



Introdução às Redes Neurais

Prof. Vinicius de Carvalho Rispoli (FGA/UnB)
21 e 26 de outubro de 2025

Sumário

- Parte 1
 - Introdução ao aprendizado de máquinas
- Parte 2
 - Estrutura básica de uma rede neural
- Parte 3
 - Sobre a convergência das redes neurais
- Parte 4
 - Outros assuntos

Parte 1

INTELIGÊNCIA ARTIFICIAL

Qualquer técnica que permita as máquinas de imitar o comportamento humano



APRENDIZADO DE MÁQUINA

Habilidade de aprender sem ser explicitamente programado para a tarefa



APRENDIZADO PROFUNDO

Excelente para extrair padrões de dados usando redes neurais



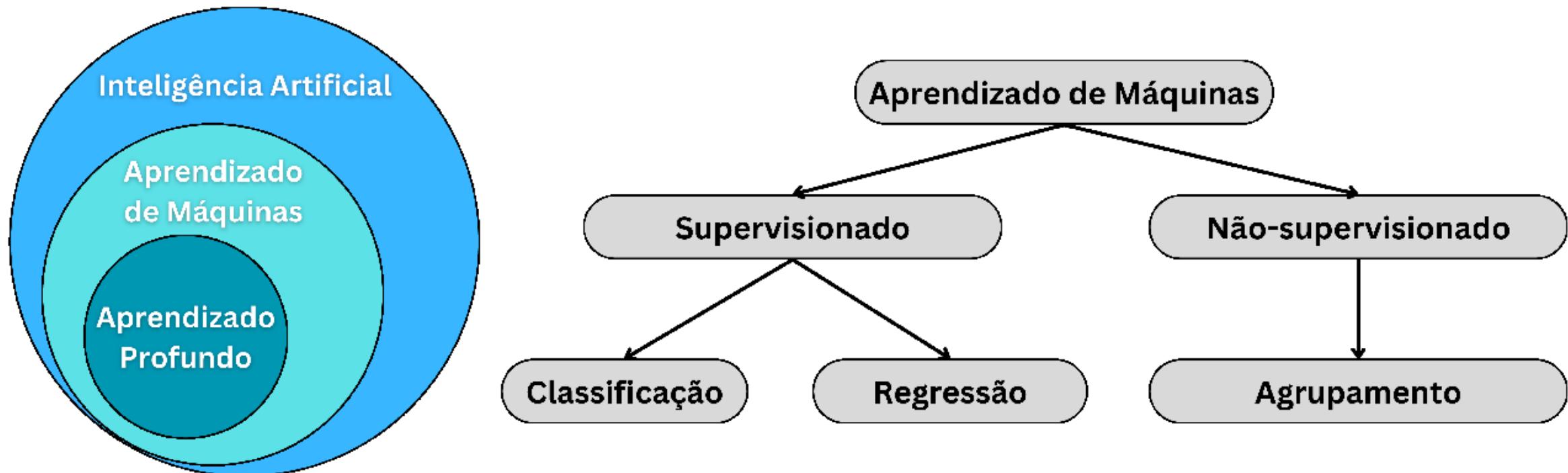
O que é a inteligência artificial?

A inteligência artificial aproveita o uso de computadores e máquinas para imitar as capacidades de resolução de problemas e tomada de decisão da mente humana (IBM)

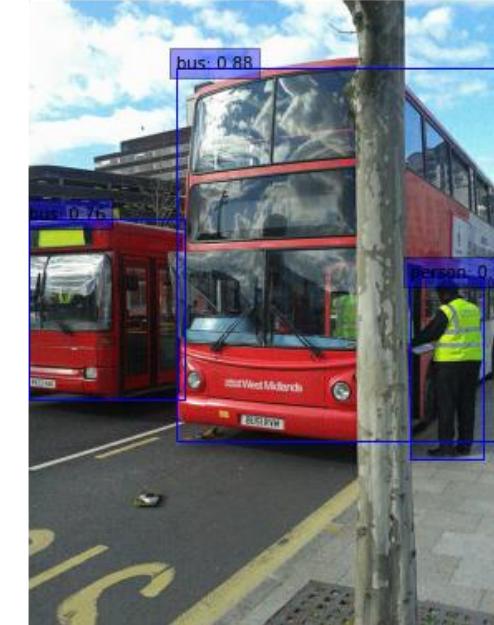
Embora a inteligência artificial (ML) seja muitas vezes considerada um sistema em si, ela é um conjunto de tecnologias implementadas num sistema para lhe permitir raciocinar, aprender e agir para resolver um problema complexo.

O que é o aprendizado de máquinas?

- IA é o conceito mais amplo de permitir que uma máquina ou sistema sinta, raciocine, aja ou se adapte como um ser humano
- ML é uma aplicação de IA que permite que máquinas extraiam conhecimento de dados e aprendam com eles de forma autônoma



- É definido pelo uso de conjuntos de dados rotulados para treinar algoritmos que classificam dados ou prevêem resultados com precisão.
- Requerimentos:
 - dados rotulados para treinar o modelo e avaliá-lo
 - uma métrica para avaliar o resultado do modelo criado
- Tipos de problemas: classificação e regressão

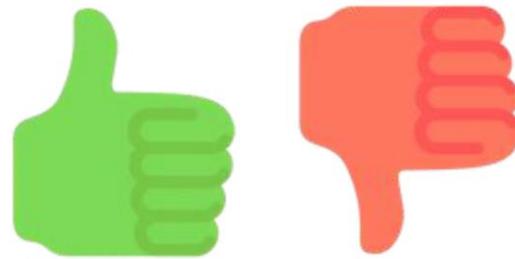


Aprendizado Não-Supervisionado

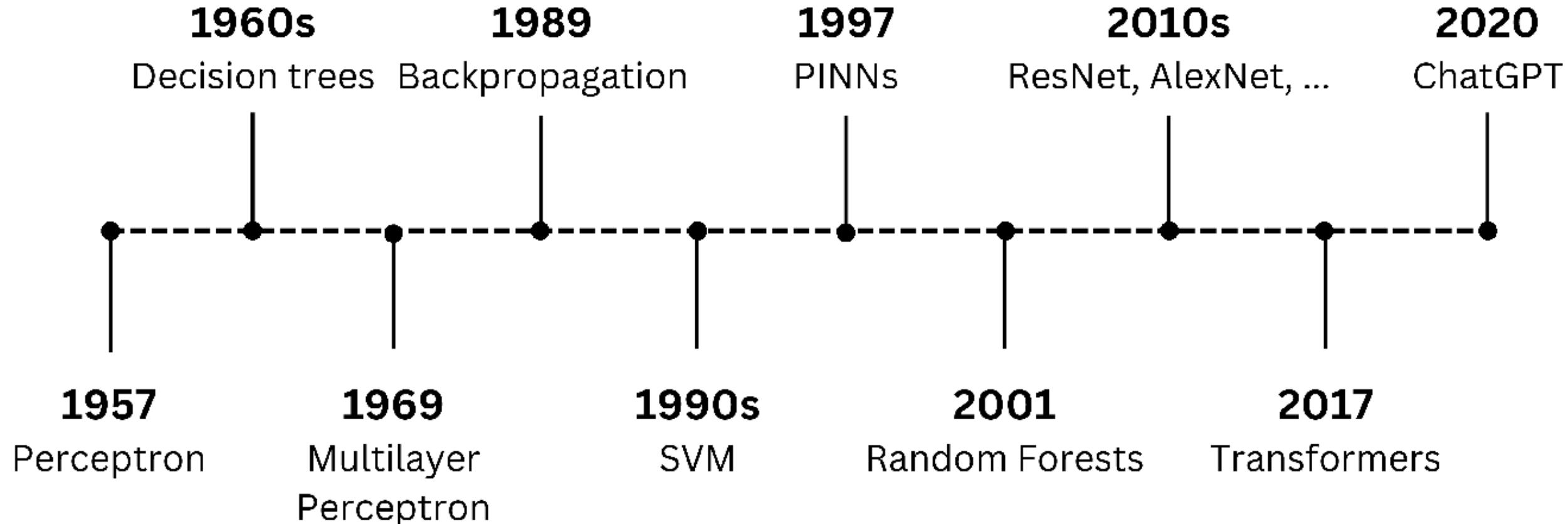
- Objetivo: Descobrir subgrupos (exemplos similares) em um conjunto de dados
- Requerimentos:
 - dados de treinamento para treinar o modelo e avaliá-lo
 - **NÃO** precisa de uma variável resposta



Segmentar grupos de clientes



Sistemas de recomendação de produtos



O que é o aprendizado de máquinas?

Uma lista de técnicas de aprendizado de máquinas:

- Naïve Bayes
- Decision trees
- Support vector machines
- K-means
- K-nearest neighbors
- Regressão linear e logística
- Random forests
- Gradient boosting
- **Redes neurais artificiais**
- ...

