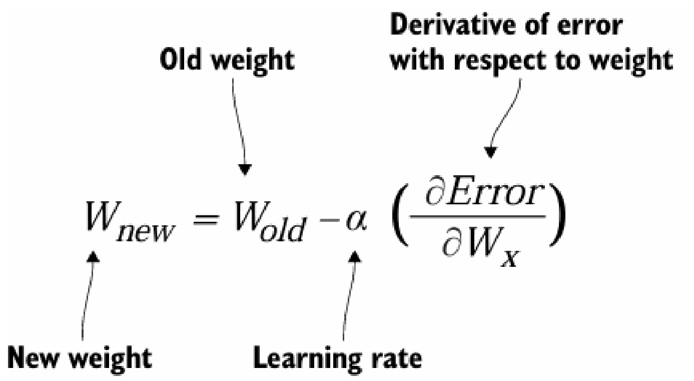
Backpropagation - Simple Notes

What is Backpropagation?

- A method used to update weights in a neural network to minimize error.
- Uses the **chain rule** to compute gradients and adjust weights layer by layer.

Steps in Backpropagation

1. **Forward Pass**: Compute the output using weights and activation functions.
2. **Calculate Error**: Compare predicted output y^ with actual output y.
3. **Backward Pass**:
    a. Compute gradients dE /dW using the **chain rule**.
    b. Adjust weights using **gradient descent**:

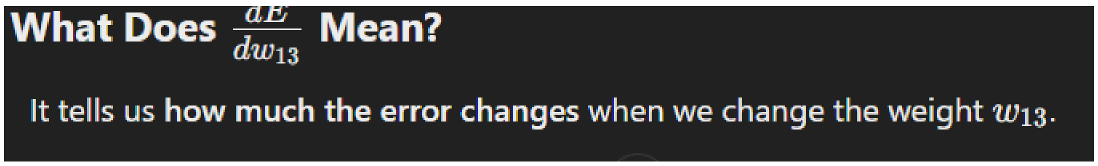


$$W_{new} = W_{old} - \alpha \left(\frac{\partial Error}{\partial W_x}\right)$$

    c. Repeat until error is minimized.

Key Takeaways

- **Gradient descent** reduces error step by step.
- **Hidden layers** learn meaningful patterns through weight updates.
- **The chain rule** allows error to flow backward for correct weight adjustments.

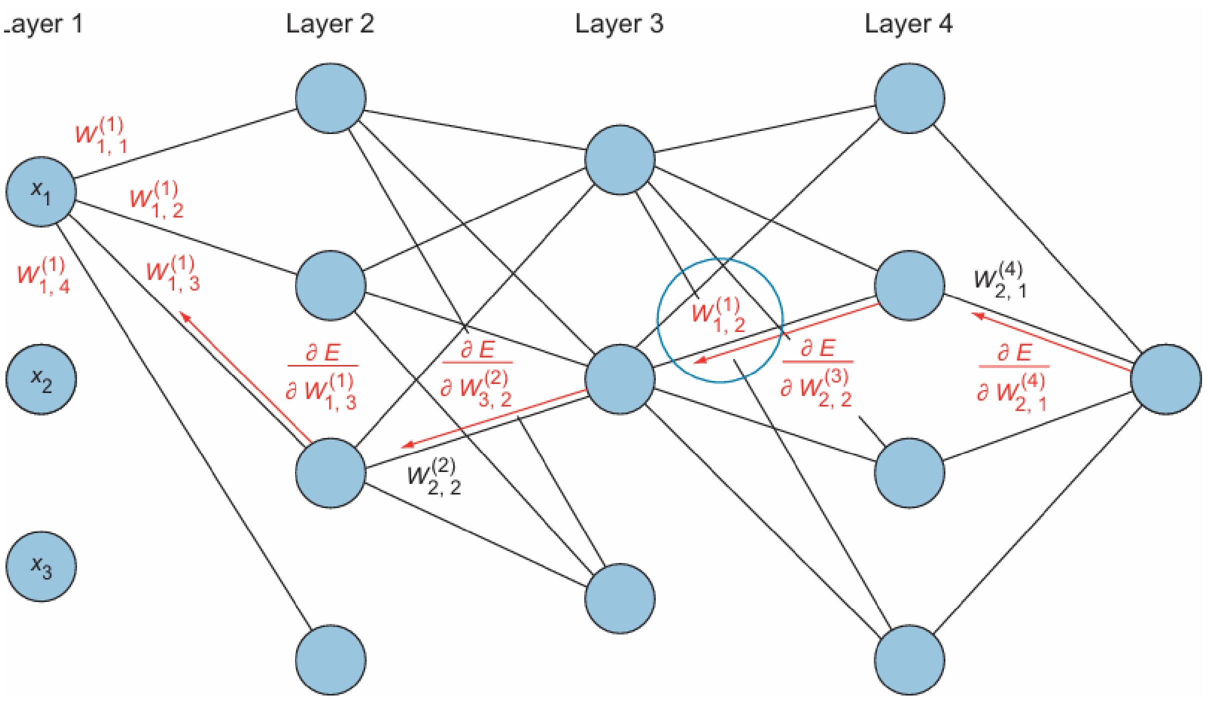


What Does $\frac{dE}{dw_{13}}$ Mean?

