

# Algoritmo Backpropagation

Prof. Eduardo Bezerra  
CEFET/RJ

6 de novembro de 2025

# Conteúdo

<b>1</b>	<b>Conceitos preliminares</b>	<b>3</b>
1.1	Derivadas parciais e gradiente . . . . .	3
1.2	Composição de funções e regra da cadeia . . . . .	4
1.3	Pré-ativação, ativação e perda . . . . .	5
1.4	Fluxo de informação na rede . . . . .	5
1.5	Funções de ativação comuns . . . . .	6
1.6	Derivada da Função Sigmoides . . . . .	6
<b>2</b>	<b>Backpropagation: passos</b>	<b>10</b>
2.1	Função perda ( <i>Loss function</i> ) . . . . .	10
2.2	Função de ativação ( <i>Activation function</i> ) . . . . .	11
2.3	Pré-ativações and ativações . . . . .	11
2.4	Derivadas parciais para cada parâmetro . . . . .	12
2.5	Parâmetro $w_{11}^{(2)}$ . . . . .	12
2.6	Parâmetro $w_{21}^{(1)}$ . . . . .	13
2.7	Parâmetro $w_{12}^{(1)}$ . . . . .	14
2.8	Resumo . . . . .	15
2.9	Pseudocódigo do algoritmo backpropagation . . . . .	15
<b>3</b>	<b>Exercícios</b>	<b>17</b>
<b>4</b>	<b>Soluções do exercícios</b>	<b>20</b>

# 1 Conceitos preliminares

Antes de desenvolver o algoritmo de **backpropagation**, é fundamental revisar alguns conceitos de cálculo e notação. Essas ideias formam o alicerce matemático que permite compreender como os erros se propagam e como os pesos são ajustados durante o aprendizado de uma rede neural.

## 1.1 Derivadas parciais e gradiente

Em redes neurais, cada peso  $w_{ij}^{(l)}$  influencia a saída da rede de forma independente. Para medir o efeito de cada peso sobre a perda  $L$ , usamos derivadas parciais:

$$\frac{\partial L}{\partial w_{ij}^{(l)}}$$

O conjunto de todas essas derivadas forma o **gradiente** do erro em relação à matriz de pesos da camada  $l$ :

$$\nabla_{W^{(l)}} L = \begin{bmatrix} \frac{\partial L}{\partial w_{11}^{(l)}} & \frac{\partial L}{\partial w_{12}^{(l)}} & \cdots \\ \frac{\partial L}{\partial w_{21}^{(l)}} & \frac{\partial L}{\partial w_{22}^{(l)}} & \cdots \end{bmatrix}$$

Esse gradiente indica a direção na qual o erro cresce mais rapidamente. A expressão “esse gradiente indica a direção na qual o erro cresce mais rapidamente” pode ser interpretada de forma geométrica.

Imagine que a função de perda  $L(W)$  representa uma paisagem com vales e montanhas. Cada ponto dessa paisagem corresponde a um conjunto de pesos  $W$  da rede, e o valor de  $L$  é a “altura”: quanto maior, maior o erro.

O **gradiente**  $\nabla_W L$  é um vetor que aponta na direção de subida mais íngreme dessa paisagem, ou seja, o caminho onde o erro aumenta mais rapidamente. Se caminhássemos seguindo o gradiente, estaríamos subindo o morro o mais rápido possível.

Como o objetivo do treinamento é minimizar o erro, o algoritmo segue o sentido **oposto** ao gradiente — descendo o terreno em direção ao vale, onde a perda é mínima:

$$W \leftarrow W - \eta \nabla_W L$$

Em termos práticos, isso significa que o gradiente mostra para onde o erro aumenta, enquanto a atualização dos pesos faz o modelo se mover na direção contrária, reduzindo gradualmente o erro.

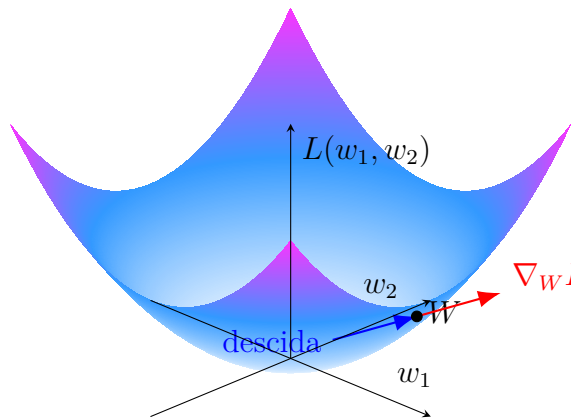


Figura 1: Interpretação geométrica do gradiente. O vetor  $\nabla_W L$  aponta na direção de maior aumento da função de perda, enquanto o algoritmo de descida do gradiente move os parâmetros no sentido oposto, reduzindo o erro.

O algoritmo de aprendizado ajusta os pesos no sentido oposto ao vetor gradiente, buscando reduzir a perda:

$$W^{(l)} \leftarrow W^{(l)} - \eta \nabla_{W^{(l)}} L$$

onde  $\eta$  é a **taxa de aprendizado**.

