\partial y_1} = (y_1 - y_1^{\text{target}})$$

Step 2: Compute $\frac{\partial y_1}{\partial z_3}$

The output y_1 is the sigmoid function of the weighted sum z_3:

$$y_1 = \sigma(z_3) = \frac{1}{1 + e^{-z_3}}$$

The derivative of the sigmoid function is:

$$\frac{\partial y_1}{\partial z_3} = y_1(1 - y_1)$$

Step 3: Compute $\frac{\partial z_3}{\partial w_6}$

The weighted sum z_3 for the output neuron is:

$$z_3 = w_5 \cdot o_1 + w_6 \cdot o_2 + b_2$$

The partial derivative of z_3 with respect to w_6 is:

$$\frac{\partial z_3}{\partial w_6} = o_2$$

`Here:` o_2 `is output at neuron 2 of output layer.`

Step 4: Combine Everything

Now, we can combine all the pieces using the chain rule:

$$\frac{\partial E_{\text{total}}}{\partial w_6} = (y_1 - y_1^{\text{target}}) \cdot y_1(1 - y_1) \cdot o_2$$

This is the gradient of the error with respect to weight w_6.

2. Computing Gradient against $w7$:

Hint: Computing Partial Derivative of Error Function

To compute the partial derivative of the error function E_{total} with respect to weight w_7, we use the chain rule:

$$\frac{\partial E_{\text{total}}}{\partial w_7} = \frac{\partial E_{\text{total}}}{\partial y_2} \cdot \frac{\partial y_2}{\partial z_4} \cdot \frac{\partial z_4}{\partial w_7}$$

This formula breaks down as follows:

Step 1: Compute $\frac{\partial E_{\text{total}}}{\partial y_2}$

The total error function is the sum of squared errors for all outputs. For simplicity, we focus on the second output error:

$$E_{\text{total}} = \frac{1}{2} \sum_i (y_i - y_i^{\text{target}})^2$$

For the second output neuron, the partial derivative is:

$$\frac{\partial E_{\text{total}}}{\partial y_2} = (y_2 - y_2^{\text{target}})$$

Step 2: Compute $\frac{\partial y_2}{\partial z_4}$

The output y_2 is the sigmoid function of the weighted sum z_4:

$$y_2 = \sigma(z_4) = \frac{1}{1 + e^{-z_4}}$$

The derivative of the sigmoid function is:

$$\frac{\partial y_2}{\partial z_4} = y_2(1 - y_2)$$

Step 3: Compute $\frac{\partial z_4}{\partial w_7}$

The weighted sum z_4 for the second output neuron is:

$$z_4 = w_7 \cdot o_1 + w_8 \cdot o_2 + b_2$$

The partial derivative of z_4 with respect to w_7 is:

$$\frac{\partial z_4}{\partial w_7} = o_1$$

Step 4: Combine Everything

Now, we can combine all the pieces using the chain rule:

$$\frac{\partial E_{\text{total}}}{\partial w_7} = (y_2 - y_2^{\text{target}}) \cdot y_2(1 - y_2) \cdot o_1$$

`Here:` o_1 `is output at neuron 1 of output layer.`

This is the gradient of the error with respect to weight w_7.

3. Computing Gradient against $w8$:

Hint: Computing Partial Derivative of Error Function

To compute the partial derivative of the error function E_{total} with respect to weight w_8, we use the chain rule:

$$\frac{\partial E_{\text{total}}}{\partial w_8} = \frac{\partial E_{\text{total}}}{\partial y_2} \cdot \frac{\partial y_2}{\partial z_4} \cdot \frac{\partial z_4}{\partial w_8}$$

This formula breaks down as follows:

