Gradient Descent

Gradient Descent is the most basic and most important method for training neural networks. The idea is simple: adjust the network's parameters (weights and biases) to reduce the loss, meaning make the prediction closer and closer to the real value.

1) What we are trying to minimize

Training a neural network means finding a vector of parameters, called θ, that minimizes the loss function. With a dataset (x_i, y_i), we define:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(x_i), y_i)$$

Here:

- $f_\theta(x)$ is the network's prediction with parameters θ
- ℓ is the individual loss (e.g., MSE)
- $L(\theta)$ is the total error of the network

The goal is to find a θ that makes $L(\theta)$ as small as possible.

2) The intuitive idea in the one-variable case

Imagine a case with a single parameter, called w, and a loss $L(w)$.

The derivative $L'(w)$ indicates how the loss changes when you move w:

- if $L'(w) > 0$: increasing w makes the loss go up
- if $L'(w) < 0$: increasing w makes the loss go down

Therefore, to decrease $L(w)$, you should update w by "walking" to the derivative:

$$w_{new} = w_{old} - \eta L'(w_{old})$$

The number η is the learning rate, controlling the step size.
This is the core of gradient descent.

3) The mathematical justification

Near a point w, you can approximate the function using first-order Taylor expansion:

$$L(w + \Delta w) \approx L(w) + L'(w)\,\Delta w$$

Now choose the step:

$$\Delta w = -\eta L'(w)$$

$L(w)$: perda antes de atualizar

$L(w + \Delta w)$: perda no ponto atualizado

$\Delta w = -\eta L'(w)$: o novo valor da função é menor ou igual ao anterior.

Substituting:

$$L(w + \Delta w) \approx L(w) + L'(w)\,(-\eta\, L'(w))$$

$$L(w + \Delta w) \approx L(w) - \eta\,[L'(w)]^2$$

↳ this term is always positive

Therefore:

$$L(w + \Delta w) \leq L(w)$$

Meaning that small steps always push the loss downward. This is what guarantees that the update rule makes mathematical sense.

4) Generalizing: many parameters at once

Now consider that the neural network has many parameters. We group everything in a vector:

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_d \end{bmatrix}$$

loss downward: impulsionar a perda para baixo.

steepest ascent: direção de maior subida da função.

The loss function becomes:

$$L : \mathbb{R}^d \to \mathbb{R}$$

The analogue of the derivative is the gradient:

$$\nabla_\theta L(\theta) = \begin{bmatrix} \frac{\partial L}{\partial \theta_1} \\ \frac{\partial L}{\partial \theta_2} \\ \vdots \\ \frac{\partial L}{\partial \theta_d} \end{bmatrix}$$

This vector points in the direction of steepest ascent of the function. Thus, to go down, we do the opposite:

$$\theta_{new} = \theta_{old} - \eta \nabla_\theta L(\theta_{old})$$

5) Justification in the vector case

The Taylor expansion for vector-valued function says:

$$L(\theta + \Delta\theta) \approx L(\theta) + \nabla_\theta L(\theta)^T \Delta\theta$$

If we choose

$$\Delta\theta = -\eta \nabla_\theta L(\theta)$$

then substituting gives: ↗ never negative

$$L(\theta + \Delta\theta) \approx L(\theta) - \eta \|\nabla_\theta L(\theta)\|^2$$

The loss goes down.

7) How the neural network computes de gradient

The function $L(\theta)$ depends on parameters spread across many layers, activations, nonlinearities, etc. To compute all derivatives:

$$\frac{\partial L}{\partial \theta_k}$$

the network uses **backpropagation**, which is simply the chain rule written efficiently.

- the forward pass stores all intermediate values.
- the backward pass computes the gradients in reverse order.

Once backprop finishes, you already have $\nabla_\theta L(\theta)$ and can apply:

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$$

atribuition

8) The complete training routine

Training with gradient descent follows this cycle:

1. initialize parameters θ_0
2. for each step t:
   - compute the loss $L(\theta_t)$

- compute the gradient $\nabla L(\theta t)$
- update :

$$\theta t+1 = \theta t - \eta \nabla L(\theta t)$$

3. repeat until convergence or until reaching the required number of epochs.

Backpropagation is the algorithm that efficiently computes the gradient of the loss function with respect to all parameters of a neural network. It applies the chain rule layer by layer, moving from the output back toward the input, reusing intermediate quantities from the forward pass to avoid redundant calculations. This allows the network to determine how each weight contributed to the final error and therefore how it should be updated during training.