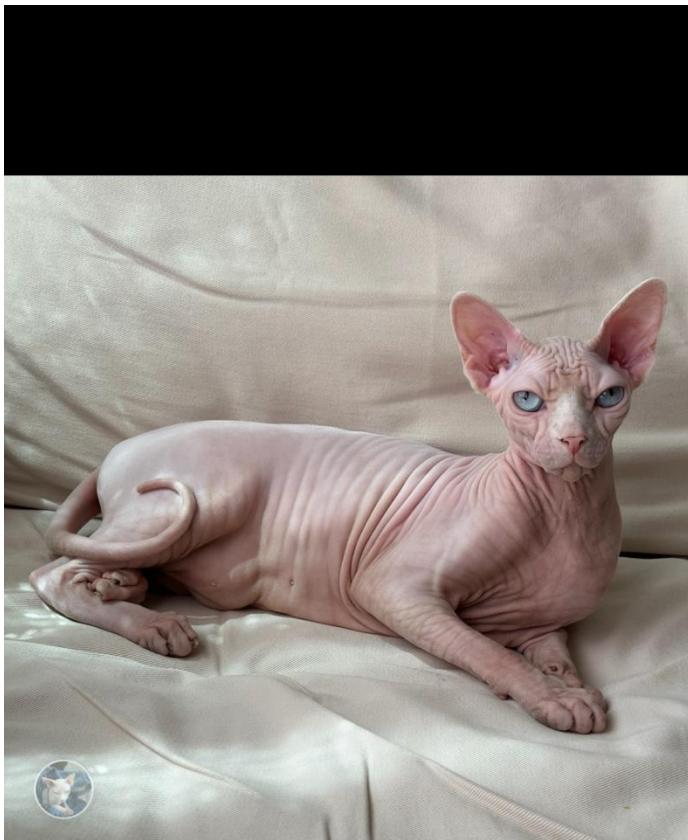
Aplicações do aprendizado de máquinas

Classificação



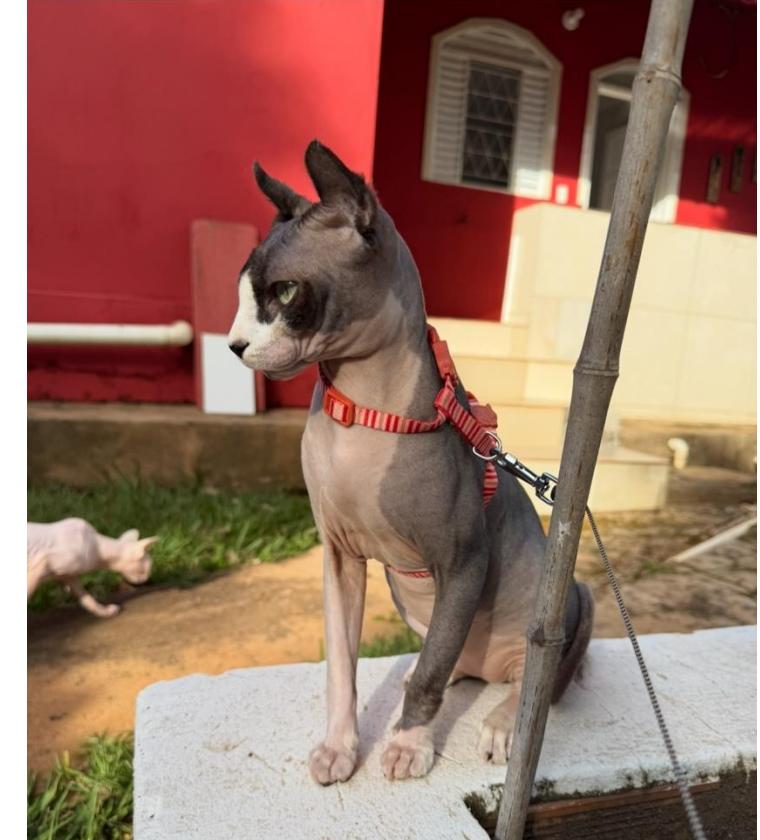
Add a Caption

 Look Up Maine Coon >



Add a Caption

 Look Up Sphynx >

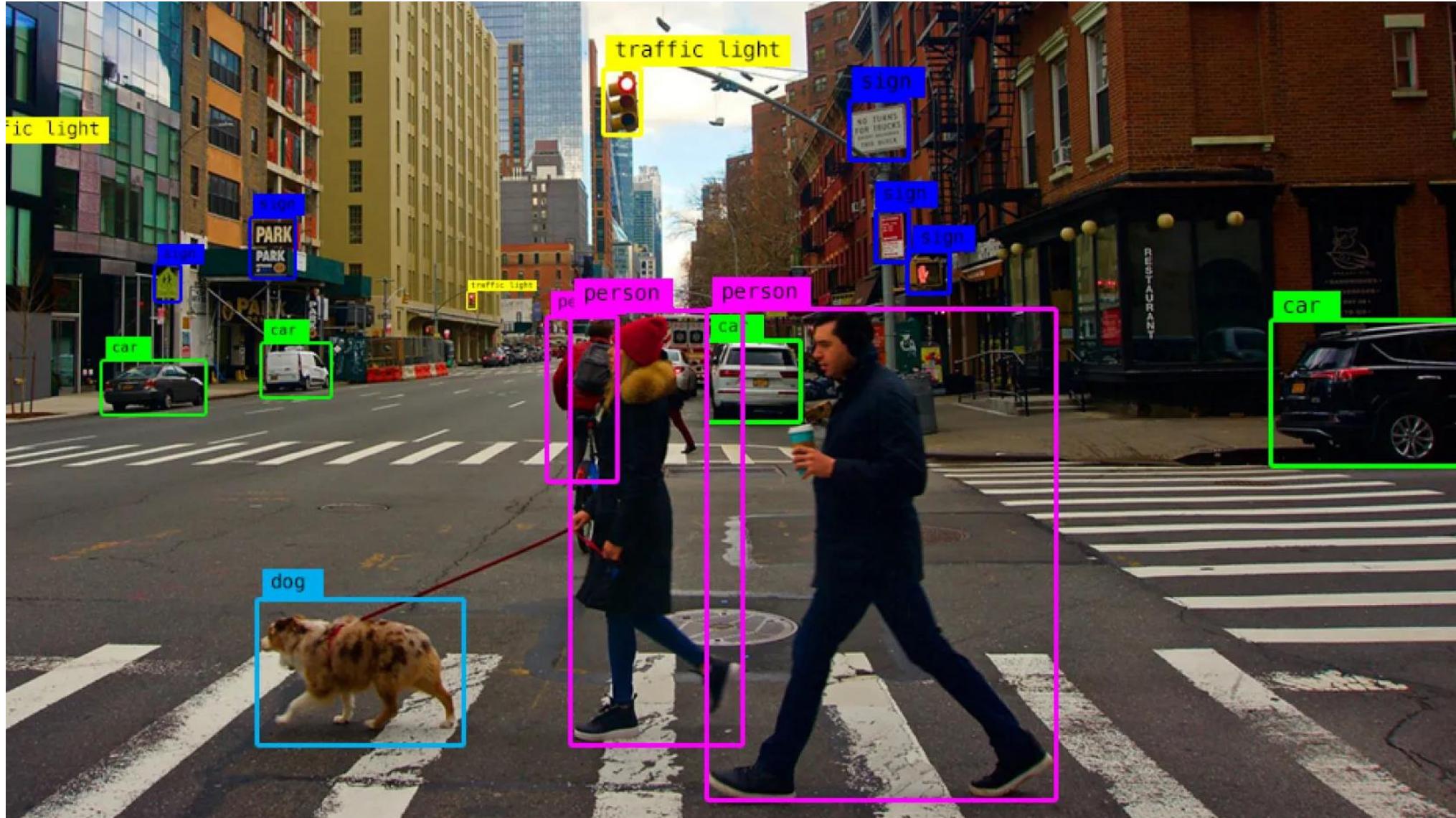


Add a Caption

 Look Up Dog >

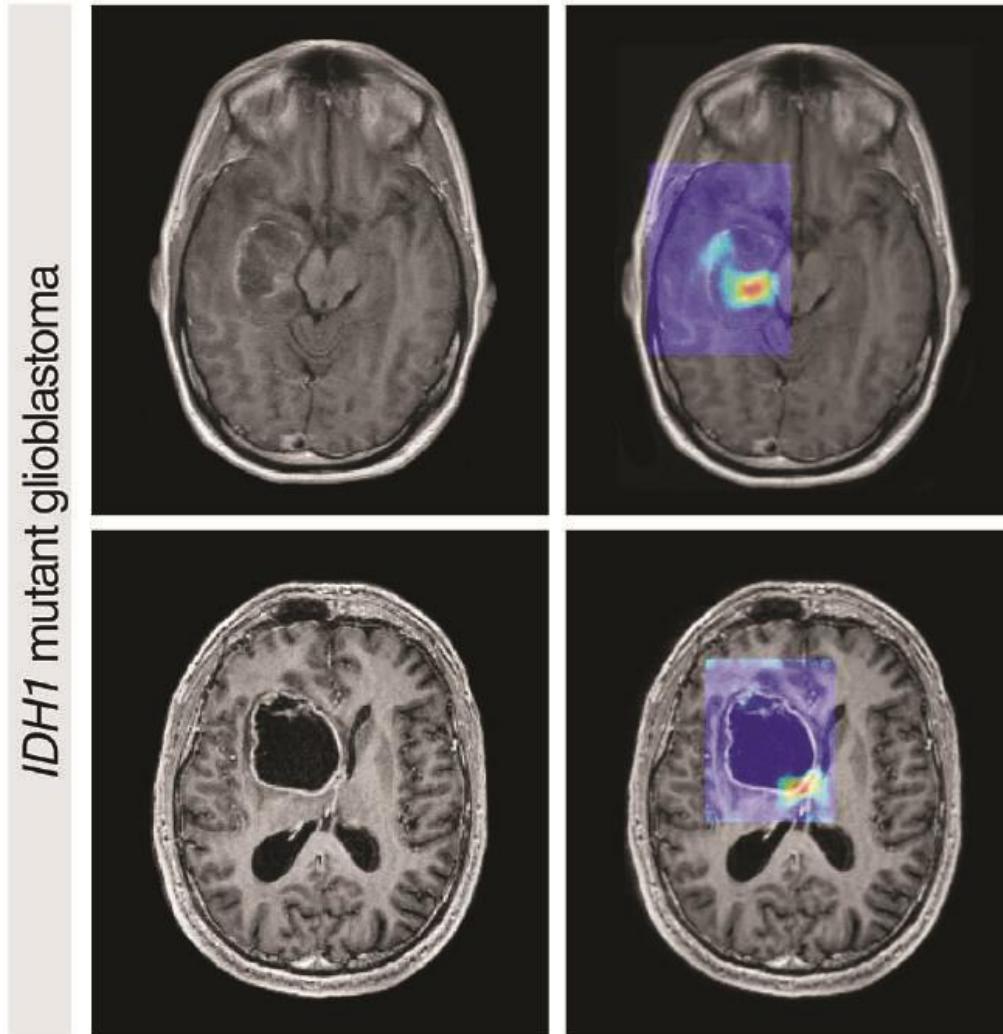
Aplicações do aprendizado de máquinas

Detecção, classificação e localização de objetos



Aplicações do aprendizado de máquinas

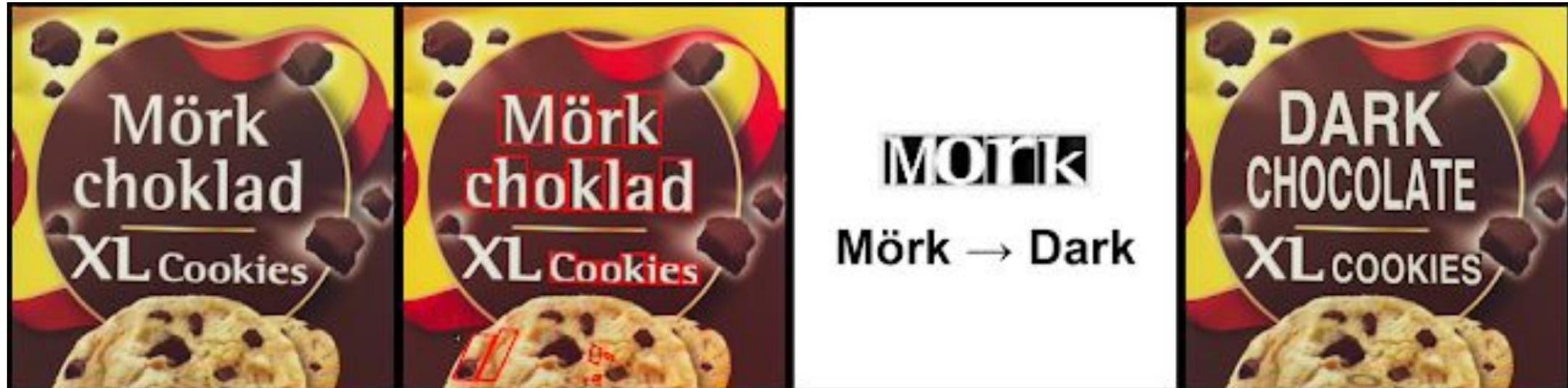
Auxílio ao Diagnóstico



Referência: <https://doi.org/10.3322/caac.21552>

Aplicações do aprendizado de máquinas

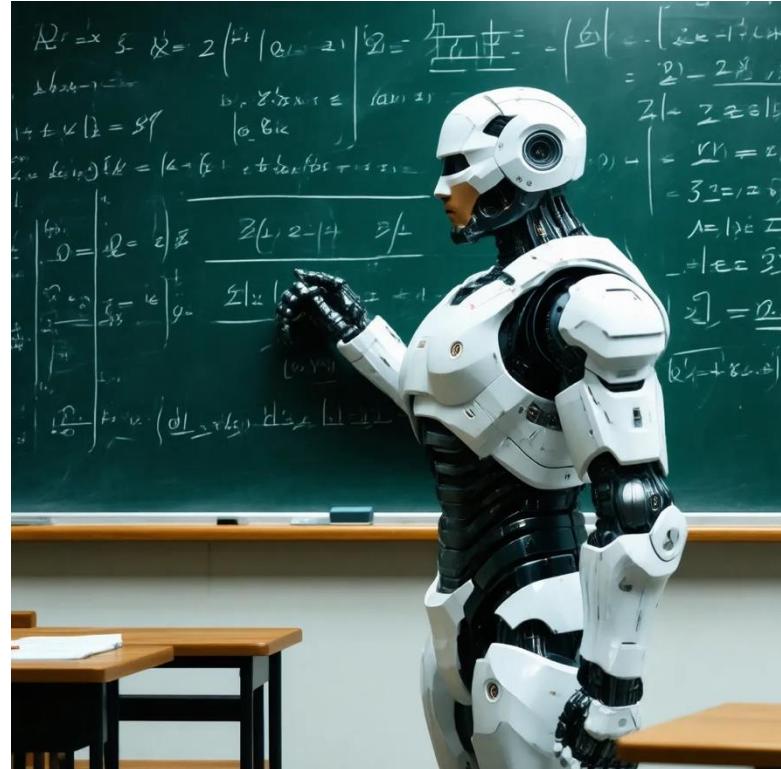
Tradução automática



Aplicações do aprendizado de máquinas

Geração de imagens

- Variational Autoencoders, Generative Adversarial Networks, Diffusion Models ...



Entrada: Robocop teaching a calculus class on a blackboard realistic 4k



Entrada: Pikachu Riding a motorcycle cartoon

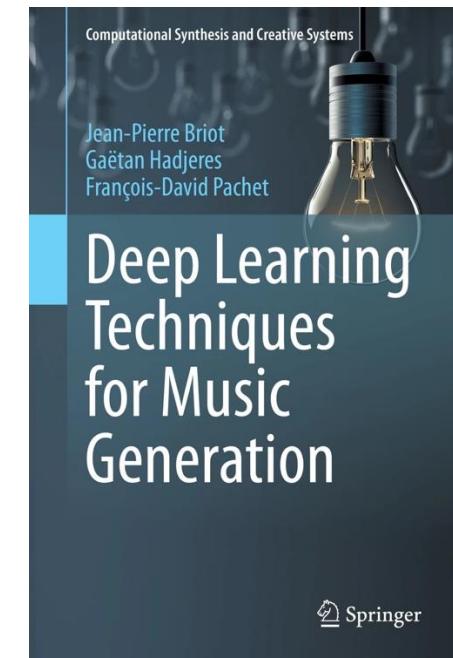
Aplicações do aprendizado de máquinas

Criação de músicas

- Redes Neurais Recorrentes, LSTM



<https://www.youtube.com/watch?v=Emidxpkyk6o>



Aplicações do aprendizado de máquinas

Large Language Models

- ChatGPT (OpenAI), Gemini (Google), Llama (Meta), Claude (Anthropic),...



You

could you solve step by step the integral of $x \sin(x) dx$?



ChatGPT

Certainly! To solve the integral $\int x \sin(x) dx$, we'll use integration by parts. The integration by parts formula is given by:

$$\int u dv = uv - \int v du$$

Let's choose u and dv :

$$u = x \quad \text{and} \quad dv = \sin(x) dx$$

Now, we'll differentiate u to find du and integrate dv to find v :

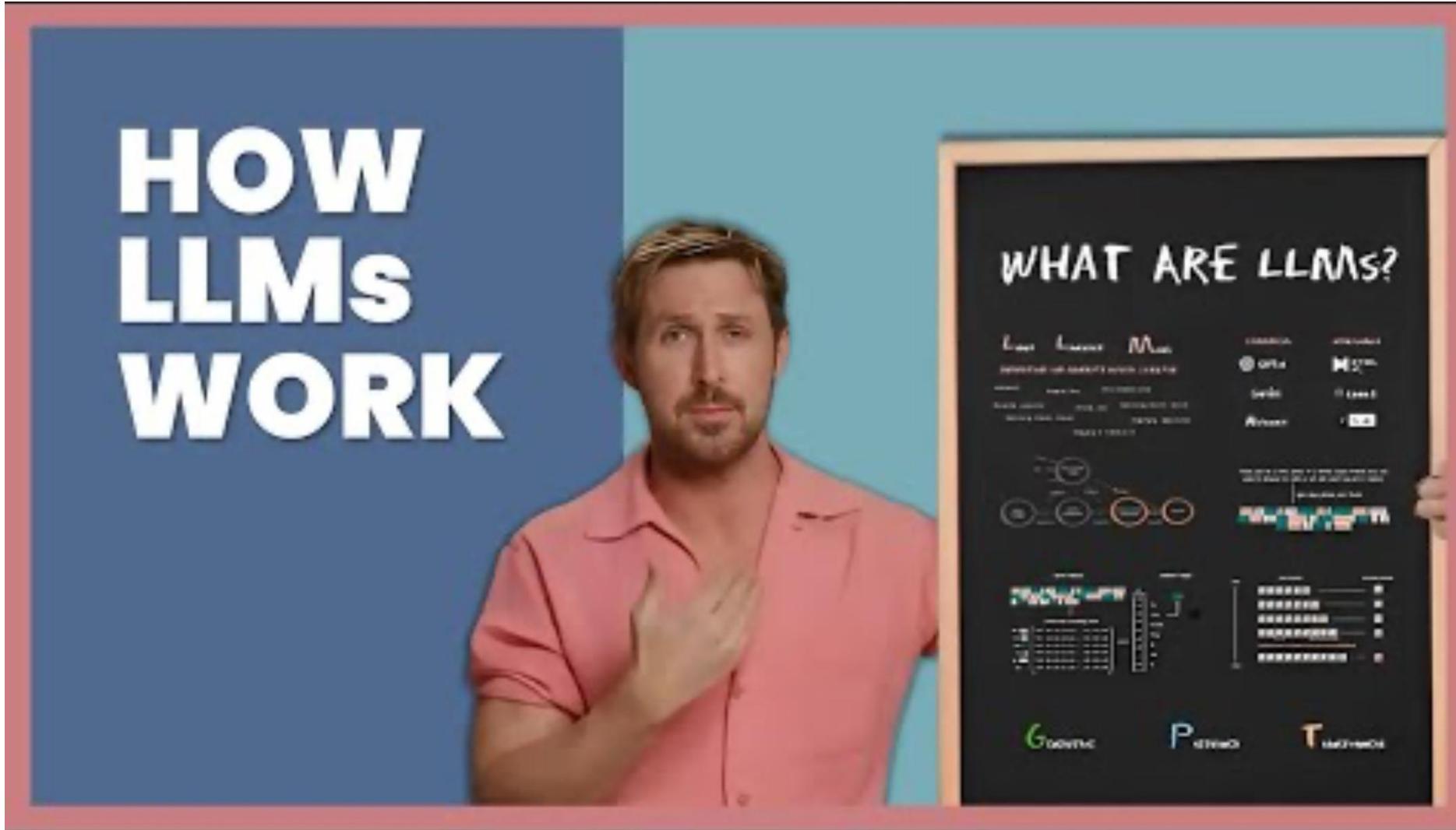
$$du = dx \quad \text{and} \quad v = -\cos(x)$$