Step 1: Compute $\frac{\partial E_{\text{total}}}{\partial y_2}$

The total error function is the sum of squared errors for all outputs. For simplicity, we focus on the second output error:

$$E_{\text{total}} = \frac{1}{2} \sum_i (y_i - y_i^{\text{target}})^2$$

For the second output neuron, the partial derivative is:

$$\frac{\partial E_{\text{total}}}{\partial y_2} = (y_2 - y_2^{\text{target}})$$

Step 2: Compute $\frac{\partial y_2}{\partial z_4}$

The output y_2 is the sigmoid function of the weighted sum z_4:

$$y_2 = \sigma(z_4) = \frac{1}{1 + e^{-z_4}}$$

The derivative of the sigmoid function is:

$$\frac{\partial y_2}{\partial z_4} = y_2(1 - y_2)$$

Step 3: Compute $\frac{\partial z_4}{\partial w_8}$

The weighted sum z_4 for the second output neuron is:

$$z_4 = w_7 \cdot o_1 + w_8 \cdot o_2 + b_2$$

The partial derivative of z_4 with respect to w_8 is:

$$\frac{\partial z_4}{\partial w_8} = o_2$$

`Here:` o_2 `is output at neuron 2 of output layer.`

Step 4: Combine Everything

Now, we can combine all the pieces using the chain rule:

$$\frac{\partial E_{\text{total}}}{\partial w_8} = (y_2 - y_2^{\text{target}}) \cdot y_2(1 - y_2) \cdot o_2$$

This is the gradient of the error with respect to weight w_8.

4. Computing Gradient against $w2, w3, w4$:

1. Computing Gradient for w_2:

Gradient for w_2:

Following a similar procedure as for w_1, the gradient for w_2 is:

$$\frac{\partial E_{\text{total}}}{\partial w_2} = \frac{\partial E_{\text{total}}}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_2}$$

Where:

$$\frac{\partial z_1}{\partial w_2} = x_2$$

Given:

- $x_2 = 0.3$

Thus:

$$\frac{\partial E_{\text{total}}}{\partial w_2} = \frac{\partial E_{\text{total}}}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_1} \cdot 0.3$$

2. Computing Gradient for w_3:

Gradient for w_3:

The gradient for w_3 is:

$$\frac{\partial E_{\text{total}}}{\partial w_3} = \frac{\partial E_{\text{total}}}{\partial o_2} \cdot \frac{\partial o_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_3}$$

Where:

$$\frac{\partial z_2}{\partial w_3} = x_3$$

Given:

- $x_3 = 0.4$

Thus:

$$\frac{\partial E_{\text{total}}}{\partial w_3} = \frac{\partial E_{\text{total}}}{\partial o_2} \cdot \frac{\partial o_2}{\partial z_2} \cdot 0.4$$

3. Computing Gradient for w_4:

Gradient for w_4

Gradient for w_4:
The gradient for w_4 is:

$$\frac{\partial E_{\text{total}}}{\partial w_4} = \frac{\partial E_{\text{total}}}{\partial o_3} \cdot \frac{\partial o_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_4}$$

Where:

$$\frac{\partial z_3}{\partial w_4} = x_4$$

Given:
- $x_4 = 0.5$

Thus:

$$\frac{\partial E_{\text{total}}}{\partial w_4} = \frac{\partial E_{\text{total}}}{\partial o_3} \cdot \frac{\partial o_3}{\partial z_3} \cdot 0.5$$

Q: How can you build a Python program to perform the above operations, allowing similar calculations to be repeated over multiple epochs?

Hint:

Instead of writing a program with numerous for-loops, can we leverage the vectorization and broadcasting capabilities of NumPy? To take advantage of NumPy, the above mathematical representation must first be converted into a matrix form.