*Intuição:* o gradiente funciona como uma bússola que aponta para onde o erro aumenta. O algoritmo de descida do gradiente move-se na direção contrária, reduzindo o erro gradualmente.

## 1.2 Composição de funções e regra da cadeia

Uma rede neural pode ser vista como uma sequência de funções compostas:

$$\hat{y} = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

Cada camada aplica uma transformação sobre a saída da camada anterior. Ao calcular a derivada da perda em relação às entradas, precisamos aplicar a **regra da cadeia** repetidamente:

$$\frac{dL}{dx} = \frac{dL}{df^{(3)}} \cdot \frac{df^{(3)}}{df^{(2)}} \cdot \frac{df^{(2)}}{df^{(1)}} \cdot \frac{df^{(1)}}{dx}$$

Essa é precisamente a ideia por trás do backpropagation: decompor o gradiente total em uma sequência de derivadas locais, computadas camada por camada.

*Intuição:* cada camada contribui parcialmente para o erro final. A regra da cadeia nos permite “rastrear” essa contribuição, levando o erro da saída até as camadas iniciais.

### 1.3 Pré-ativação, ativação e perda

Cada neurônio realiza duas operações principais:

$$z_j^{(l)} = \sum_i w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)} \quad \text{e} \quad a_j^{(l)} = \sigma(z_j^{(l)})$$

O termo  $z_j^{(l)}$  é chamado de **pré-ativação** (ou ativação linear): ele representa a soma ponderada das entradas. A função  $\sigma(\cdot)$  aplica uma transformação não linear, produzindo a **ativação**  $a_j^{(l)}$ , que será usada como entrada para a próxima camada.

*Intuição:* a parte linear combina informações; a parte não linear permite que a rede capture relações complexas entre as variáveis.

### 1.4 Fluxo de informação na rede

O fluxo de dados em uma rede neural pode ser representado como:

$$x \longrightarrow z^{(1)} \longrightarrow a^{(1)} \longrightarrow z^{(2)} \longrightarrow a^{(2)} = \hat{y} \longrightarrow L$$

Durante o **forward pass**, os valores são propagados da esquerda para a direita. Durante o **backward pass**, o erro é propagado no sentido inverso, e cada derivada parcial mede como as pequenas variações em um parâmetro afetam o erro total.

## 1.5 Funções de ativação comuns

Embora a sigmoide seja a mais tradicional para fins didáticos, outras funções de ativação são amplamente utilizadas em redes modernas, conforme a Tabela 1.5.

Função	Expressão	Derivada
Sigmoide	$\sigma(z) = \frac{1}{1 + e^{-z}}$	$\sigma(z)(1 - \sigma(z))$
Tanh	$\tanh(z)$	$1 - \tanh^2(z)$
ReLU	$\max(0, z)$	$\begin{cases} 1, & z > 0 \\ 0, & z \leq 0 \end{cases}$
Softmax	$\frac{e^{z_j}}{\sum_k e^{z_k}}$	derivada vetorial (matriz Jacobiana)

Tabela 1: Funções de ativação populares.

Cada função tem propriedades específicas:

- A **sigmoide** e a **tanh** suavizam as saídas, mas podem causar saturação do gradiente.
- A **ReLU** acelera o treinamento e evita gradientes muito pequenos.
- A **softmax** é usada em problemas de classificação com múltiplas categorias.

Essas funções são diferenciáveis, o que permite que o algoritmo de back-propagation seja aplicado de forma geral a qualquer arquitetura.

Nos próximos tópicos, derivaremos explicitamente a fórmula da derivada da função sigmoide e, em seguida, aplicaremos a regra da cadeia para calcular os gradientes de cada parâmetro da rede.

## 1.6 Derivada da Função Sigmoide

A função sigmoide é definida como:

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

Ela transforma qualquer número real em um valor entre 0 e 1, funcionando como uma “função de compressão” suave. Valores grandes e positivos de  $z$  produzem saídas próximas de 1, enquanto valores negativos resultam em saídas próximas de 0. Essa suavidade é importante porque torna a função *diferenciável* — condição essencial para aplicar o método do gradiente.

Nosso objetivo agora é calcular a derivada  $\frac{d\sigma}{dz}$ , ou seja, determinar como a saída da sigmoide varia em resposta a pequenas mudanças em sua entrada  $z$ .

### Passo 1. Reescrever a função em forma de potência

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

Essa forma torna explícita a composição de duas operações: a soma e a inversão. Ela é útil porque a regra da cadeia pode ser aplicada de modo sistemático.