Now, we can apply the integration by parts formula:

$$\int x \sin(x) dx = uv - \int v du$$

$$= x(-\cos(x)) - \int (-\cos(x)) dx$$

Deepfakes



Aplicações do aprendizado de máquinas

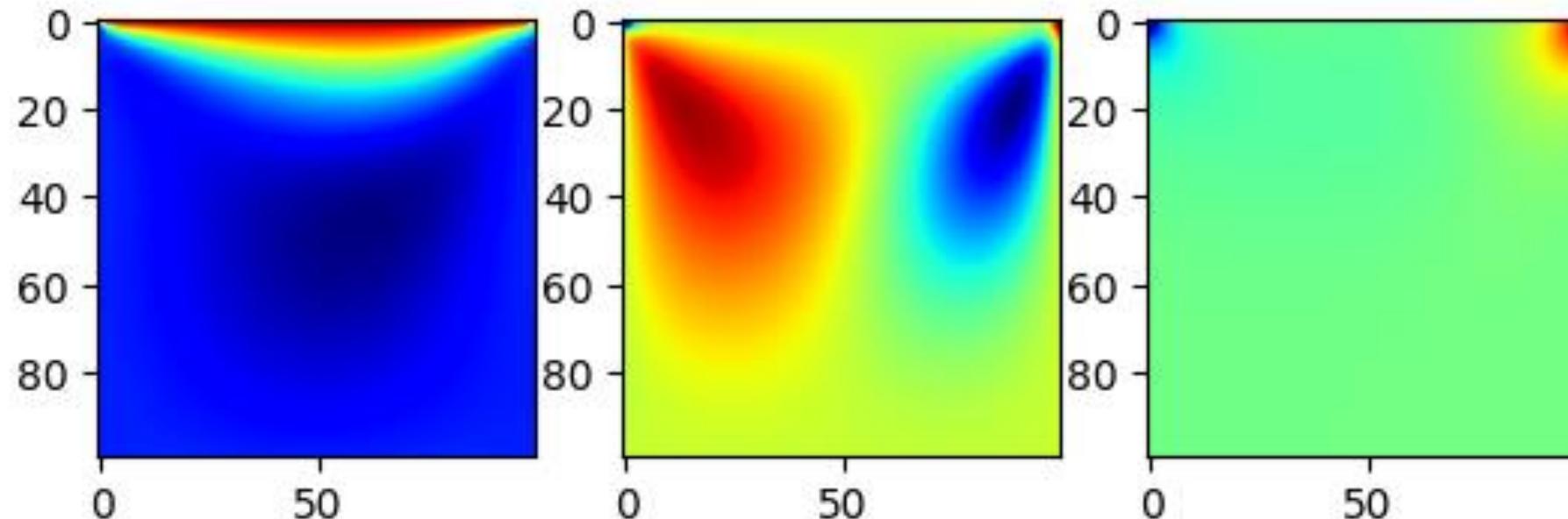
Solução de Equações Diferenciais (PINNs)

Problema da cavidade quadrada com a tampa deslizante

$$\text{Navier-Stokes: } \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \frac{1}{\text{Re}} \Delta \mathbf{u}$$

$$\text{Continuidade: } \nabla \cdot \mathbf{u} = 0$$

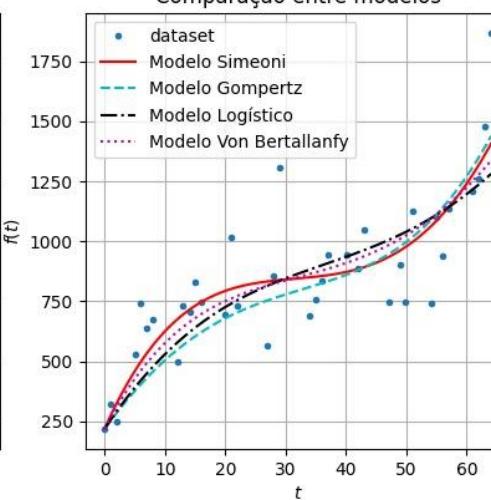
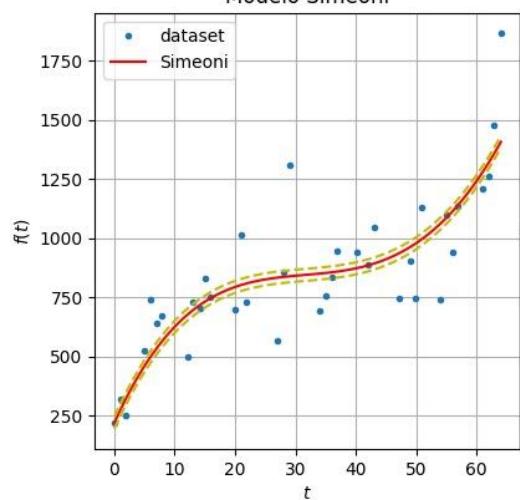
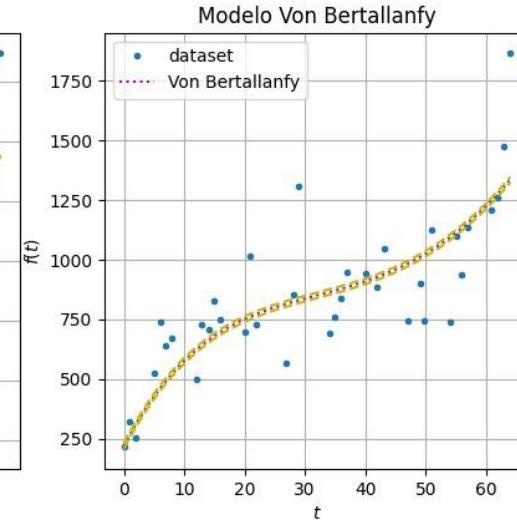
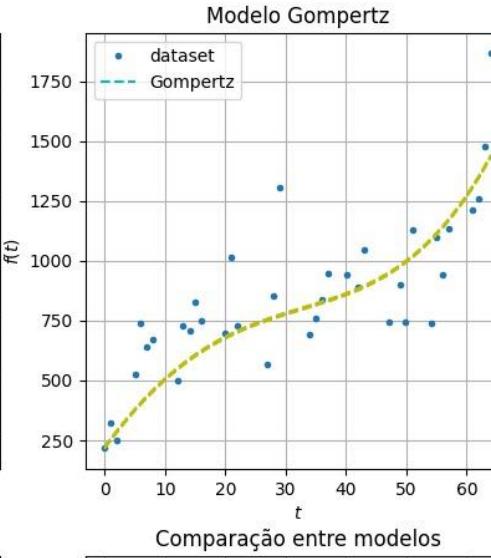
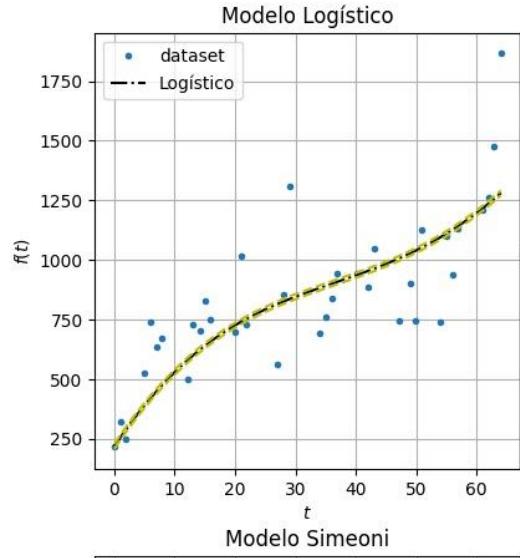
$$\text{Condições de contorno: } \mathbf{u}_{\perp} = 0 \text{ e } \mathbf{u}_{\rightarrow} = 1$$



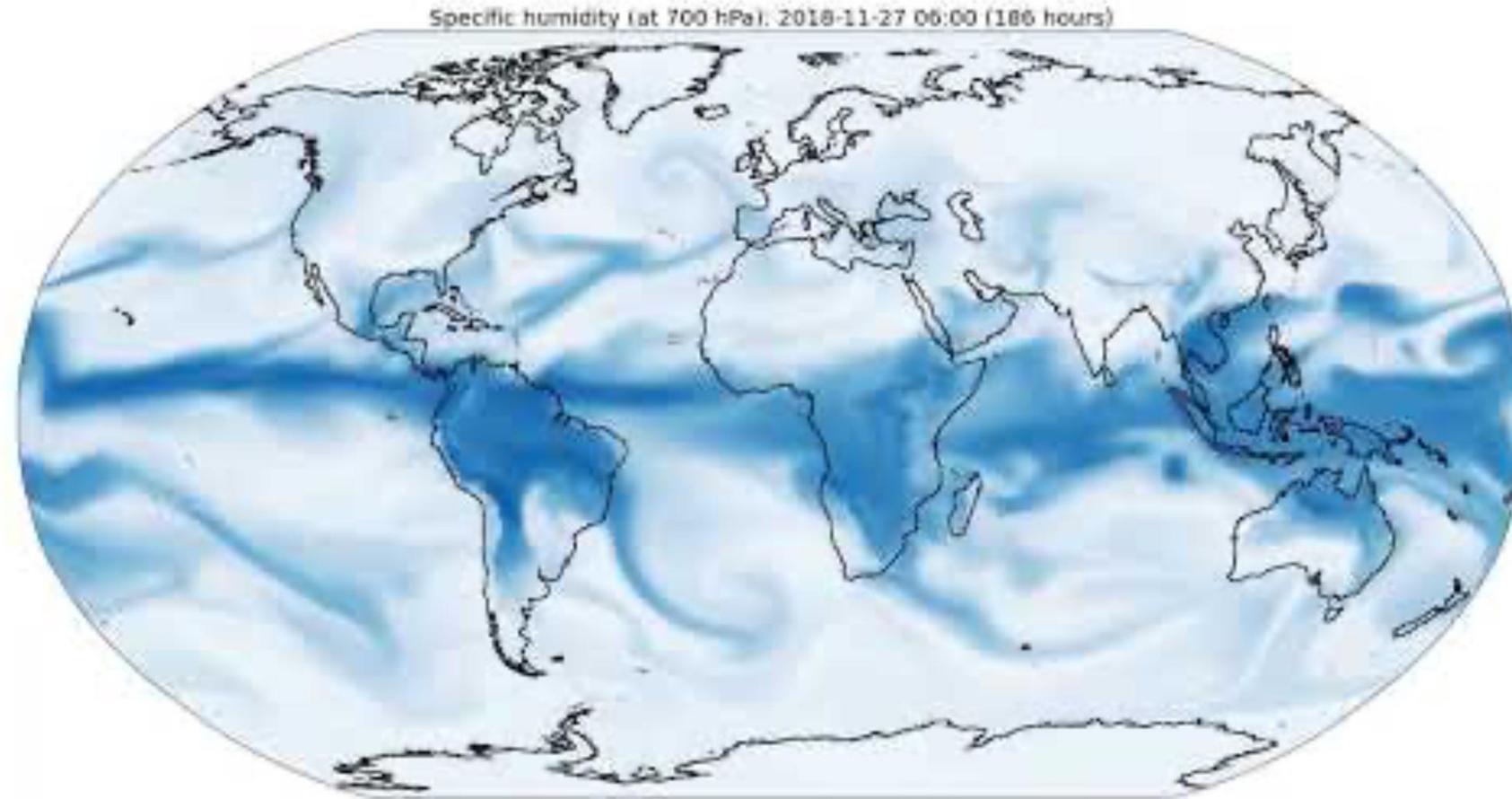
Aplicações das Redes Neurais

Modelagem do crescimento de tumores malignos

- Estimação de Parâmetros + PINNs



Meteorologia

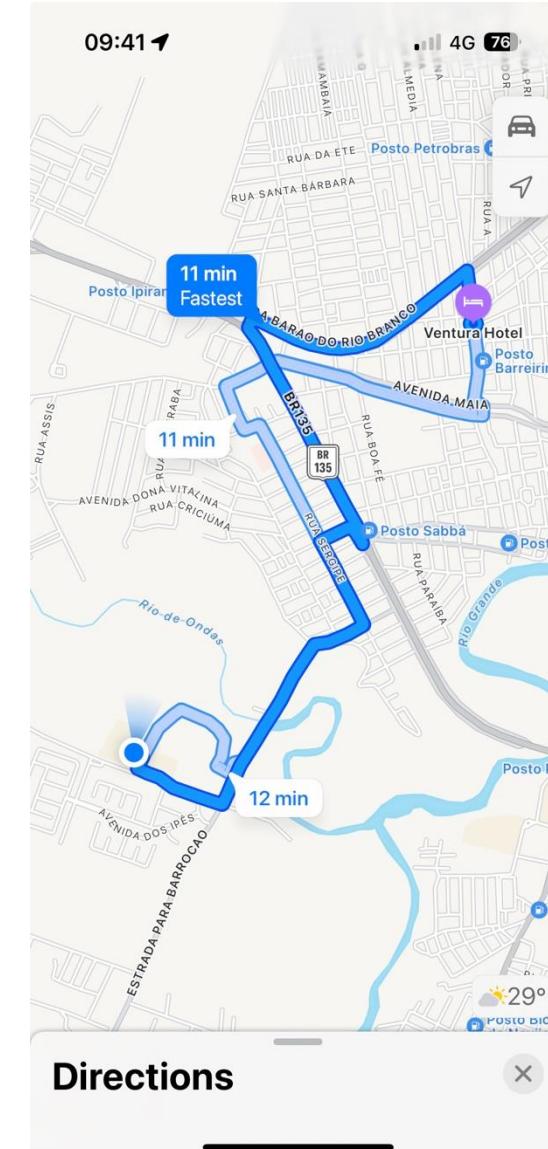


<https://deepmind.google/discover/blog/graphcast-ai-model-for-faster-and-more-accurate-global-weather-forecasting/>

<https://www.nature.com/articles/d41586-023-03552-y>

A lista de aplicações é enorme:

- Prevenção de fraudes no *e-commerce*
- Chatbots, Siri, Alexa, ChatGPT, Gemini...
- Veículos autônomos
- Filtros de *spam*
- Aplicações financeiras (LSTM e GRU)
- Gerenciamento de tráfego urbano
- Recomendações nas redes sociais
- Atacar problemas matemáticos complexos*
- ...



