Algoritmo Backpropagation

Prof. Eduardo Bezerra
CEFET/RJ

9 de outubro de 2025

Conteúdo

1 Conceitos preliminares **3**
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 Sigmoide 6

2 Backpropagation: passos **10**
2.1 Função perda (*Loss function*) 10
2.2 Função de ativação (*Activation function*) 11
2.3 Pré-activações 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

3 Exercícios **17**
3.1 1. Derivada da função sigmoide 17
3.2 2. Cálculo manual de gradientes 18
3.3 3. Propagação do erro para a camada oculta 18

3.4 4. Implementação completa do backpropagation 18
3.5 5. Exploração de funções de ativação 19
3.6 6. Discussão conceitual 19

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.

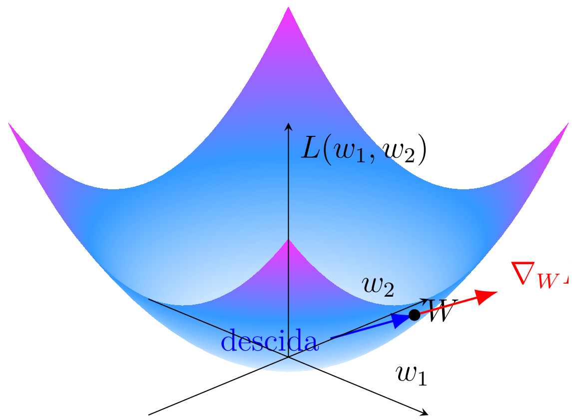


Figura 1: Interpretação geométrica do gradiente. O vetor $\nabla_W L$ aponta na direção de maior aumento da função de perda, enquanto o algoritmo de descida do gradiente move os parâmetros no sentido oposto, reduzindo o erro.

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 η 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 backpropagation 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.

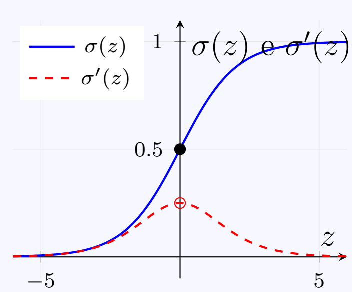
Resultado final:

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

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 (≈ 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* δ. 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:

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

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:

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

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-activações 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:

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

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 $w_{11}^{(2)}$

Comecemos 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:

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

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

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

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

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

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:

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

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

Entrada: conjunto de exemplos $(x^{(i)}, y^{(i)})$, taxa de aprendizado η

Inicialize pesos $W^{(l)}$ e vieses $b^{(l)}$ com pequenos valores aleatórios

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

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

    (1) Propagação direta

    Calcule as pré-ativações e ativações em cada camada:

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

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

    Calcule a perda:

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

    (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:

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

    (3) Cálculo dos gradientes

    Gradientes em relação aos pesos e vieses:

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

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

    (4) Atualização dos parâmetros

    Ajuste pesos e vieses na direção oposta ao gradiente:

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

  end for

end for

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 δ 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:

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

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`.

3.1 1. Derivada da função sigmoide

(a) Implemente a função sigmoide $\sigma(z) = \frac{1}{1+e^{-z}}$ em Python.

(b) Implemente sua derivada, $\sigma'(z) = \sigma(z)\,(1 - \sigma(z))$.

(c) Gere um gráfico comparando a sigmoide e sua derivada no intervalo $z \in [-6, 6]$.

(d) Interprete os resultados: em que regiões o gradiente é maior? o que isso significa para o aprendizado da rede?

3.2 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).

3.3 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?

3.4 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.

3.5 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.

3.6 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 sigmoides 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.