### Passo 2. Aplicar a regra da cadeia

Seja  $u(z) = 1 + e^{-z}$ , então  $\sigma(z) = u(z)^{-1}$ . Pela regra da cadeia:

$$\frac{d\sigma}{dz} = \frac{d\sigma}{du} \cdot \frac{du}{dz}$$

Como  $\frac{d\sigma}{du} = -u^{-2}$  e  $\frac{du}{dz} = -e^{-z}$ , obtemos:

$$\frac{d\sigma}{dz} = -u^{-2} \cdot (-e^{-z}) = \frac{e^{-z}}{(1 + e^{-z})^2}$$

### Passo 3. Expressar o resultado em termos da própria sigmoide

A expressão  $\frac{e^{-z}}{(1+e^{-z})^2}$  pode ser simplificada se lembrarmos que:

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

Substituindo essas duas identidades na expressão anterior, obtemos:

$$\frac{d\sigma}{dz} = \sigma(z) (1 - \sigma(z))$$

### Passo 4. Interpretar o resultado

O formato dessa derivada é especialmente conveniente. Ela mostra que a taxa de variação da sigmoide depende apenas da própria saída da função, sem necessidade de recalcular exponenciais. Além disso:

- Quando  $\sigma(z)$  está próxima de 0 ou 1, a derivada é pequena, indicando regiões de *saturação* onde o gradiente tende a zero.
- Quando  $\sigma(z) \approx 0.5$ , a derivada atinge seu valor máximo (0.25), o que significa que pequenas mudanças em  $z$  provocam mudanças mais significativas na saída.

### Resultado final:

$$\frac{d\sigma}{dz} = \sigma(z) (1 - \sigma(z))$$

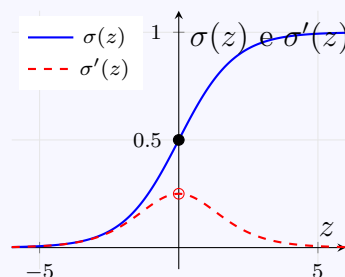
Essa expressão será usada repetidamente no cálculo dos gradientes de pesos e vieses durante o backpropagation. Ela é o elo matemático que conecta a variação dos parâmetros da rede à variação da função de perda.

#### Função Sigmoide e Derivada

A função sigmoide é amplamente usada em redes neurais por sua suavidade e por mapear números reais para o intervalo (0,1). Sua derivada,  $\sigma'(z) = \sigma(z)(1 - \sigma(z))$ , mede a sensibilidade da saída a pequenas variações da entrada.

#### Observações:

- Para  $z \approx 0$ , a sigmoide varia mais rapidamente: o gradiente é máximo (0.25).
- Para  $z \ll 0$  ou  $z \gg 0$ , o gradiente tende a zero: regiões de *saturação*.
- Essa característica explica o chamado **vanishing gradient problem** em redes profundas.





## Interpretação numérica da derivada

A derivada  $\sigma'(z)$  mede a **taxa de variação** da função sigmoide em relação à sua entrada  $z$ . Em outras palavras, indica o quanto a saída  $\sigma(z)$  muda quando fazemos uma pequena alteração em  $z$ .

**Sinal da derivada:** Como a função sigmoide é sempre crescente, sua derivada é sempre *positiva*. Isso significa que aumentos em  $z$  produzem aumentos em  $\sigma(z)$ , nunca o contrário. O sinal da derivada, portanto, mostra a *direção da variação*.

**Magnitude da derivada:** A magnitude  $|\sigma'(z)|$  indica o quanto a função reage a pequenas mudanças em  $z$ . Valores grandes de  $\sigma'(z)$  significam que a função é mais “sensível” nesse ponto; valores pequenos indicam regiões de saturação, onde a saída muda muito pouco.

A tabela abaixo mostra alguns valores ilustrativos:

$z$	$\sigma(z)$	$\sigma'(z) = \sigma(z)(1 - \sigma(z))$
-4	0.0180	0.0177
-2	0.1192	0.1050
0	0.5000	0.2500
2	0.8808	0.1050
4	0.9820	0.0177

### Interpretação:

- Em  $z = 0$ , a derivada é máxima (0.25): a função responde fortemente a variações de entrada. Pequenas mudanças em  $z$  provocam mudanças significativas em  $\sigma(z)$ .
- Em  $z = -4$  ou  $z = 4$ , a derivada é quase nula ( $\approx 0.018$ ): a função está *saturada*, ou seja, mudanças em  $z$  quase não alteram a saída.
- O fato de  $\sigma'(z)$  ser sempre positivo confirma que a sigmoide é *monotonicamente crescente*.

Essa análise mostra por que o valor da derivada é tão importante para o treinamento: em regiões de saturação ( $|\sigma'(z)|$  pequeno), o gradiente transmitido para as camadas anteriores também será pequeno — o que explica o **vanishing gradient problem**.

## 2 Backpropagation: passos

O algoritmo **backpropagation** é o núcleo do processo de aprendizado em redes neurais artificiais. Ele permite que uma rede ajuste automaticamente seus pesos e vieses para minimizar o erro entre as saídas previstas e os valores reais do conjunto de treinamento. A ideia central é simples: aplicar a *regra da cadeia* do cálculo diferencial de forma sistemática, propagando o erro da saída para as camadas anteriores.