*<https://www.quantamagazine.org/ai-reveals-new-possibilities-in-matrix-multiplication-20221123/>

Parte 2

De onde vem as redes neurais?

Cérebro: estimado em 85 bilhões de neurônios

O movimento de um braço pode induzir o disparo de cerca de 50 milhões desses neurônios

Os modelos de redes neurais profundas não são cérebros nem tentativas de criar cérebros artificiais

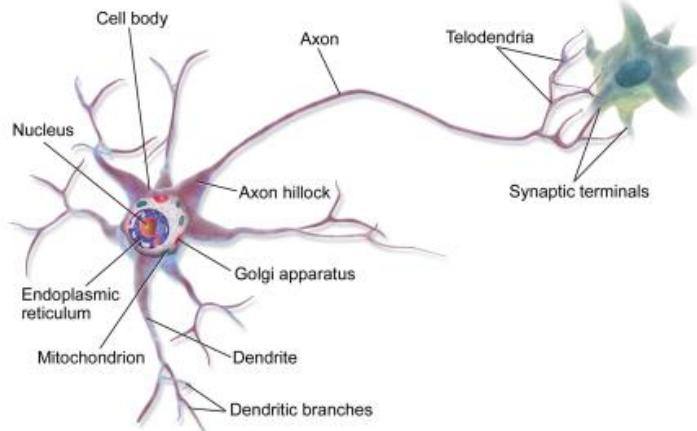
Modelos são motivados pela estrutura biológica do cérebro.

O bloco de construção básico de um modelo de rede neural profunda é o neurônio (perceptron) que abstrai a conexão sináptica entre os neurônios por meio de um único número chamado valor de ativação.

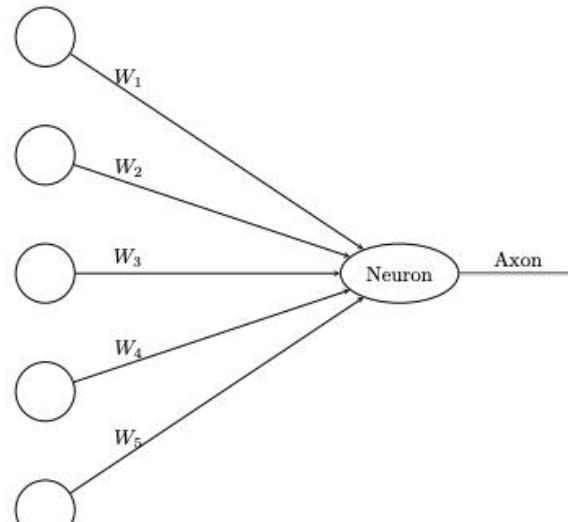
O perceptron foi inventado em 1943 por Warren McCulloch e Walter Pitts.

A primeira implementação foi uma máquina construída em 1958 no Laboratório Aeronáutico Cornell por Frank Rosenblatt, financiada pelo Escritório de Pesquisa Naval dos Estados Unidos.

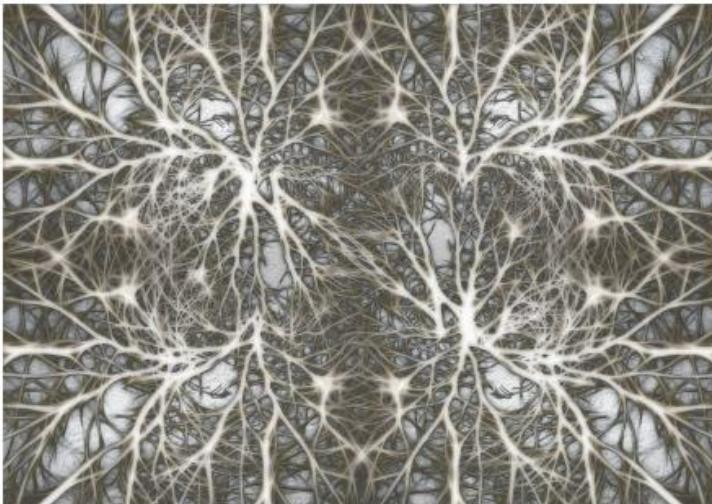
De onde vem as redes neurais?



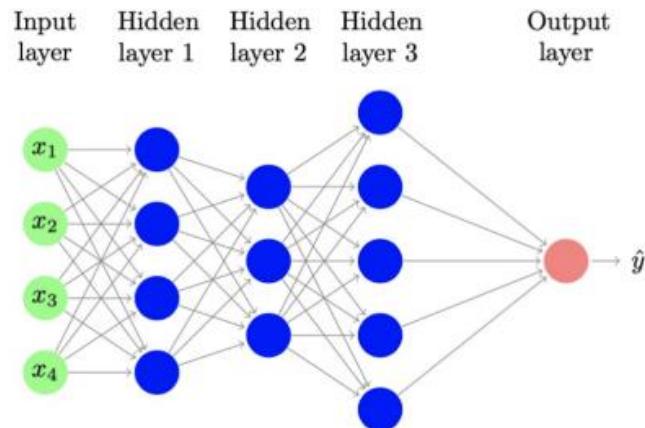
(a)



(b)



(c)

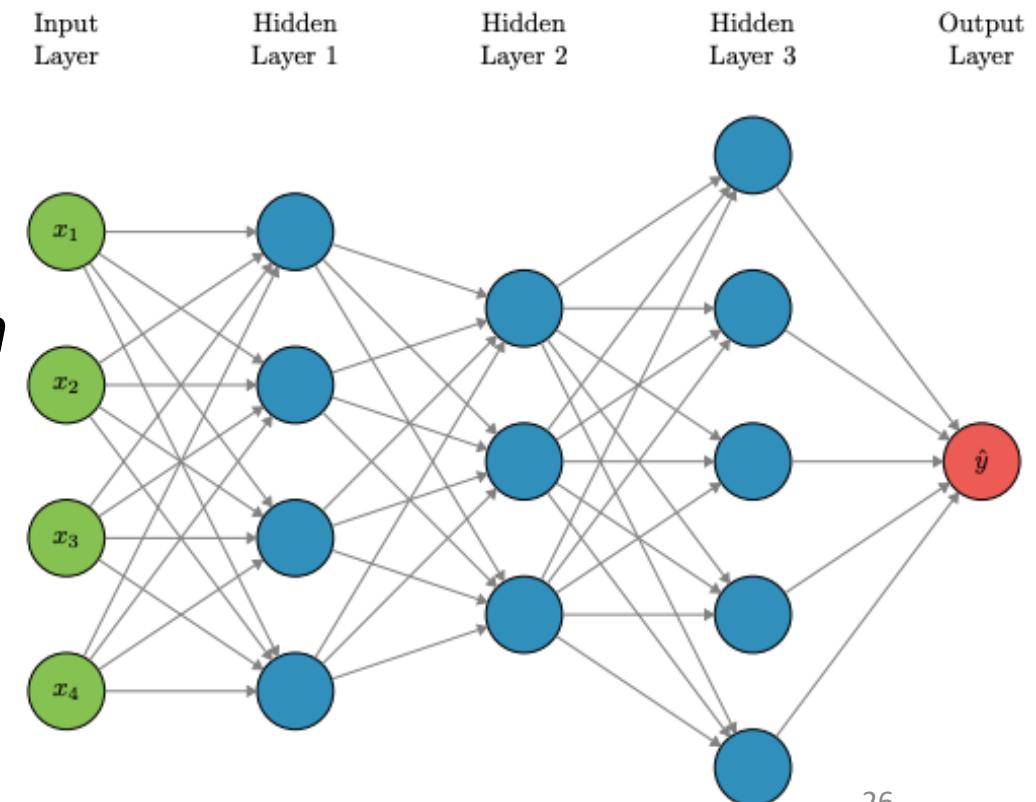


(d)

Estrutura baseada no neurônio artificial (perceptron)

Perceptron: função álgebraica comum com entrada e saída

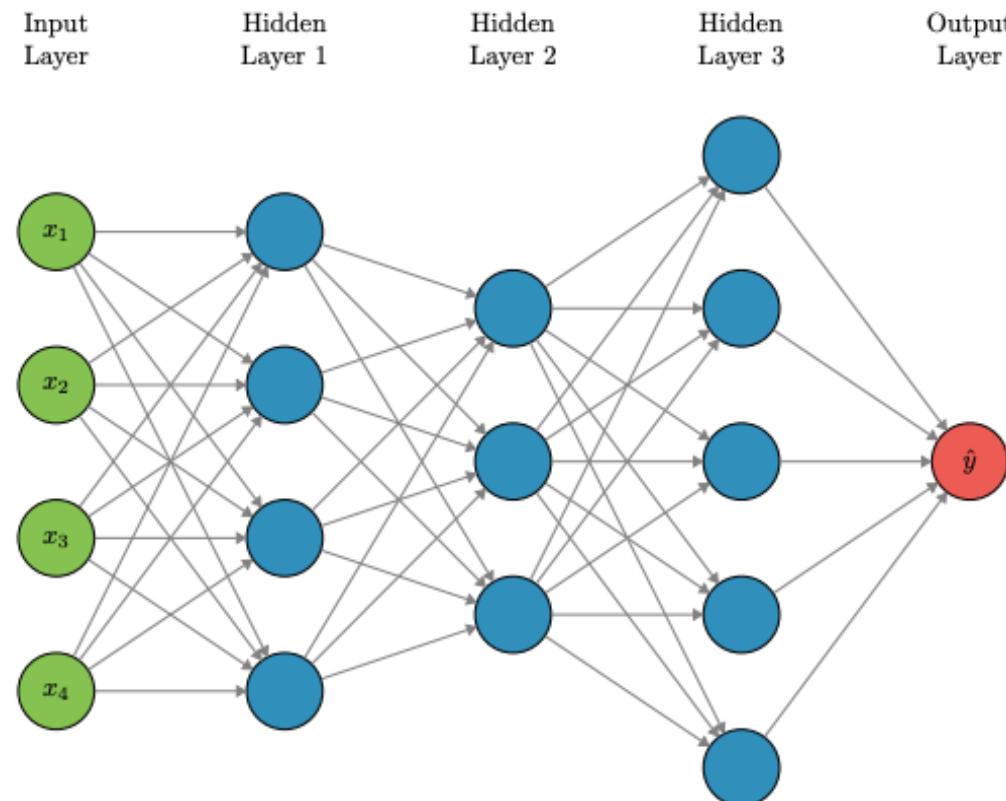
Esses neurônios artificiais são dispostos ao longo de várias camadas que interagem entre si criando a estrutura conhecida como rede neural ou *multilayer perceptron*



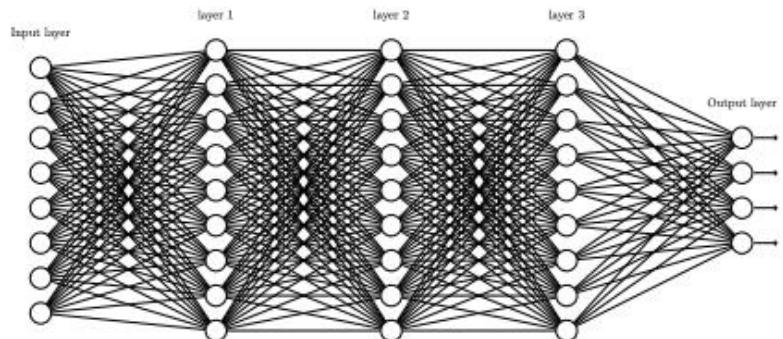
De onde vem as redes neurais?

