9. Backward Pass

→ Definition

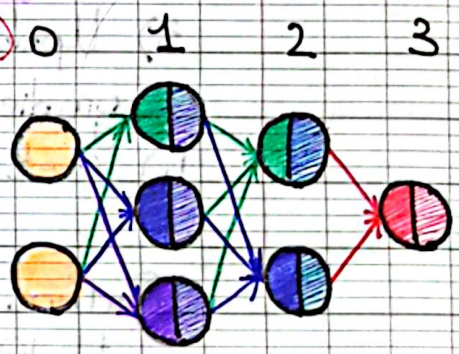
. The backward pass, or backpropagation, adjusts the weights (W) in a neural network by applying the gradient descent principle to the cost function (J) across all layers, starting from the output layer back to the input layer

→ Purpose

. Minimize the cost function (J) by calculating how each weight contributes to the overall error and updating them accordingly.

→ Key Steps in Backward Pass

1) Start at output Layer (Layer3)



. Compute the derivative of the cost function (J) with respect to the weights (W^3) connecting the output layer to the previous layer (2)

$\partial J / \partial W^3$

. Includes weights for connections from:

↳ Layer 2, Index 0 → Layer 3, Index 0

↳ Layer 2, Index 1 → Layer 3, Index 0

$$\frac{\partial J}{\partial w^3} = \left[\frac{\partial J}{\partial W^3_{00}} \quad \frac{\partial J}{\partial W^3_{01}} \right] \cdot \frac{\partial J}{\partial b^3} = \left[\frac{\partial J}{\partial b^3_0} \right]$$

↳ Derivatives also include biases for each connection

2. Move to Hidden Layers

- For layer 2 calculate the derivative of J with respect to all weights (W_2)
  - Include weights for connection from:
    - Layer 1, Index 0 → Layer 2, Index 0
    - Layer 1, Index 1 → Layer 2, Index 0
    - Layer 1, Index 2 → Layer 2, Index 0
    - Layer 1, Index 0 → Layer 2, Index 1
    - Layer 1, Index 1 → Layer 2, Index 1
    - Layer 1, Index 2 → Layer 2, Index 1

$$\frac{dJ}{dw^2} = \begin{bmatrix} dJ/dw^2_{00} & dJ/dw^2_{01} & dJ/dw^2_{02} \\ dJ/dw^2_{10} & dJ/dw^2_{11} & dJ/dw^2_{12} \end{bmatrix} \quad \frac{dJ}{db^2} = \begin{bmatrix} dJ/db^2_0 \\ dJ/db^2_1 \end{bmatrix}$$

- For layer 1 calculate the derivative of J with respect to all weights (W_1)
  - Each weight corresponds to connections from Layer 0 neurons to Layer 1 neurons.

$$\frac{dJ}{dw^1} = \begin{bmatrix} dJ/dw^1_{00} & dJ/dw^1_{01} \\ dJ/dw^1_{10} & dJ/dw^1_{11} \\ dJ/dw^1_{20} & dJ/dw^1_{21} \end{bmatrix} \quad \frac{dJ}{db^1} = \begin{bmatrix} dJ/db^1_0 \\ dJ/db^1_1 \\ dJ/db^1_2 \end{bmatrix}$$

10. Derivates in the Output Layer

→ Overview

- In a back propagation, the first step involves calculating the derivative of the cost function (J) with respect to each weight (w_{ij}^L) in the output layer
- The chain rule is used to break down this derivative into smaller, more quantifiable components.

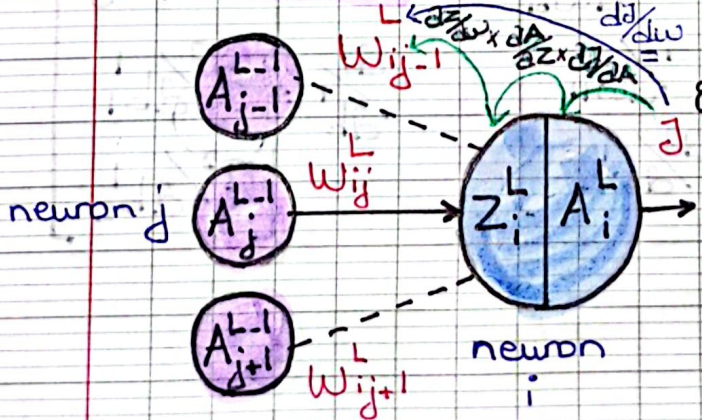
→ Key Terms

- **Cost Function (J)**: Measures the error between the predicted (a_i) and the actual (y) outputs
- **Weighted Sum (Z_i)**: The sum of the weighted inputs plus a bias

$$Z_i^L = w_{i0}^L a_0^{L-1} + w_{i1}^L a_1^{L-1} + w_{i2}^L a_2^{L-1} + \dots + b_j^L$$