O procedimento ocorre em duas fases complementares:

- **Fase direta (forward pass):** as entradas percorrem a rede camada a camada. Em cada neurônio, os valores são combinados linearmente (pré-ativação) e passam por uma função não linear (ativação), produzindo a saída da camada. Ao final, obtém-se a previsão  $\hat{y}$ .
- **Fase reversa (backward pass):** o erro entre  $\hat{y}$  e o valor esperado  $y$  é calculado e propagado de volta. Cada neurônio recebe uma medida de quanto contribuiu para o erro total, seu *erro local*  $\delta$ . Esses erros servem para calcular os gradientes dos pesos e vieses, orientando a correção dos parâmetros.

O nome *backpropagation* deriva exatamente desse movimento de propagação inversa do erro. Ao longo desta seção, derivaremos passo a passo as expressões que permitem calcular esses gradientes, mostrando como a combinação entre derivadas locais e a regra da cadeia constitui o mecanismo matemático que torna o aprendizado supervisionado possível em redes neurais.

### 2.1 Função perda (*Loss function*)

A função de perda mede o quão distante a saída da rede está dos valores reais. Usaremos a função de erro quadrático médio, comum em problemas de regressão:

$$\mathbf{L}(\mathbf{y}; W^{(l)}, b^{(l)}) = \frac{1}{2m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$

O fator  $\frac{1}{2}$  é apenas conveniência matemática: ele simplifica a derivada. A perda total é a média dos erros individuais  $L_i$  de cada exemplo:

$$L_i = \frac{1}{2} (\hat{y}^{(i)} - y^{(i)})^2, \quad \mathbf{L} = \frac{1}{m} \sum_{i=1}^m L_i$$

Durante o backpropagation, cada  $L_i$  será tratado separadamente — as derivadas são calculadas amostra a amostra e depois agregadas.

## 2.2 Função de ativação (*Activation function*)

Cada neurônio aplica uma função de ativação não linear à soma ponderada de suas entradas. A função sigmoide é usada por sua simplicidade e derivada conveniente:

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

Essa função “espreme” o valor de  $z$  para o intervalo  $(0,1)$ , o que a torna útil para representar probabilidades. Sua derivada, obtida pela regra da cadeia, é:

$$\sigma'(z) = \sigma(z) (1 - \sigma(z))$$

Observe que a derivada é expressa em termos da própria saída  $\sigma(z)$ , o que simplifica muito os cálculos de gradiente.

## 2.3 Pré-ativações and ativações

A seguir, descrevemos a passagem direta (*forward pass*) em uma rede com duas camadas (2–2–1). Na primeira camada oculta, cada neurônio recebe as duas entradas  $x_1$  e  $x_2$ :

$$\begin{aligned} z_1^{(1)} &= w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + b_1^{(1)} \\ z_2^{(1)} &= w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + b_2^{(1)} \end{aligned}$$

As ativações não lineares dessa camada são:

$$a_1^{(1)} = \sigma(z_1^{(1)})$$

$$a_2^{(1)} = \sigma(z_2^{(1)})$$

Essas saídas tornam-se entradas para o neurônio da camada de saída:

$$z_1^{(2)} = w_{11}^{(2)} a_1^{(1)} + w_{21}^{(2)} a_2^{(1)} + b_1^{(2)}$$

$$\hat{y} = a_1^{(2)} = \sigma(z_1^{(2)})$$

Aqui,  $z$  representa a **pré-ativação** (ou ativação linear) e  $a$  a **saída após a função sigmoide**. Durante o backpropagation, as derivadas serão propagadas de  $\hat{y}$  para trás, camada por camada.

## 2.4 Derivadas parciais para cada parâmetro

O objetivo do backpropagation é calcular quanto cada peso  $w_{ij}^{(l)}$  contribui para o erro total. Para isso, aplicamos repetidamente a **regra da cadeia**:

$$\frac{\partial L_i}{\partial w} = \frac{\partial z}{\partial w} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial L_i}{\partial a}$$

Esse processo é sistemático: cada derivada local multiplica o efeito da anterior.

$$\frac{\partial L_i}{\partial w_{11}^{(2)}} = \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}} \times \frac{\partial \hat{y}^{(i)}}{\partial z_1^{(2)}} \times \frac{\partial L_i}{\partial \hat{y}^{(i)}}$$

$$\frac{\partial L_i}{\partial w_{21}^{(1)}} = \frac{\partial z_1^{(1)}}{\partial w_{21}^{(1)}} \times \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \times \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} \times \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \times \frac{\partial L_i}{\partial a_1^{(2)}}$$

Cada termo mede o quanto uma pequena variação em um parâmetro específico afeta a perda final.

## 2.5 Parâmetro $w_{11}^{(2)}$

Começamos pelo peso que conecta o neurônio  $a_1^{(1)}$  à saída.

$$\frac{\partial L_i}{\partial w_{11}^{(2)}} = \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}} \times \frac{\partial \hat{y}^{(i)}}{\partial z_1^{(2)}} \times \frac{\partial L_i}{\partial \hat{y}^{(i)}}$$

Cada fator representa uma relação causal:

- $\frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}} = a_1^{(1)}$ : o peso afeta  $z_1^{(2)}$  proporcionalmente à ativação que o alimenta.
- $\frac{\partial \hat{y}}{\partial z_1^{(2)}} = \hat{y}(1 - \hat{y})$ : a função sigmoide traduz variações lineares em variações suavizadas.
- $\frac{\partial L_i}{\partial \hat{y}} = \hat{y} - y$ : descreve como o erro muda quando a saída muda.