Estrutura básica em camadas:

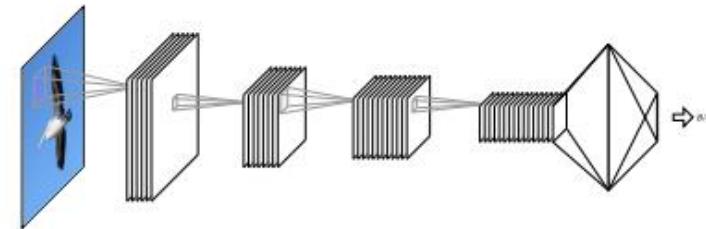
- uma camada de entrada
- camada(s) intermediária(s) (escondida(s))
- camada de saída



Diferentes Arquiteturas

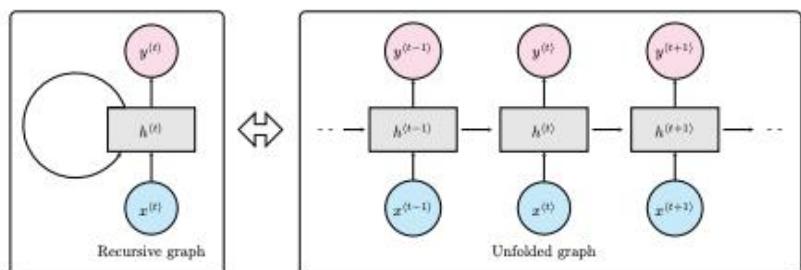


(a)



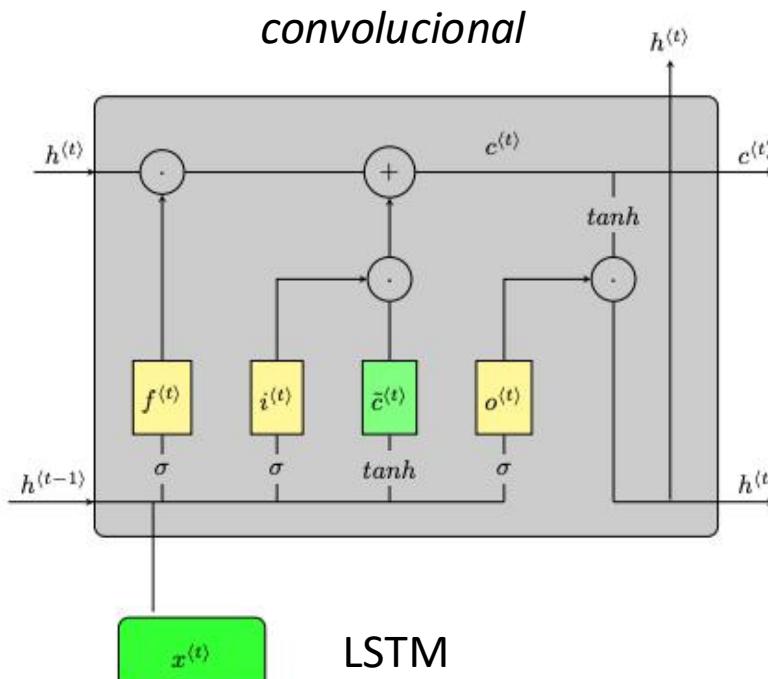
(b)

feedforward



(c)

recorrente



(d)

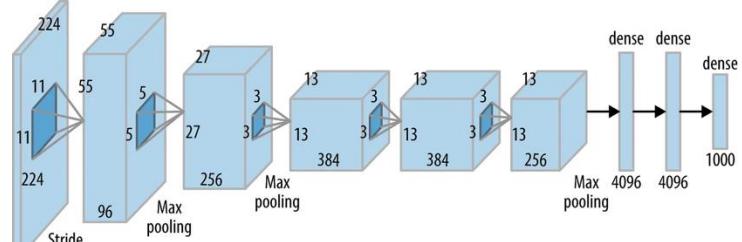
DNN: compressão de dados, reconhecimento de padrões, visão computacional, reconhecimento de alvos de sonar, reconhecimento de fala, reconhecimento de caracteres manuscritos.

CNN: análise de imagens, processamento de linguagem natural e outros problemas complexos de classificação de imagens.

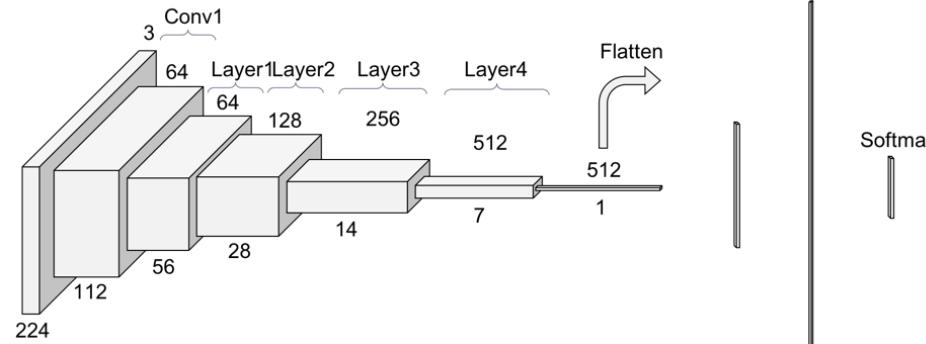
Recorrente: reconhecimento de fala e processamento de linguagem natural

LSTM: usada para dados sequenciais como análise de sentimento, modelagem de linguagem, reconhecimento de fala e análise de vídeo.

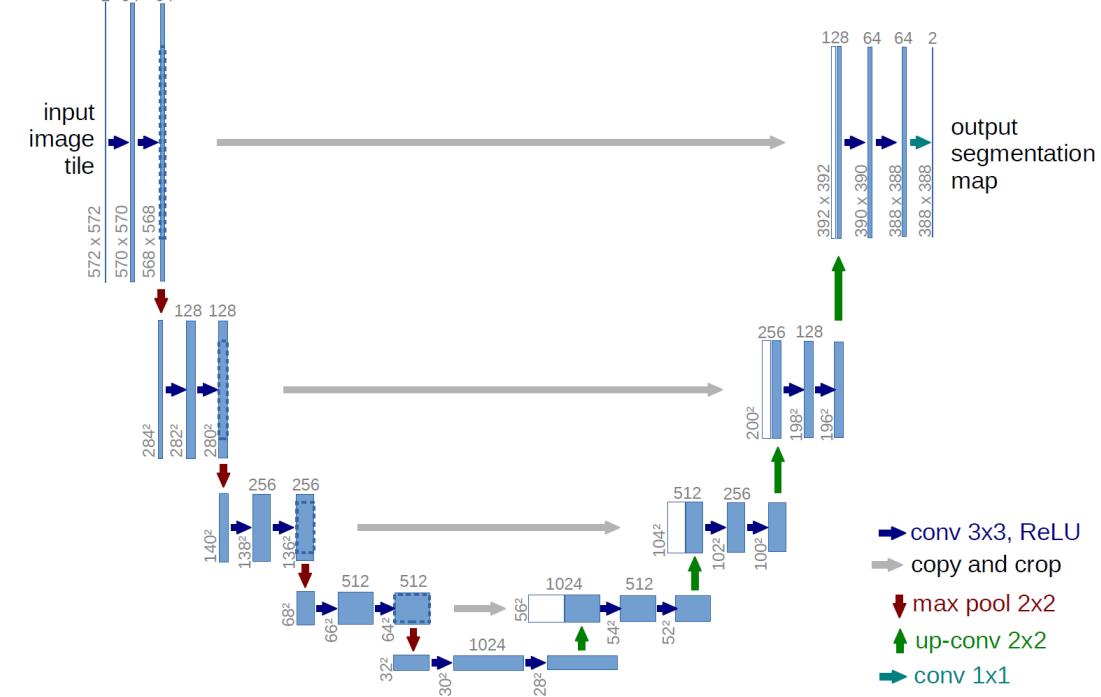
Ex: Diferentes Arquiteturas nas CNN



AlexNET



ResNET

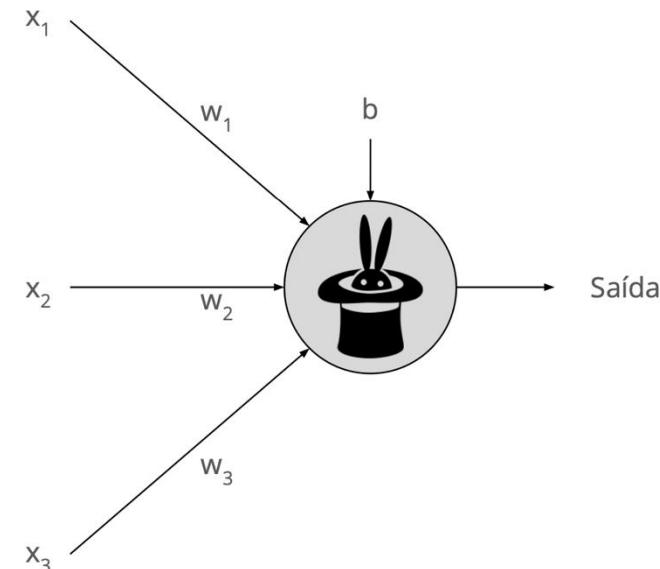


U-NET

Entrada: linha da base de dados ou saída dos neurônios da camada anterior (x_i)

Saída: uma combinação das entradas

- Cada ligação tem um peso (w_i)
- Cada neurônio tem um viés (b)
- A saída do neurônio é igual a:
$$\hat{y} = \sigma(\sum w_i x_i + b) = \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3 + b)$$



Um neurônio faz uma transformação afim dos dados de entrada e em seguida aplica uma função de ativação

Funções de Ativação, por quê?

Vamos entender aqui porquê a necessidade das funções de ativação. Considere:

$$n_1 = w_{11}I_1 + w_{12}I_2 + b_1$$

$$n_2 = w_{21}I_1 + w_{22}I_2 + b_2$$

$$n_3 = w_{31}n_1 + w_{32}n_2 + b_3$$

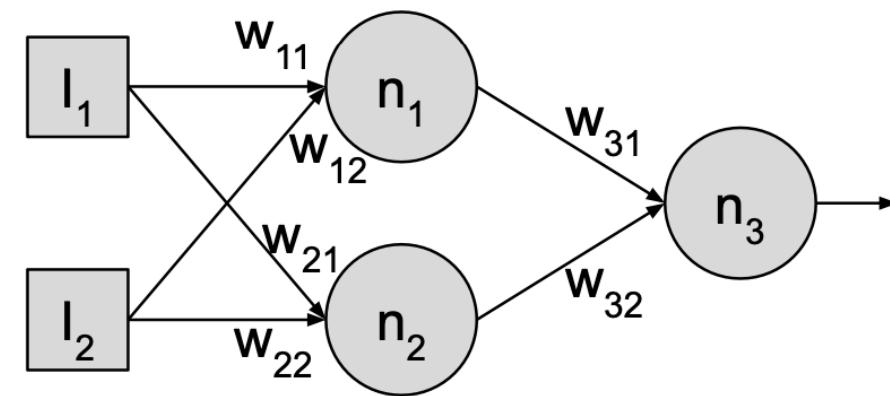
Substituindo n_1 e n_2 em n_3 , encontramos:

$$n_3 = w_{31}n_1 + w_{32}n_2 + b_3$$

$$= w_{31}(w_{11}I_1 + w_{12}I_2 + b_1) + w_{32}(w_{21}I_1 + w_{22}I_2 + b_2) + b_3$$

$$= (w_{31}w_{11} + w_{32}w_{21})I_1 + (w_{31}w_{12} + w_{32}w_{22})I_2 + (w_{31}b_1 + w_{32}b_2 + b_3)$$

$$= W_1I_1 + W_2I_2 + B_3$$

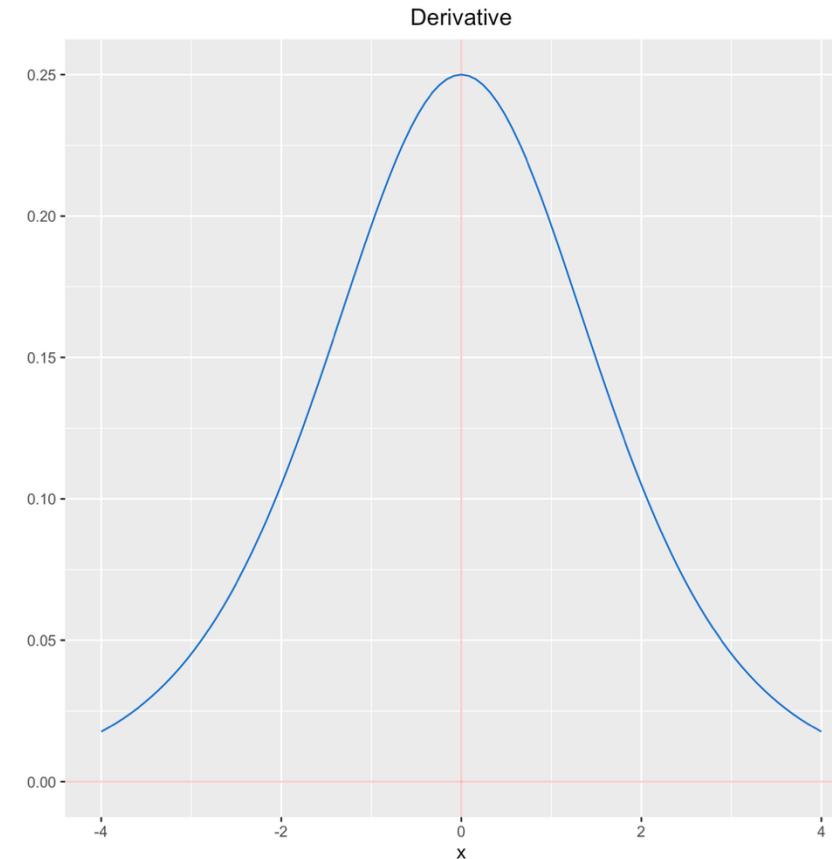
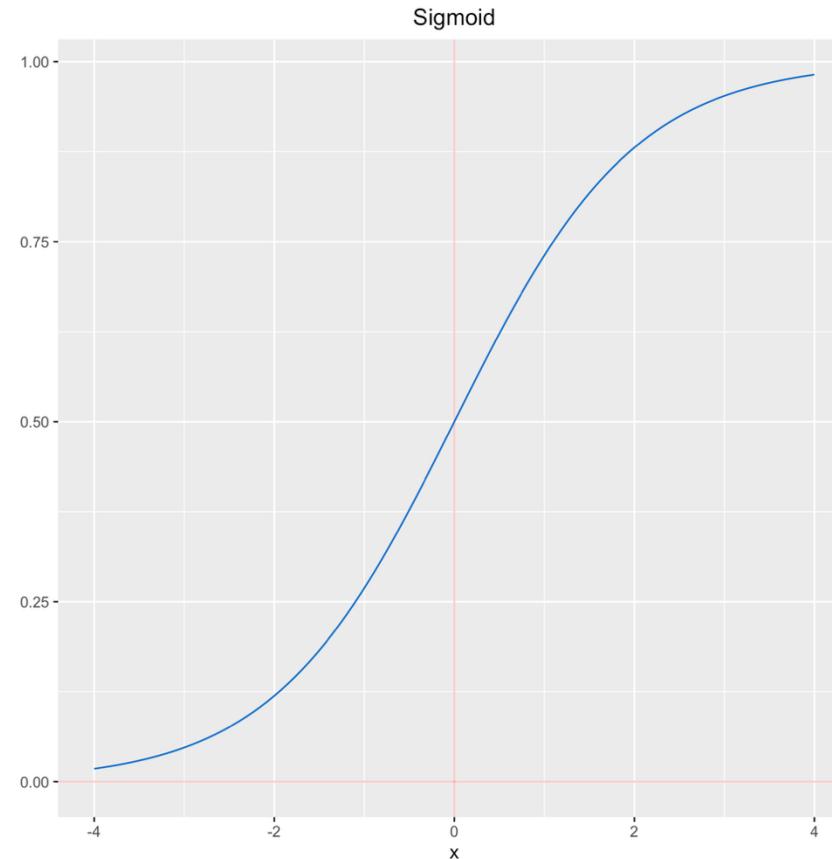


Sem as funções de ativação não teríamos a necessidade das múltiplas camadas!
 As funções de ativação inserem não-lineadades na rede neural.