It tells us **how much the error changes** when we change the weight w_{13}.

In a dense neural network (also called a fully connected network), weights are adjusted during backpropagation by following a specific route from the output layer back to the input layer. This route is determined by the connections between neurons in the network. Let's break this down step by step in a simple and clear way:

$$\frac{dE}{dw_{1,3}^{(1)}} = \frac{dE}{dw_{2,1}^{(4)}} \times \frac{dw_{2,1}^{(4)}}{dw_{2,2}^{(3)}} \times \frac{dw_{2,2}^{(3)}}{dw_{3,2}^{(2)}} \times \frac{dw_{3,2}^{(2)}}{dw_{1,3}^{(1)}}$$



.ayer 1
Layer 2
Layer 3
Layer 4
x_1
x_2
x_3
$W_{1,1}^{(1)}$
$W_{1,2}^{(1)}$
$W_{1,3}^{(1)}$
$W_{1,4}^{(1)}$
$\frac{\partial E}{\partial W_{1,3}^{(1)}}$
$\frac{\partial E}{\partial W_{3,2}^{(2)}}$
$W_{2,2}^{(2)}$
$W_{1,2}^{(1)}$
$\frac{\partial E}{\partial W_{2,2}^{(3)}}$
$W_{2,1}^{(4)}$
$\frac{\partial E}{\partial W_{2,1}^{(4)}}$

1. Structure of a Dense Network

In a dense network:

Each neuron in a layer is connected to every neuron in the next layer.

Each connection has a weight that needs to be adjusted during training.

The network has an input layer, one or more hidden layers, and an output layer.

2. Why Do We Need a Specific Route?

During backpropagation, we need to calculate the gradient of the error with respect to each weight (dEdw dwdE). This requires us to:

-Start from the output layer (where the error is computed).

-Move backward through the hidden layers.

-Finally reach the input layer.

This is because the error at the output layer depends on the weights in the hidden layers, which in turn depend on the weights in the input layer. The chain rule helps us propagate the error backward through these connections.

3. The Route for Adjusting Weights

The specific route for adjusting weights in a dense network is as follows:

Step 1: Forward Pass

Compute the output of the network using the current weights.

Calculate the error (loss) between the predicted output (y^y^) and the actual output (yy).

Step 2: Backward Pass

Start at the Output Layer:

Compute the gradient of the error with respect to the weights in the output layer.

Formula:

$$\frac{dE}{dw_{\text{output}}} = \frac{dE}{d\hat{y}} \cdot \frac{d\hat{y}}{dw_{\text{output}}}$$

Move to the Last Hidden Layer:

Compute the gradient of the error with respect to the weights in the last hidden layer.

Use the chain rule to propagate the error backward:

$$\frac{dE}{dw_{\text{hidden}}} = \frac{dE}{d\hat{y}} \cdot \frac{d\hat{y}}{dz_{\text{hidden}}} \cdot \frac{dz_{\text{hidden}}}{dw_{\text{hidden}}}$$

Where:

zhidden = weighted sum of inputs to the hidden layer.

Continue Backward Through All Hidden Layers:

Repeat the process for each hidden layer, moving backward toward the input layer.

At each layer, compute:

$$\frac{dE}{dw_{\text{layer}}} = \frac{dE}{dz_{\text{next layer}}} \cdot \frac{dz_{\text{next layer}}}{dz_{\text{current layer}}} \cdot \frac{dz_{\text{current layer}}}{dw_{\text{current layer}}}$$

Reach the Input Layer:

Compute the gradient of the error with respect to the weights in the input layer.

Update the weights using gradient descent:

$$w_{\text{new}} = w_{\text{old}} - \alpha \frac{dE}{dw}$$

Where:

α= learning rate.

4. Why Follow This Route?

Dependencies: The error at the output layer depends on the weights in the hidden layers, which in turn depend on the weights in the input layer. The chain rule allows us to account for these dependencies.

Efficiency: By following this route, we efficiently compute the gradients for all weights in the network without redundant calculations.

Order Matters: We must start from the output layer and move backward because the gradients at each layer depend on the gradients of the next layer.