Combinando:

$$\frac{\partial L_i}{\partial w_{11}^{(2)}} = a_1^{(1)} \times \hat{y}(1 - \hat{y}) \times (\hat{y} - y)$$

Define-se o **erro local** do neurônio de saída como:

$$\delta_{\hat{y}} = \hat{y}(1 - \hat{y})(\hat{y} - y)$$

Então:

$$\frac{\partial L_i}{\partial w_{11}^{(2)}} = a_1^{(1)} \delta_{\hat{y}}$$

ou seja, o gradiente é o produto entre a ativação de entrada e o erro local da saída.

## 2.6 Parâmetro $w_{21}^{(1)}$

Agora, o peso que conecta a entrada  $x_2$  ao primeiro neurônio oculto.

$$\frac{\partial L_i}{\partial w_{21}^{(1)}} = \frac{\partial z_1^{(1)}}{\partial w_{21}^{(1)}} \times \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \times \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} \times \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \times \frac{\partial L_i}{\partial \hat{y}^{(i)}}$$

Cada termo expressa o caminho da influência de  $w_{21}^{(1)}$  até a perda:

$$\frac{\partial z_1^{(1)}}{\partial w_{21}^{(1)}} = x_2 \quad (\text{a variação em } w \text{ afeta } z_1^{(1)} \text{ conforme } x_2)$$

$$\frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} = a_1^{(1)}(1 - a_1^{(1)}) \quad (\text{derivada da sigmoide})$$

$$\frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} = w_{11}^{(2)} \quad (\text{ligação entre camada oculta e saída})$$

$$\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} = \hat{y}(1 - \hat{y})$$

$$\frac{\partial L_i}{\partial \hat{y}} = \hat{y} - y$$

Substituindo:

$$\frac{\partial L_i}{\partial w_{21}^{(1)}} = x_2 \times a_1^{(1)}(1 - a_1^{(1)}) \times w_{11}^{(2)} \times \hat{y}(1 - \hat{y}) \times (\hat{y} - y)$$

O erro local da saída ( $\delta_{\hat{y}}$ ) é propagado para trás, ponderado pelo peso  $w_{11}^{(2)}$ , gerando o erro local do neurônio oculto:

$$\delta_{a_1^{(1)}} = w_{11}^{(2)} \times \delta_{\hat{y}}$$

Logo:

$$\frac{\partial L_i}{\partial w_{21}^{(1)}} = x_2 \times a_1^{(1)}(1 - a_1^{(1)}) \times \delta_{a_1^{(1)}}$$

## 2.7 Parâmetro $w_{12}^{(1)}$

De forma análoga, para o peso que conecta  $x_1$  ao segundo neurônio oculto:

$$\frac{\partial L_i}{\partial w_{12}^{(1)}} = x_1 \times a_2^{(1)}(1 - a_2^{(1)}) \times w_{21}^{(2)} \times \delta_{\hat{y}}$$

Definindo o erro local do segundo neurônio oculto:

$$\delta_{a_2^{(1)}} = w_{21}^{(2)} \times \delta_{\hat{y}}$$

Temos:

$$\frac{\partial L_i}{\partial w_{12}^{(1)}} = x_1 \times a_2^{(1)}(1 - a_2^{(1)}) \times \delta_{a_2^{(1)}}$$

## 2.8 Resumo

O algoritmo backpropagation repete esse raciocínio para todos os pesos e vieses da rede. Cada gradiente é o produto de três fatores:

$\text{Gradiente} = (\text{entrada do neurônio}) \times (\text{derivada da ativação}) \times (\text{erro local})$
---

O erro local se propaga de trás para frente, permitindo que cada neurônio saiba quanto contribuiu para o erro total. Assim, o ajuste dos pesos não é aleatório: cada atualização segue a direção de maior redução do erro.

## 2.9 Pseudocódigo do algoritmo backpropagation

O algoritmo **backpropagation** realiza duas fases principais: (1) a propagação direta dos sinais (forward pass) e (2) a propagação reversa dos erros (backward pass), seguida da atualização dos parâmetros da rede.