Funções de Ativação: Sigmóide

A sigmóide é a solução da equação logística: $\frac{d\sigma}{dz} = \sigma(1 - \sigma)$

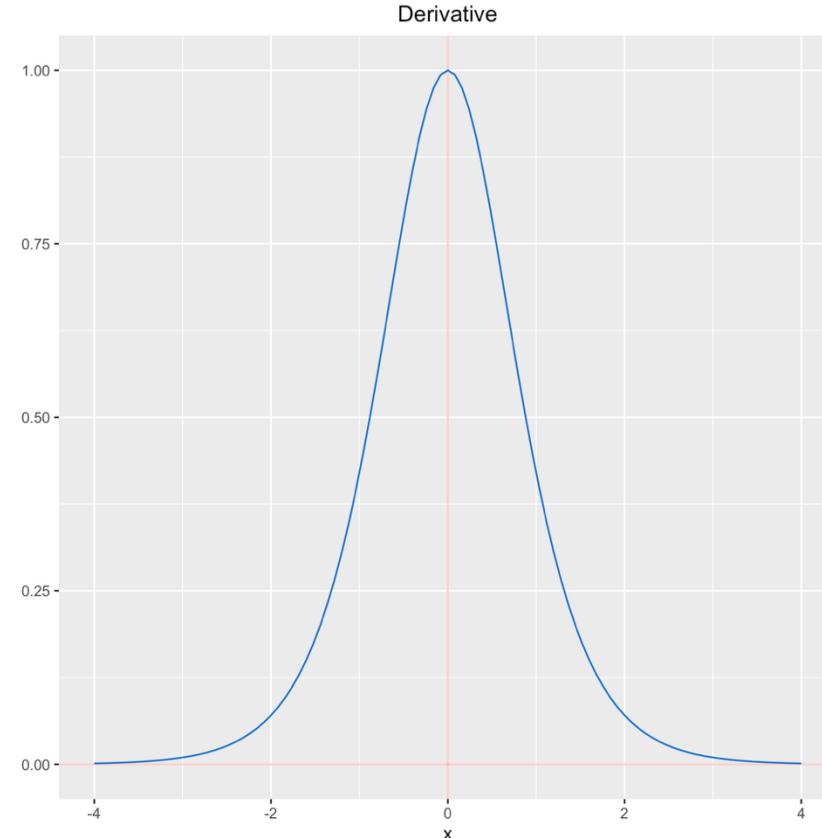
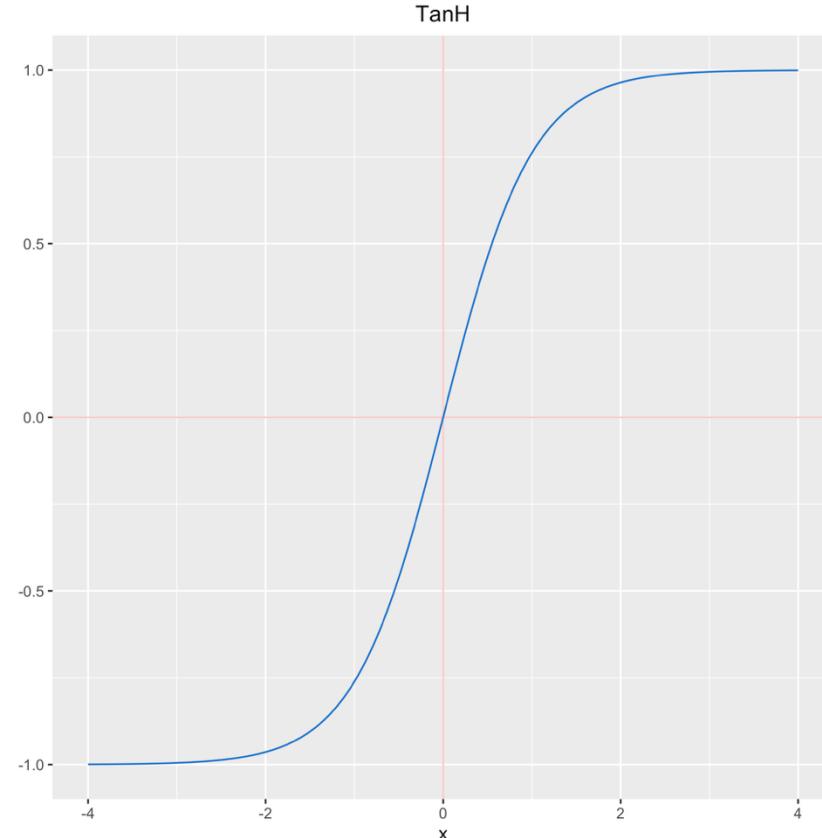
Isto é: $\sigma(z) = \frac{1}{1+e^{-z}} \in [0,1]$



Funções de Ativação: Tangente Hiperbólica

A tangente hiperbólica $\tgh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \in [-1, 1]$

A derivada da $\tgh(z)$ satisfaz a equação: $\frac{d\tgh}{dz} = 1 - \tgh^2(z)$

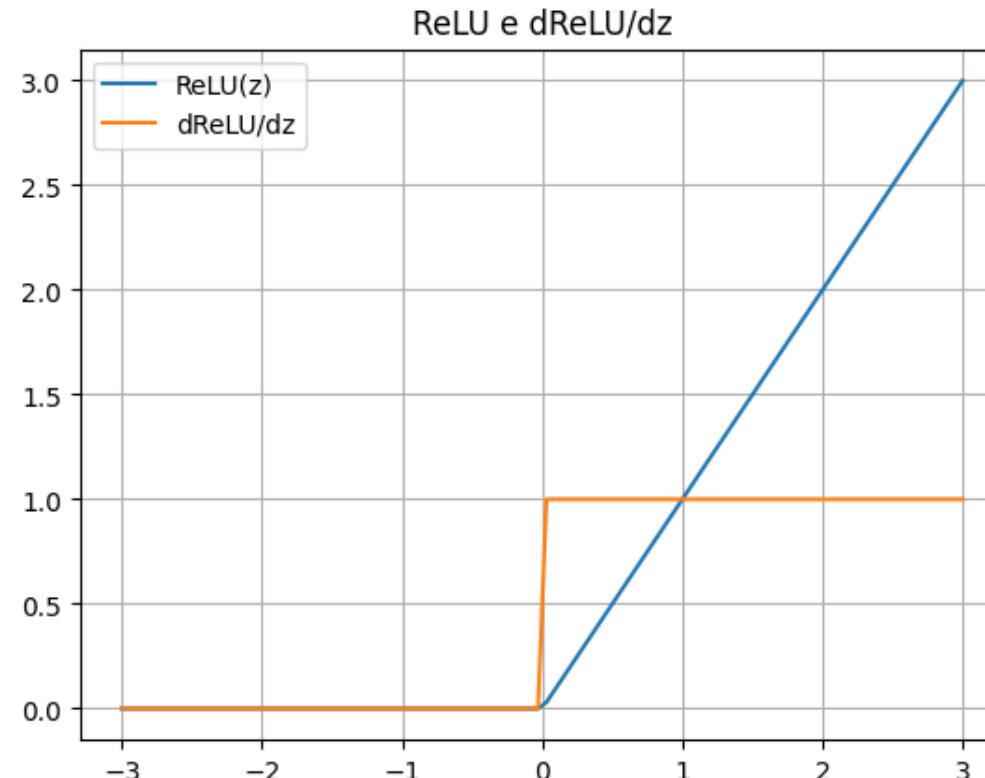


Funções de Ativação: ReLU

ReLU significa *Rectified Linear Unity* e é definida por:

$$\text{ReLU}(z) = \max(0, z)$$

Sua derivada é dada por: $\frac{d}{dz} \text{ReLU}(z) = \begin{cases} 1, & \text{se } x > 0 \\ 0, & \text{se } x \leq 0 \end{cases}$

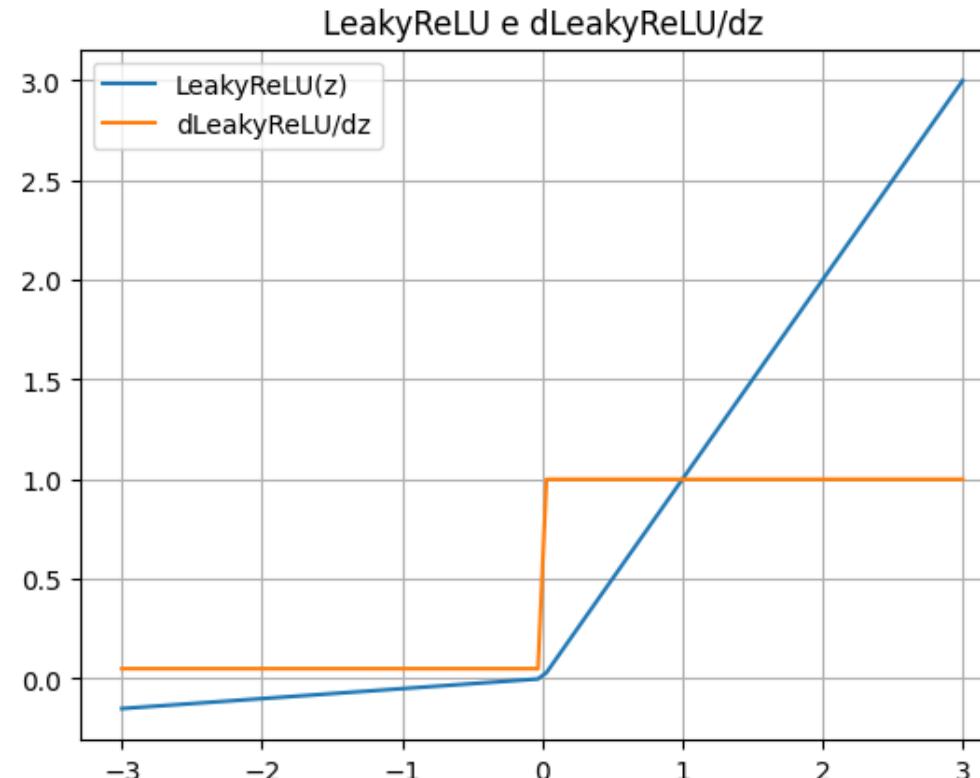


Funções de Ativação: LeakyReLU

LeakyReLU é definida para $0 < \alpha \ll 1$ como:

$$\text{LeakyReLU}(z) = \max(\alpha z, z)$$

Sua derivada é dada por: $\frac{d}{dz} \text{LeakyReLU}(z) = \begin{cases} 1, & \text{se } x > 0 \\ \alpha, & \text{se } x \leq 0 \end{cases}$



A função **softmax** é utilizada em problemas de classificação em que se tem mais de uma classe, pois ela atribui probabilidades decimais a cada classe do problema.

Dado $\mathbf{z} \in \mathbb{R}^K$ a função de ativação **softmax** é uma função $\mathbb{R}^K \rightarrow (0,1)^K$ definida como

$$\text{softmax}(\mathbf{z}) = \frac{1}{\sum_{i=1}^K e^{z_i}} [e^{z_1} \ e^{z_2} \ \dots \ e^{z_K}]^T$$

Note que $\|\text{softmax}(\mathbf{z})\| < 1$, para todo \mathbf{z} .

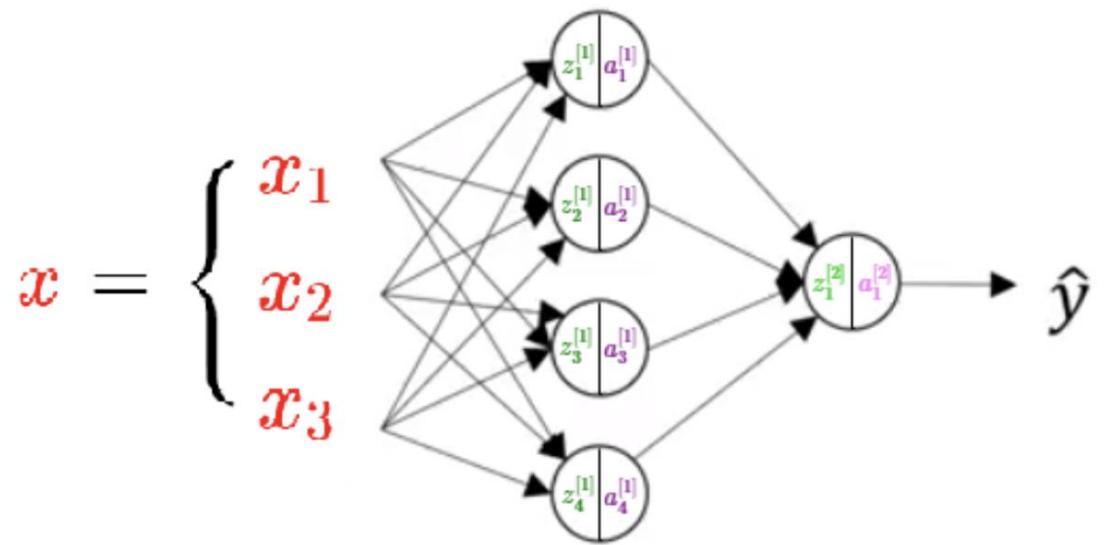
Mas, o que faz uma rede neural afinal?