Putting Everything in Matrix Notations:

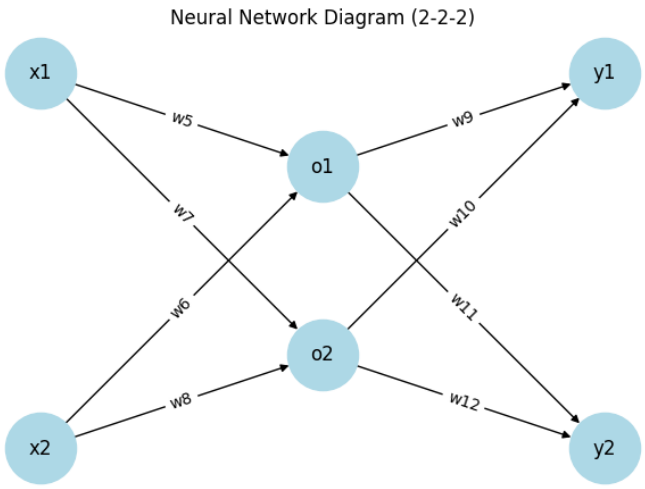

Figure 2: Network to Solve.

Step 1: Define Network Structure

Assume we have:

2 inputs: x_1, x_2
2 hidden neurons: o_1, o_2
2 output neurons: y_1, y_2
Weights:

- $W^{(1)}$ (weights from input to hidden)
- $W^{(2)}$ (weights from hidden to output)

The network connections are:

$$z_3 = w_5 x_1 + w_6 x_2 + b_1 \Rightarrow o_1 = \sigma(z_3)$$
$$z_4 = w_7 x_1 + w_8 x_2 + b_2 \Rightarrow o_2 = \sigma(z_4)$$
$$z_5 = w_9 o_1 + w_{10} o_2 + b_3 \Rightarrow y_1 = \sigma(z_5)$$
$$z_6 = w_{11} o_1 + w_{12} o_2 + b_4 \Rightarrow y_2 = \sigma(z_6)$$

Matrix representation:

$$W^{(1)} = \begin{bmatrix} w_5 & w_6 \\ w_7 & w_8 \end{bmatrix}, W^{(2)} = \begin{bmatrix} w_9 & w_{10} \\ w_{11} & w_{12} \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Hidden layer pre-activation:

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

Applying activation:

$$o = \sigma(z^{(1)})$$

Output layer pre-activation:

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

Applying activation:

$$y = \sigma(z^{(2)})$$

Step 2: Compute Gradients

Compute the error gradients at the output:

$$\frac{\partial E_{total}}{\partial y} = y - y_{target}$$

Apply element-wise sigmoid derivative:

$$\frac{\partial y}{\partial z^{(2)}} = y \odot (1 - y)$$

Calculate the error signal at the output layer:

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

Compute gradient w.r.t. output weights:

$$\frac{\partial E_{total}}{\partial W^{(2)}} = \delta^{(2)} o^T$$

Expanded:

$$\frac{\partial E_{total}}{\partial W^{(2)}} = \begin{bmatrix} \delta_1^{(2)} o_1 & \delta_1^{(2)} o_2 \\ \delta_2^{(2)} o_1 & \delta_2^{(2)} o_2 \end{bmatrix}$$

For the hidden layer:

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

Explicitly:

$$\delta^{(1)} = \begin{bmatrix} w_9 \delta_1^{(2)} + w_{11} \delta_2^{(2)} \\ w_{10} \delta_1^{(2)} + w_{12} \delta_2^{(2)} \end{bmatrix} \odot \begin{bmatrix} o_1(1 - o_1) \\ o_2(1 - o_2) \end{bmatrix}$$

Weight gradient at input layer:

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

Expanded:

$$\frac{\partial E_{total}}{\partial W^{(1)}} = \begin{bmatrix} \delta_1^{(1)} x_1 & \delta_1^{(1)} x_2 \\ \delta_2^{(1)} x_1 & \delta_2^{(1)} x_2 \end{bmatrix}$$

Step 3: Update Weights

Gradient descent updates:

$$W^{(2)} = W^{(2)} - \eta \frac{\partial E_{total}}{\partial W^{(2)}}$$
$$W^{(1)} = W^{(1)} - \eta \frac{\partial E_{total}}{\partial W^{(1)}}$$

Summary of Benefits

- No need for manual weight-by-weight calculations.
- Efficient matrix multiplications (use NumPy, TensorFlow, etc.).
- Scales naturally to larger networks.

Good Luck.