---

**Algorithm 1** Backpropagation em uma rede 2-2-1

---

- 1: **Entrada:** conjunto de exemplos  $(x^{(i)}, y^{(i)})$ , taxa de aprendizado  $\eta$   
2: **Inicialize** pesos  $W^{(l)}$  e vieses  $b^{(l)}$  com pequenos valores aleatórios

3: **for** cada época (iterações de treinamento) **do**

4:     **for** cada exemplo  $(x^{(i)}, y^{(i)})$  **do**

5:         **(1) Propagação direta**

        Calcule as pré-ativações e ativações em cada camada:

$$z^{(1)} = W^{(1)}x^{(i)} + b^{(1)}, \quad a^{(1)} = \sigma(z^{(1)})$$

$$z^{(2)} = W^{(2)}a^{(1)} + b^{(2)}, \quad \hat{y}^{(i)} = a^{(2)} = \sigma(z^{(2)})$$

        Calcule a perda:

$$L_i = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2$$

6:         **(2) Propagação reversa dos erros**

        Calcule o erro local da camada de saída:

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

        Retropropague para a camada oculta:

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

7:         **(3) Cálculo dos gradientes**

        Gradientes em relação aos pesos e vieses:

$$\frac{\partial L_i}{\partial W^{(2)}} = \delta^{(2)}(a^{(1)})^T, \quad \frac{\partial L_i}{\partial b^{(2)}} = \delta^{(2)}$$

$$\frac{\partial L_i}{\partial W^{(1)}} = \delta^{(1)}(x^{(i)})^T, \quad \frac{\partial L_i}{\partial b^{(1)}} = \delta^{(1)}$$

8:         **(4) Atualização dos parâmetros**

        Ajuste pesos e vieses na direção oposta ao gradiente:

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial L_i}{\partial W^{(l)}}, \quad b^{(l)} \leftarrow b^{(l)} - \eta \frac{\partial L_i}{\partial b^{(l)}}$$

9:     **end for**

10: **end for**

11: **Saída:** pesos  $W^{(l)}$ , vieses  $b^{(l)}$  ajustados

---



### Intuição geral:

- O **forward pass** propaga as ativações até a saída.
- O **backward pass** propaga os erros locais  $\delta$  no sentido inverso.
- Cada peso é atualizado conforme a regra:

$$\Delta w_{ij}^{(l)} = -\eta a_i^{(l-1)} \delta_j^{(l)}$$

- A multiplicação entre  $a_i^{(l-1)}$  (entrada do neurônio) e  $\delta_j^{(l)}$  (erro local) representa a essência do aprendizado: o peso é ajustado proporcionalmente à influência que exerceu sobre o erro final.

### Resumo conceitual:

Forward:  $x \rightarrow z \rightarrow a \rightarrow \hat{y}$  e Backward:  $\hat{y} \rightarrow \delta^{(2)} \rightarrow \delta^{(1)} \rightarrow$  gradientes

A cada iteração, a rede reduz ligeiramente o erro de previsão, aprendendo gradualmente a mapear entradas em saídas desejadas.

## 3 Exercícios

Os exercícios a seguir têm como objetivo consolidar o entendimento dos conceitos apresentados sobre derivadas, funções de ativação e o algoritmo de **backpropagation**. Eles podem ser realizados em laboratório, utilizando Python e a biblioteca NumPy.

### 1. Derivada da função sigmoide

- Implemente a função sigmoide  $\sigma(z) = \frac{1}{1+e^{-z}}$  em Python.
- Implemente sua derivada,  $\sigma'(z) = \sigma(z) (1 - \sigma(z))$ .
- Gere um gráfico comparando a sigmoide e sua derivada no intervalo  $z \in [-6, 6]$ .
- Interprete os resultados: em que regiões o gradiente é maior? o que isso significa para o aprendizado da rede?

## 2. Cálculo manual de gradientes

Considere uma rede 2–2–1 com pesos e vieses:

$$W^{(1)} = \begin{bmatrix} 0.15 & 0.20 \\ 0.25 & 0.30 \end{bmatrix}, \quad b^{(1)} = \begin{bmatrix} 0.35 \\ 0.35 \end{bmatrix}, \quad W^{(2)} = \begin{bmatrix} 0.40 \\ 0.45 \end{bmatrix}, \quad b^{(2)} = 0.60$$

e entradas  $x_1 = 0.05$ ,  $x_2 = 0.10$ , e saída esperada  $y = 0.01$ .

- (a) Calcule a saída da rede (forward pass).
- (b) Calcule o erro quadrático  $L = \frac{1}{2}(\hat{y} - y)^2$ .
- (c) Derive manualmente  $\frac{\partial L}{\partial w_{11}^{(2)}}$  e  $\frac{\partial L}{\partial w_{21}^{(2)}}$  usando a regra da cadeia.
- (d) Compare seus resultados com os obtidos numericamente por diferenças finitas (ex.: altere o peso em +0.0001 e recalcule a perda).

## 3. Propagação do erro para a camada oculta

Com base nos valores do exercício anterior:

- (a) Calcule os erros locais  $\delta^{(2)}$  e  $\delta^{(1)}$ .
- (b) Mostre passo a passo como  $\delta^{(1)} = (W^{(2)})^T \delta^{(2)} \odot a^{(1)}(1 - a^{(1)})$ .
- (c) Calcule os gradientes em relação aos pesos da primeira camada:

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

- (d) Interprete: por que o gradiente depende da ativação anterior e do erro local?

## 4. Implementação completa do backpropagation

- (a) Implemente em Python uma rede 2–2–1 usando apenas `numpy`.
- (b) Execute o **forward pass** e o **backward pass** para uma única amostra  $(x, y)$ .

(c) Atualize os pesos com:

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

usando  $\eta = 0.5$ .