Já vimos um pouco, e sem querer, sobre a estrutura básica das redes neurais quando falamos sobre os perceptrons e as funções de ativação.

O objetivo de uma rede *feedforward* é aproximar alguma função $g: \mathbb{R}^p \rightarrow \mathbb{R}^q$. Um modelo de rede *feedforward* define uma aplicação $f_\theta: \mathbb{R}^p \rightarrow \mathbb{R}^q$ e aprende o valor dos parâmetros θ que idealmente resultam em $g(x) \approx f_\theta(x)$.

Então vamos construir uma rede neural e entender como ela efetivamente aprende.

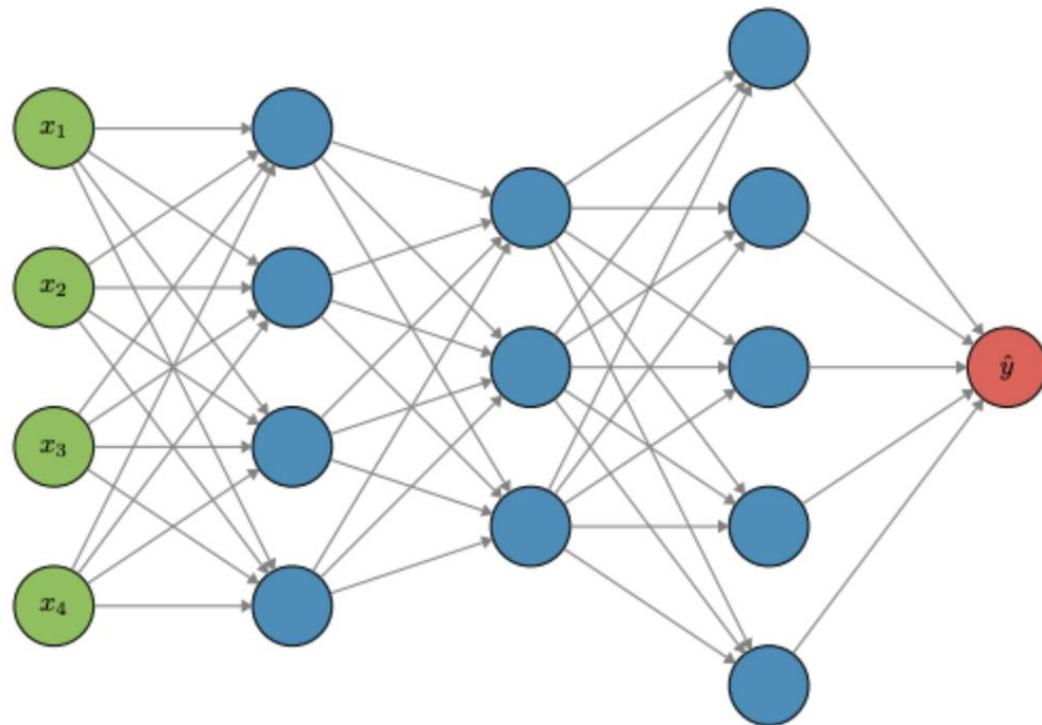
Rede Neural Feedforward



$$\begin{aligned}
 \mathbf{x} = \left\{ \begin{array}{l} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{array} \right\} & \quad z^{[1]} = W^{[1]}\mathbf{x} + b^{[1]} \quad a^{[1]} = \sigma(z^{[1]}) \\
 \left\{ \begin{array}{l} z_1^{[1]} = w_1^{[1]T}\mathbf{x} + b_1^{[1]} \\ z_2^{[1]} = w_2^{[1]T}\mathbf{x} + b_2^{[1]} \\ z_3^{[1]} = w_3^{[1]T}\mathbf{x} + b_3^{[1]} \\ z_4^{[1]} = w_4^{[1]T}\mathbf{x} + b_4^{[1]} \end{array} \right. & \Rightarrow \quad \left\{ \begin{array}{l} a_1^{[1]} = \sigma(z_1^{[1]}) \\ a_2^{[1]} = \sigma(z_2^{[1]}) \\ a_3^{[1]} = \sigma(z_3^{[1]}) \\ a_4^{[1]} = \sigma(z_4^{[1]}) \end{array} \right. \Rightarrow \quad z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\
 & \quad W^{[2]} = \begin{bmatrix} w_{1,1}^{[2]} \\ w_{1,2}^{[2]} \\ w_{1,3}^{[2]} \\ w_{1,4}^{[2]} \end{bmatrix} \quad b^{[2]} = \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \\ b_3^{[2]} \\ b_4^{[2]} \end{bmatrix} \Rightarrow \quad a^{[2]} = \sigma(z^{[2]}) \longrightarrow \hat{y}
 \end{aligned}$$

Rede Neural Feedforward

Input Layer Hidden Layer 1 Hidden Layer 2 Hidden Layer 3 Output Layer



$$f_{\theta}(x) = S^{[4]}(W^{[4]}S^{[3]}(W^{[3]}S^{[2]}(W^{[2]}S^{[1]}(W^{[1]}x + b^{[1]}) + b^{[2]}) + b^{[3]}) + b^{[4]}$$

$$\underbrace{4 \times 4 + 4}_{\text{Hidden layer 1}} + \underbrace{3 \times 4 + 3}_{\text{Hidden layer 2}} + \underbrace{5 \times 3 + 5}_{\text{Hidden layer 3}} + \underbrace{1 \times 5 + 1}_{\text{Output layer}} = 61.$$

Para um conjunto de pesos e vieses, representado por θ , precisamos encontrá-los de forma que a rede neural se aproxime efetivamente de, isto é, $g(\mathbf{x}) \approx f_\theta(\mathbf{x})$.

Para fazer essa procura dos parâmetros θ precisamos definir uma forma de mensurar o erro entre $g(\mathbf{x})$ e $f_\theta(\mathbf{x})$ para que seja possível encontrar o menor erro possível. Há diferentes maneiras de se definir uma medida de erro. Uma delas é o erro médio quadrático (MSE):

$$J(\theta) = \frac{1}{2} \|g(\mathbf{x}) - f_\theta(\mathbf{x})\|^2$$

Vantagem: derivável e fácil de tratar

Desvantagem: muito sensível aos outliers.

Em problemas de classificação, por exemplo, quando se deseja verificar se o objeto contido em uma imagem tem um determinado rótulo a rede neural irá calcular a probabilidade do objeto ter aquela rotulação. Nestas situações utiliza-se outro tipo de medida de erro, que neste caso é chamada de *cross-entropy*

$$K(\theta) = - \sum_i p_i \ln q_i$$

para $f_\theta(x) = [q_1 \ q_2 \ q_3 \dots q_n]$, aqui a verdadeira probabilidade p_i é o verdadeiro rótulo, e a distribuição dada q_i é o valor previsto pelo modelo atual.

As medidas de erro são chamadas **funções de perda (loss functions)**.

Parte 3

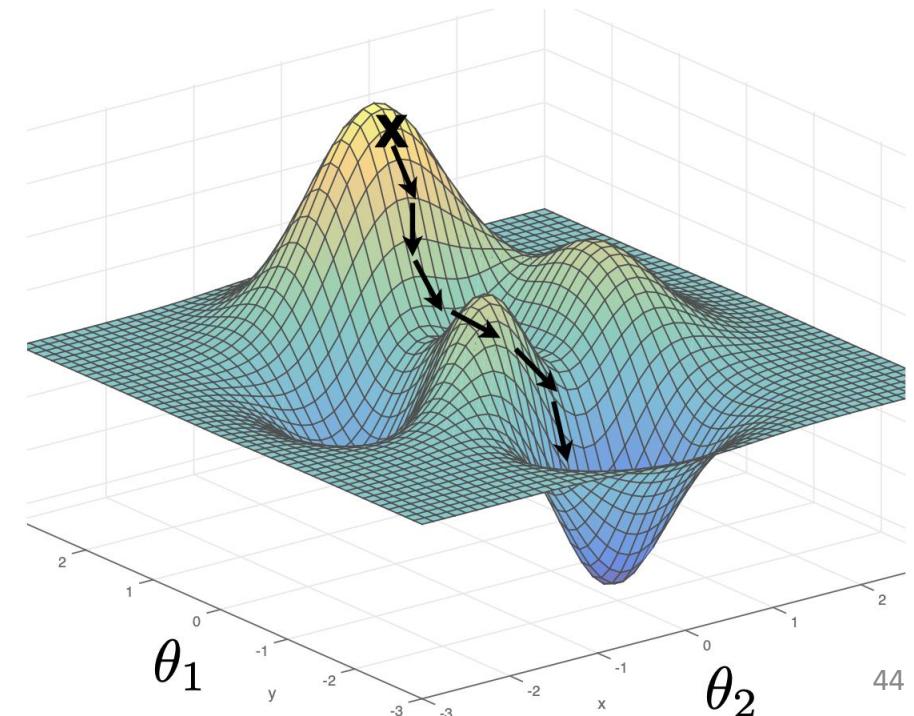
Método do Gradiente (Gradient Descent)

Dada uma função de perda $J(\theta)$ queremos encontrar o conjunto de parâmetros θ^* que minimiza $J(\theta)$.

Sabemos que dado um ponto θ^t , a função $J(\theta)$ irá decrescer mais rapidamente na direção $-\nabla J(\theta^t)$.

Vamos usar essa estratégia para Determinar o mínimo desejado.

$$J(\theta)$$

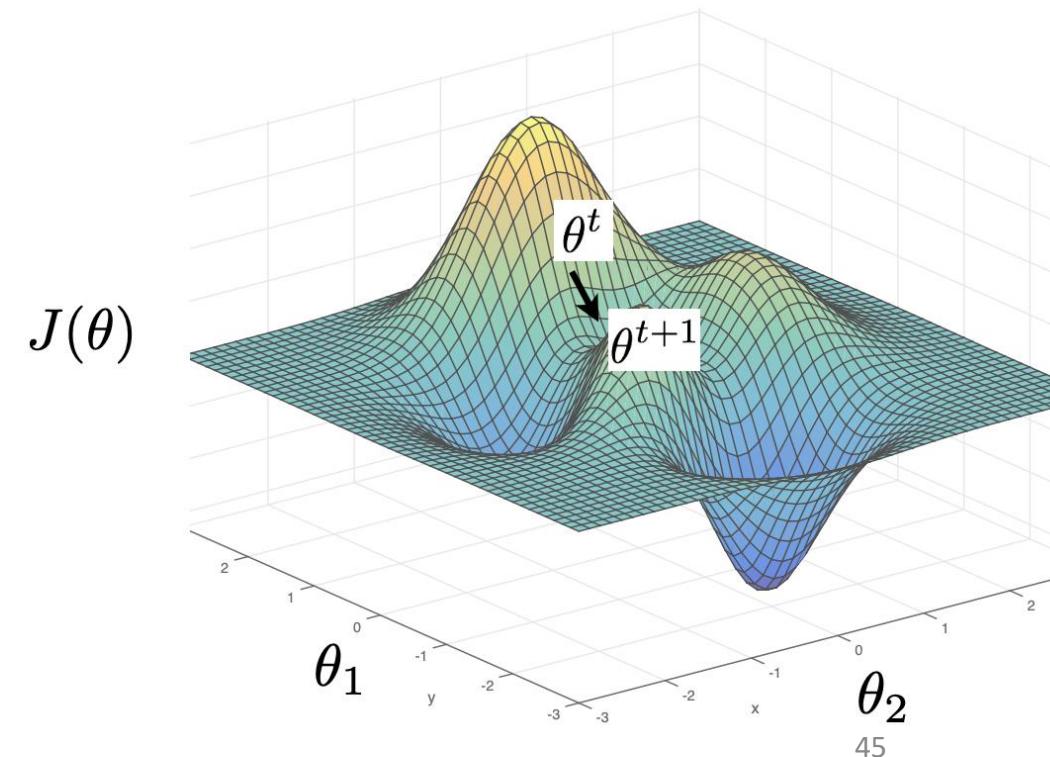


Método do Gradiente (Gradient Descent)

Os passos que vão em direção ao mínimo desejado são calculados usando: $\theta^{t+1} = \theta^t - \eta_t \nabla J(\theta^t)$

A constante η_t é chamada de taxa de aprendizagem (*learning rate*)

Como as atualizações dependem do gradiente, então o cálculo das derivadas $\frac{\partial J}{\partial \theta_i}$ será necessário!