- **Activation Function ($G(Z_i)$)**: Produces the neuron's output $a_j^L = G(z_j^L)$

→ Derivative Breakdown: Using the Chain Rule



Our goal is to find $\frac{dJ}{dw_{ij}^L}$, in other words we want to find how sensitive the cost function is to small changes in w

The derivative of J with respect to a weight w_{ij}^L is split into three terms:

$$\frac{\partial J}{\partial w_{ij}^L} = \frac{\partial J}{\partial a_i^L} \cdot \frac{\partial a_i^L}{\partial z_i^L} \cdot \frac{\partial z_i^L}{\partial w_{ij}^L}$$

- Links the cost function (MSE or Cross-Entropy) with the predicted value (a_i^L).
- Nudge in J caused by the nudge in a

- Links the activation function ($a_i^L = G(z_i^L)$) to the weighted sum (z_i^L)
- Nudge in a caused by z

- Links the weighted sum (z_i^L) to the weight (w_{ij}^L)
- Nudge in z caused by w

Calculations

1) Third Term ($\partial z_i^L / \partial w_{ij}^L$)

$$z_i^L = b_i^L + \dots + a_{j-1}^{L-1} \times w_{i,j-1}^L + a_j^{L-1} \times w_{i,j}^L + a_{j+1}^{L-1} \times w_{i,j+1}^L + \dots$$

$$\Rightarrow \frac{\partial z_i^L}{\partial w_{ij}^L} = a_j^{L-1}$$

2) Second Term (da_i^L / dz_i^L)

$$a_i^L = G^L(z_i^L) \Rightarrow \frac{da_i^L}{dz_i^L} = G'^L(z_i^L)$$

3) First Term (dJ / da_i^L)

→ Depends on the cost function
→ J is a function of the output of the NN

Final Formula for Weight Derivatives

$$\frac{dJ}{w_{ij}^L} = J'(a_i^L) \cdot G'^L(z_i^L) \cdot A_i^{L-1}$$

Derivative for Biases

The chain rule applies similarly to biases, but with one key difference:

$$\frac{dJ}{db_i^L} = \frac{dJ}{da_i^L} \cdot \frac{da_i^L}{dz_i^L} \cdot \frac{dz_i^L}{db_i^L} \quad \rightarrow = 1$$

→ Final Formula:

$$\frac{dJ}{db_i^L} = J'(a_i^L) \cdot G'^L(z_i^L)$$

11. Derivates in other layers

- Overview

. The calculation of derivatives in hidden layers follows the same principles as the output layer but is slightly more complex due to the dependency on subsequent layers

. Again our goal is to calculate how the cost function J changes with respect to:

1. Weights (W_{ij}^{L}) - the connections between neurons
2. Biases (b_i^{L}) - the offsets added to the weighted sum

. This is done using the chain rule. we use it because J depends indirectly on W_{ij}^{L} and b_i^{L} through several steps:

1. Each weight W_{ij}^{L} influence the weighted sum z_i^{L}
2. z_i^{L} influence the activation output a_i^{L} of a neuron
3. a_i^{L} influence the cost function J indirectly through the next layer

- Calculate Derivates

1. Derivative of J with respect to W_{ij}^{L}

: Using the chain rule, this derivative can be broken into three terms:

$$\frac{\partial J}{\partial W_{ij}^{L}} = \frac{\partial J}{\partial a_i^{L}} \cdot \frac{\partial a_i^{L}}{\partial z_i^{L}} \cdot \frac{\partial z_i^{L}}{\partial W_{ij}^{L}}$$

- Third Term $\left(\frac{dz_i^L}{dw_j^L}\right)$

  - Recall the formula for the weighted sum

$$z_i^L = \sum_j w_{ij}^L a_j^{L-1} + b_i^L$$

  - The derivative of z_i^L with respect to w_{ij}^L is simply the input from the previous layer

$$\frac{dz_i^L}{dw_j^L} = a_j^{L-1}$$

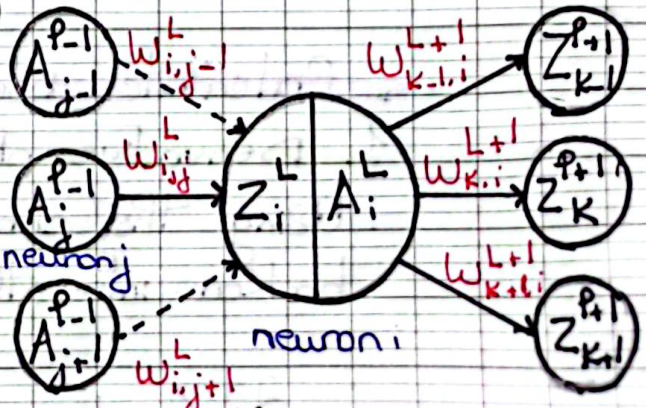
- Second Term $\left(\frac{da_i^L}{dz_i^L}\right)$