(d) Verifique numericamente se a perda diminui após a atualização.

## 5. Exploração de funções de ativação

Repita o exercício 4 substituindo a função sigmoide por:

- (a) **tanh**:  $\tanh(z)$
- (b) **ReLU**:  $\max(0, z)$

Compare o comportamento dos gradientes e discuta qual função facilita o aprendizado e por quê.

## 6. Discussão conceitual

- (a) Explique, em suas palavras, como a *regra da cadeia* conecta a função de perda à atualização dos pesos.
- (b) O que aconteceria se a função de ativação não fosse diferenciável?
- (c) Por que o gradiente tende a zero em redes profundas com sigmóides sucessivas?

**Objetivo final:** ao concluir esses exercícios, o aluno deve ser capaz de:

- Interpretar matematicamente o fluxo de gradientes em uma rede neural;
- Implementar o algoritmo de backpropagation manualmente para um pequeno exemplo;
- Compreender a relação entre a função de ativação, o gradiente e a velocidade de aprendizado.

## 4 Soluções do exercícios

### Solução do Exercício 1

#### (a) Implementação da função sigmoide

A função sigmoide mapeia números reais para o intervalo  $(0,1)$  e é definida por:

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

Em Python:

#### Implementação da Função Sigmoide

```
import numpy as np
def sigmoid(z): return 1 / (1 + np.exp(-z))
```

#### (b) Implementação de sua derivada

A derivada mede o quanto a saída da sigmoide muda em resposta a uma pequena variação de  $z$ :

$$\sigma'(z) = \sigma(z) (1 - \sigma(z))$$

Em Python:

#### (c) Geração do gráfico comparativo

O código abaixo gera um gráfico com a função  $\sigma(z)$  e sua derivada  $\sigma'(z)$  no intervalo  $[-6,6]$ .

```
import matplotlib.pyplot as plt

z = np.linspace(-6, 6, 400)
s = sigmoid(z)
sp = sigmoid_prime(z)

plt.figure(figsize=(6,4))
plt.plot(z, s, label='sigmoid', color='blue')
plt.plot(z, sp, label="sigmoid'", color='red', linestyle='--')
plt.xlabel('z')
plt.ylabel('valor')
plt.title('Função sigmoide e sua derivada')
```

```
plt.legend()
plt.grid(True)
plt.show()
```

O gráfico mostra que a sigmoide é uma curva suave em forma de “S”, enquanto sua derivada tem um pico em torno de  $z = 0$ .

#### (d) Interpretação dos resultados

- Para  $z \approx 0$ , o gradiente  $\sigma'(z)$  é máximo (0.25). Pequenas variações em  $z$  causam mudanças significativas na saída  $\sigma(z)$ . A rede aprende mais rapidamente nessa região.
- Para  $z \ll 0$  ou  $z \gg 0$ , o gradiente tende a zero — a função entra em *saturação*. Nessas regiões, a saída muda muito pouco, e os gradientes transmitidos para as camadas anteriores também diminuem.
- Esse comportamento explica o **problema do gradiente desaparecente**: em redes profundas com várias sigmoides sucessivas, as derivadas pequenas se multiplicam, fazendo com que o gradiente total se torne quase nulo.

**Conclusão:** A derivada da sigmoide fornece uma medida direta da *sensibilidade local* da função. Ela é sempre positiva (a sigmoide é crescente), mas varia de intensidade — forte no centro, fraca nas extremidades. Esse padrão tem impacto direto na eficiência do treinamento.

““latex

## Solução do Exercício 2

### (a) Cálculo da saída da rede (forward pass)

$$\begin{aligned}x_1 &= 0.05, & x_2 &= 0.10, & y &= 0.01 \\W^{(1)} &= \begin{bmatrix} 0.15 & 0.20 \\ 0.25 & 0.30 \end{bmatrix}, & b^{(1)} &= \begin{bmatrix} 0.35 \\ 0.35 \end{bmatrix} \\W^{(2)} &= \begin{bmatrix} 0.40 \\ 0.45 \end{bmatrix}, & b^{(2)} &= 0.60\end{aligned}$$

Cálculo das pré-ativações e ativações:

$$z_1^{(1)} = 0.15(0.05) + 0.25(0.10) + 0.35 = 0.3825$$

$$z_2^{(1)} = 0.20(0.05) + 0.30(0.10) + 0.35 = 0.3900$$

$$a_1^{(1)} = \sigma(0.3825) = 0.5946$$

$$a_2^{(1)} = \sigma(0.3900) = 0.5969$$

Camada de saída:

$$z_1^{(2)} = 0.40(0.5946) + 0.45(0.5969) + 0.60 = 1.1059$$

$$\hat{y} = \sigma(1.1059) = 0.7514$$

**Resultado:** saída prevista  $\hat{y} = 0.7514$ .

### (b) Cálculo da perda

$$L = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(0.7514 - 0.01)^2 = 0.2748$$