  - The activation output a_i^L is related to z_i^L by the activation function $g(z_i^L)$ $a_i^L = g(z_i^L)$
  - The derivative is simply the derivative of the activation function evaluated at z_i^L

$$\frac{da_i^L}{dz_i^L} = g'(z_i^L)$$

- First Term $\left(\frac{dJ}{da_i^L}\right)$

  - The change in a_i^L affects the next layer Specially, a_i^L is used in the weighted sums (z_k^{L+1}) of the neurons in the next layer.



↳ Using the chain rule again, we calculate:

$$\partial J / \partial a_i^L = \sum_K \partial J / \partial z_K^{L+1} \cdot \partial z_K^{L+1} / \partial a_i^L$$

↳ The derivative of z_K^{L+1} with respect to a_i^L is the weight w_{iK}^{L+1}

$$\partial J / \partial a_i^L = \sum_K \partial J / \partial z_K^{L+1} \cdot w_{iK}^{L+1}$$

2. Final Formula for $\partial J / \partial w_{ij}^L$:

$$\frac{\partial J}{\partial w_{ij}^L} = \left(\sum_K \frac{\partial J}{\partial z_K^{L+1}} \cdot w_{iK}^{L+1} \right) \cdot g'(z_i^L) \cdot a_j^{L-1}$$

3. Derivative of J with respect to b_i^L

$$\frac{\partial J}{\partial b_i^L} = \frac{\partial J}{\partial a_i^L} \cdot \frac{\partial a_i^L}{\partial z_i^L} \quad \Rightarrow \text{Since } \frac{\partial z_i^L}{\partial b_i^L} = 1$$

$$\Rightarrow \frac{\partial J}{\partial b_i^L} = \left(\sum \frac{\partial J}{\partial z_K^{L+1}} \cdot w_{iK}^{L+1} \right) \cdot g'(z_i^L)$$

→ Back Propagation (Vectorized Notation)

output layer

$$\begin{cases} \dfrac{\partial J}{\partial W^L} = \left[J'(A^L) \cdot G'^L(Z^L) \right] \times {}^T A^{L-1} \\ \dfrac{\partial J}{b^L} = \sum_{\text{columns}} J'(A^L) \cdot G'^L(Z^L) \end{cases}$$

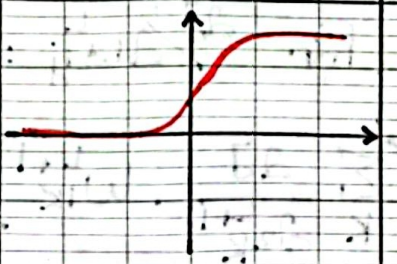
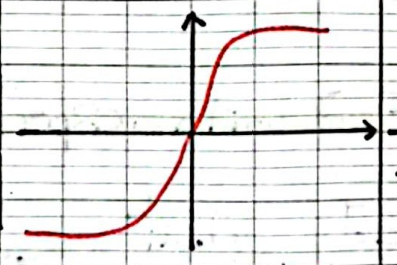
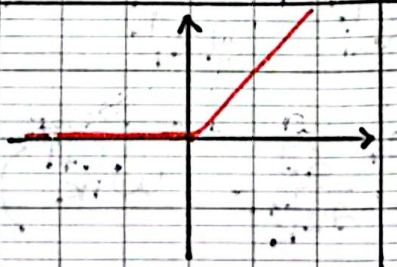
Hidden Layers

$$\frac{\partial J}{\partial w^L} = \left(w^{L+1} \times \frac{\partial J}{\partial Z^{L+1}}\right) \cdot G'^L(Z^L) \times {}^T A^{L-1}$$

$$\frac{\partial J}{\partial b^L} = \sum_{\text{columns}} \left(w^{L+1} \times \frac{\partial J}{\partial Z^{L+1}}\right) \cdot G'^L(Z^L)$$

. : element wise multip.
x: Matrix multiplication

→ Derivatives for activation functions

Name	Plot	G(x)	G'(x)
Sigmoid		$\frac{1}{1+e^{-x}}$	$G(x)(1-G(x))$
tanh		$\frac{2}{1+e^{-2x}} - 1$	$1-(G(x))^2$
ReLU		$\begin{cases} 0 \text{ if } x<0 \\ x \text{ if } x \geq 0 \end{cases}$	$\begin{cases} 0 \text{ if } x<0 \\ 1 \text{ if } x \geq 0 \end{cases}$