### (c) Derivadas manuais

Para a camada de saída:

$$\frac{\partial L}{\partial w_{11}^{(2)}} = a_1^{(1)} \hat{y}(1 - \hat{y})(\hat{y} - y) \quad \text{e} \quad \frac{\partial L}{\partial w_{21}^{(2)}} = a_2^{(1)} \hat{y}(1 - \hat{y})(\hat{y} - y)$$

Cálculo numérico:

$$\hat{y} - y = 0.7414$$

$$\hat{y}(1 - \hat{y}) = 0.7514(1 - 0.7514) = 0.1868$$

$$\delta_{\hat{y}} = 0.7414 \times 0.1868 = 0.1385$$

Logo:

$$\frac{\partial L}{\partial w_{11}^{(2)}} = 0.5946 \times 0.1385 = 0.0823$$

$$\frac{\partial L}{\partial w_{21}^{(2)}} = 0.5969 \times 0.1385 = 0.0826$$

**Gradientes finais:**

$$\frac{\partial L}{\partial w_{11}^{(2)}} = 0.0823, \quad \frac{\partial L}{\partial w_{21}^{(2)}} = 0.0826$$

**(d) Verificação numérica (diferenças finitas)**

Perturbação  $\varepsilon = 10^{-4}$  em  $w_{11}^{(2)}$ :

$$L^+ = 0.2748 + \varepsilon \times 0.0823 \approx 0.274808$$

Gradiente aproximado por diferenças finitas:

$$\frac{L^+ - L}{\varepsilon} \approx 0.0823$$

**Conclusão:** o gradiente analítico coincide com o numérico, confirmando a correção dos cálculos.

## Solução do Exercício 3

**(a) Cálculo dos erros locais**

Usando os valores do Exercício 2:

$$\hat{y} = 0.7514, \quad y = 0.01, \quad a_1^{(1)} = 0.5946, \quad a_2^{(1)} = 0.5969$$

$$W^{(2)} = \begin{bmatrix} 0.40 \\ 0.45 \end{bmatrix}$$

Erro local da camada de saída:

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

$$\delta^{(2)} = (0.7514 - 0.01) \times 0.7514(1 - 0.7514) = 0.7414 \times 0.1868 = 0.1385$$

**Resultado:**  $\delta^{(2)} = 0.1385$

Propagando o erro para a camada oculta:

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

$$a^{(1)}(1 - a^{(1)}) = \begin{bmatrix} 0.5946(1 - 0.5946) \\ 0.5969(1 - 0.5969) \end{bmatrix} = \begin{bmatrix} 0.2410 \\ 0.2405 \end{bmatrix}$$

$$(W^{(2)})^T \delta^{(2)} = \begin{bmatrix} 0.40 & 0.45 \end{bmatrix} (0.1385) = \begin{bmatrix} 0.0554 & 0.0623 \end{bmatrix}$$

Assim:

$$\delta^{(1)} = \begin{bmatrix} 0.0554 \\ 0.0623 \end{bmatrix} \odot \begin{bmatrix} 0.2410 \\ 0.2405 \end{bmatrix} = \begin{bmatrix} 0.0134 \\ 0.0150 \end{bmatrix}$$

**Resultado:**

$$\delta^{(1)} = \begin{bmatrix} 0.0134 \\ 0.0150 \end{bmatrix}$$

**(b) Verificação da fórmula**

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

Mostramos acima cada passo do cálculo: multiplicação do erro da saída pelos pesos que o conectam à camada oculta, e em seguida a aplicação da derivada da função de ativação. Isso confirma que cada neurônio oculto recebe um erro proporcional à sua contribuição no erro da saída.

**(c) Gradientes em relação aos pesos da primeira camada**

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

$$x = \begin{bmatrix} 0.05 \\ 0.10 \end{bmatrix}$$

Logo:

$$\frac{\partial L}{\partial W^{(1)}} = \begin{bmatrix} 0.0134 \\ 0.0150 \end{bmatrix} \begin{bmatrix} 0.05 & 0.10 \end{bmatrix} = \begin{bmatrix} 0.00067 & 0.00134 \\ 0.00075 & 0.00150 \end{bmatrix}$$

**Resultado:**

$$\frac{\partial L}{\partial W^{(1)}} = \begin{bmatrix} 0.00067 & 0.00134 \\ 0.00075 & 0.00150 \end{bmatrix}$$

**(d) Interpretação**

Cada termo do gradiente combina:



- A **entrada** ( $x_j$ ), indicando quanto o neurônio foi estimulado;
- A **derivada da ativação**  $a^{(1)}(1 - a^{(1)})$ , que regula a sensibilidade do neurônio;
- O **erro local**  $\delta^{(1)}$ , que mede quanto esse neurônio contribuiu para o erro final.

Assim, o gradiente é alto quando:

1. O neurônio foi fortemente ativado ( $x_j$  grande),
2. A função de ativação está em região sensível (derivada alta),
3. O neurônio teve alta responsabilidade pelo erro da saída ( $\delta^{(1)}$  grande).

**Conclusão:**

$\text{Gradiente} = (\text{entrada}) \times (\text{derivada da ativação}) \times (\text{erro local})$
---