→ Derivatives for cost functions

Use Case	Cost Function $J(A_i)$	$\partial J / \partial A_i$
Regression	$\frac{1}{2m} \sum_{i=1}^{m} (A_i - Y_i)^2$	$\frac{1}{m}(A_i - Y_i)$
Binary Class.	$\frac{1}{m} \sum_{i=1}^{m} -Y_i \log(A_i) - (1-Y_i)\log(1-A_i)$	$\frac{1}{m}\left(-\frac{Y_i}{A_i} + \frac{1-Y_i}{1-A_i}\right)$

12. Weights and Bias Updates

→ Purpose

We are using gradient descent to adjust the weights (W) and biases (b) in a neural network to minimize the cost function J

→ The Gradient Descent Update Rule

$$W := W - \alpha \cdot \frac{\partial J}{\partial W} \quad , \quad b := b - \alpha \cdot \frac{\partial J}{\partial b}$$

- W : current weight matrix
- $\partial J / \partial W$: gradient of the cost function with respect to the weights (calculated in backpropagation)
- α : learning rate
- b : current bias
- $\partial J / \partial b$: gradient of the cost function with respect to the biases

→ Recursive Nature of Updates

The weight and bias updates are performed in multiple iterations

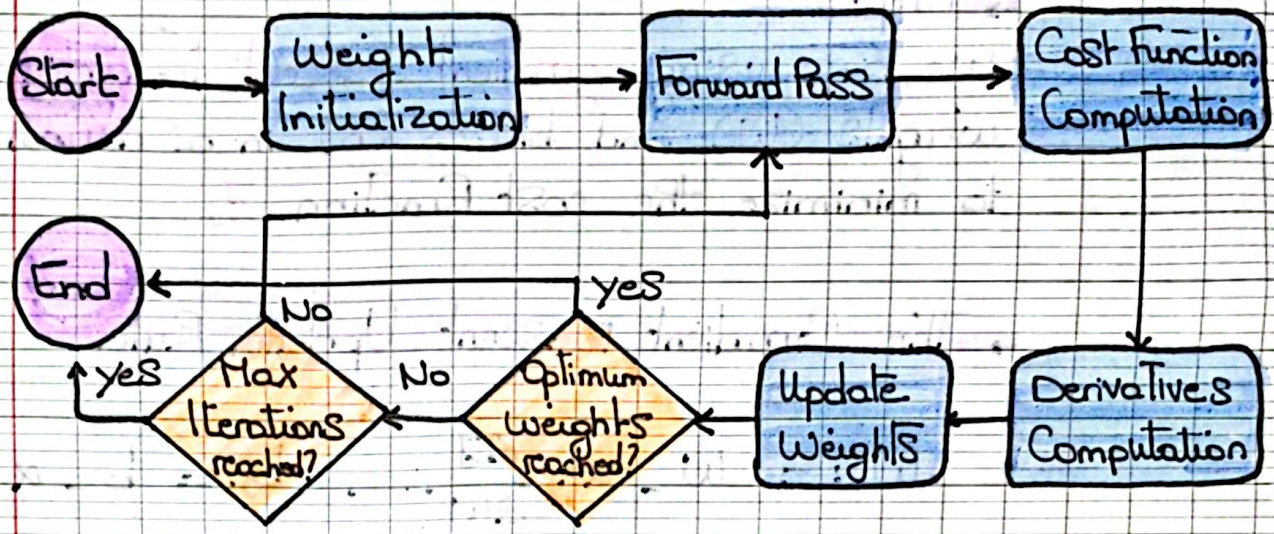
At each iteration, the updated weights and biases are used in the next forward pass.

We stop in 2 cases :

↳ we reach the maximum nb of iterations

↳ we reach an optimum value for the weight

→ Steps of the Learning Process



1. Initialize Weights and Biases Randomly

- At the start, weights and biases are initialized to random values (e.g. small numbers close to zero)

2. Forward Pass

- Input data is passed through the network to calculate the predicted output
- For each layer:

$$z^L = W^L \cdot a^{L-1} + b^L$$

$$a^L = g(z^L)$$

z^L: Weighted Sum

a^L: Activation output after applying an activation function $g(z)$

3. Compute the Cost Function

- Compare the predicted output with the true label to calculate the error using the cost function (MSE or cross-entropy)

4. Backward Pass (Backpropagation)

- Gradients ($\frac{\partial J}{\partial w}$ and $\frac{\partial J}{\partial b}$) are calculated using the chain rule.
- These gradients tell us how to adjust the weights and biases to reduce the error.

5. Update Weights and Biases

- Using the gradients and the gradient descent formula, adjust the weights and biases to reduce the cost function.

6. Repeat Until Stopping Criteria

- Maximum Number of Iterations
- Optimum Found (i.e., gradient is close to zero)
- No more Data (no more batches of data to process)