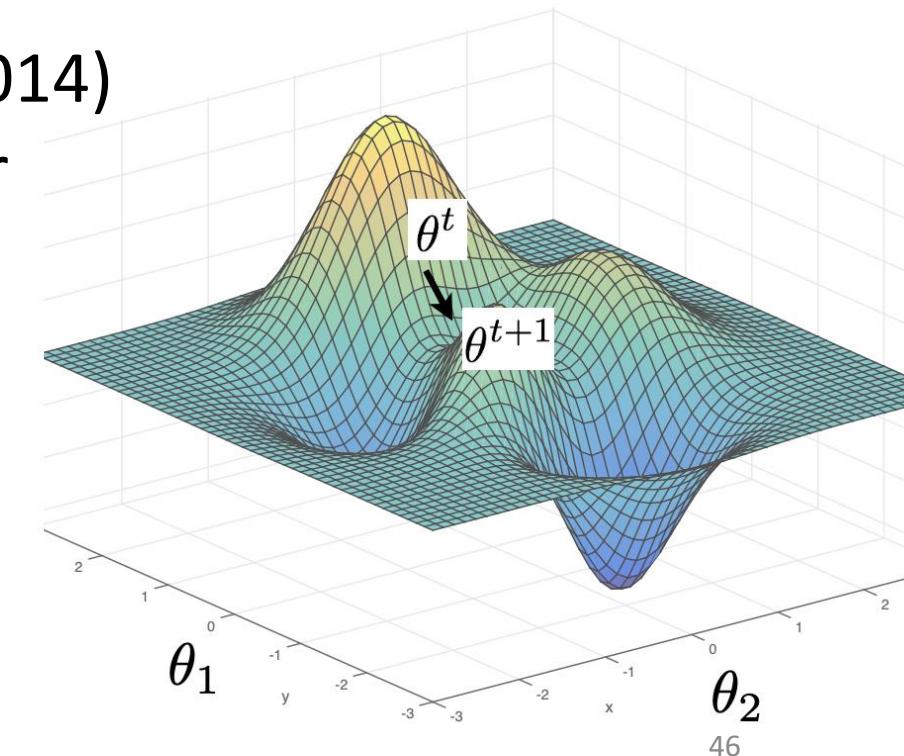
Método do Gradiente (Gradient Descent)

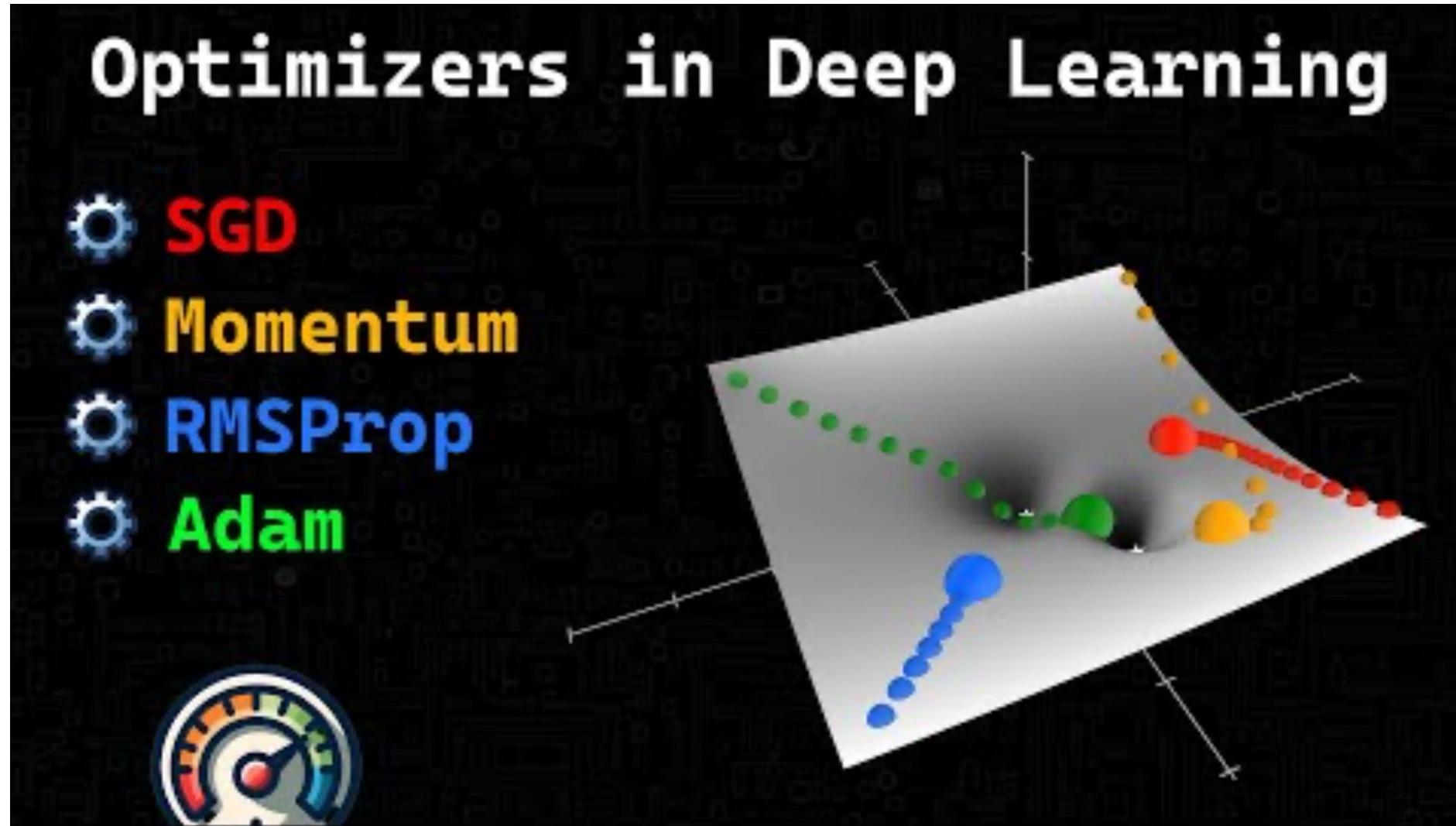
Na prática o cálculo do mínimo local (ou global) das funções de perda $J(\theta)$ é feita utilizando variações mais eficientes do método da descida do gradiente. Os métodos mais comuns são:

- SGD (Stochastic Gradient Descent - 1950)
- ADAM (Adaptive Moment Estimation - 2014)
- L-BFGS (Limited-Memory Broyden–Fletcher –Goldfarb–Shanno algorithm - 1989)

$$J(\theta)$$

Independentemente do método, todos eles dependem do cálculo da derivada de $J(\theta)$ com relação as componentes de θ .





Backpropagation (1989)

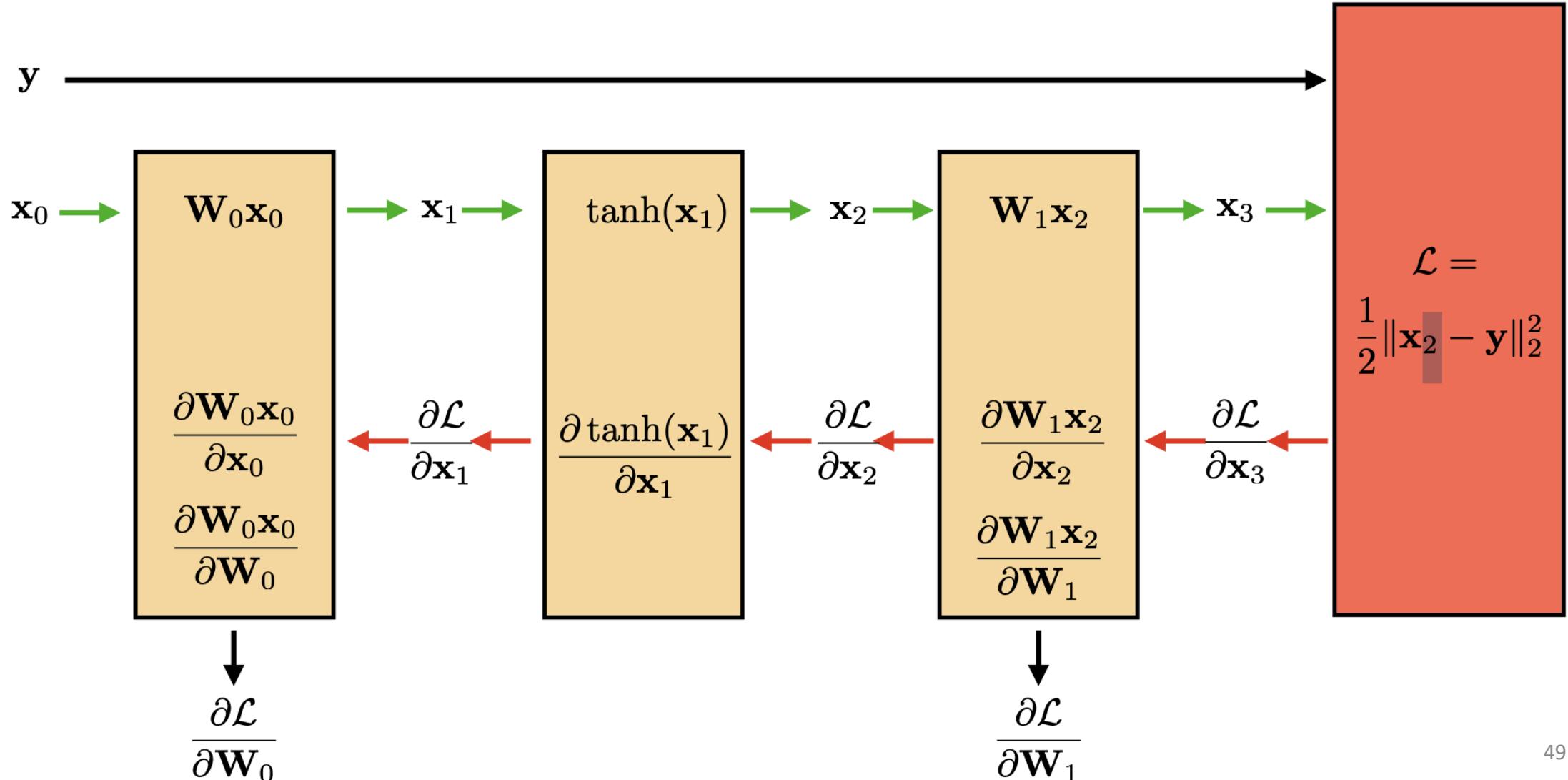
Para determinarmos o conjunto ótimo de parâmetros θ^* que minimiza o erro $J(\theta)$ observamos que $f_\theta(x)$ é uma função conhecida, considerando o seu caráter composicional de funções lineares e funções de ativação. Desta forma, as derivadas de $f_\theta(x)$ tanto em relação a x ou a θ podem ser obtidas diretamente através da regra da cadeia.

O algoritmo que calcula de forma eficiente as derivadas necessárias para a atualização constante da variável θ é conhecido como *backpropagation*.

Ele nada mais é que uma forma engenhosa de calcular as derivadas necessárias considerando as informações conhecidas e a regra da cadeia.

Backpropagation (1989)

Vamos a um exemplo!



Backpropagation (1989)

Voltando ao nosso objetivo é atualizar os pesos. Para isso precisamos calcular as derivadas:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_0} \text{ e } \frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}$$

Com elas, então podemos calcular a atualização dos pesos na forma

$$\mathbf{W}_0^{k+1} = \mathbf{W}_0^k + \eta \left(\frac{\partial \mathcal{L}}{\partial \mathbf{W}_0} \right)^T$$

$$\mathbf{W}_1^{k+1} = \mathbf{W}_1^k + \eta \left(\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} \right)^T$$

Com $\eta < 0$

Backpropagation (1989)

Primeiro calculamos a derivada da perda em relação à saída \mathbf{x}_3 :

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}} = \mathbf{x}_3 - \mathbf{y}$$

Agora, pela regra da cadeia, podemos derivar as equações, trabalhando de trás para frente, para cada termo restante que precisamos:

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_2}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_3} \frac{\partial \mathbf{x}_3}{\partial \mathbf{x}_2} = \boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}} \mathbf{W}_1 \quad \boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_2} \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_2} \frac{\partial \tanh(\mathbf{x}_1)}{\partial \mathbf{x}_1} = \boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_2}} (1 - \tanh^2(\mathbf{x}_1))$$

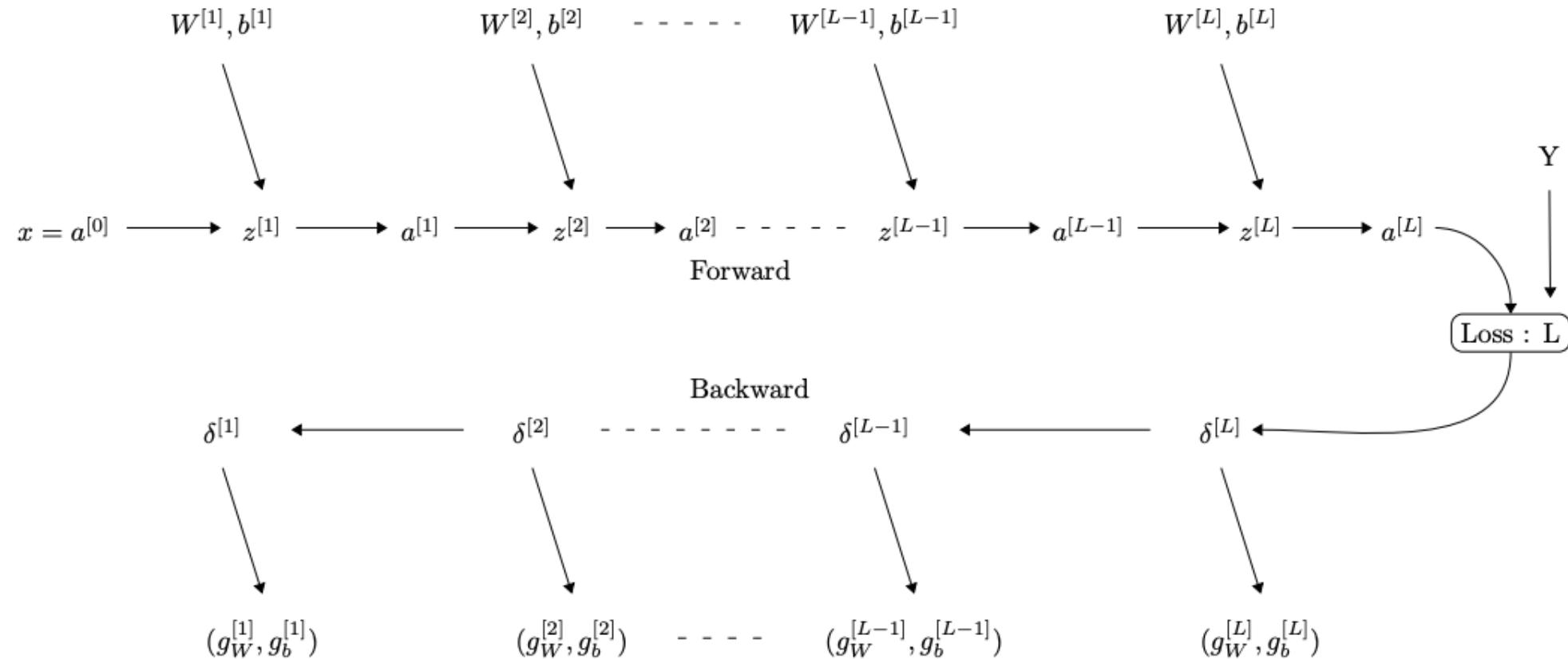
Terminando com nossos dois gradientes necessários para a atualização de peso:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_0} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_1} \frac{\partial \mathbf{x}_1}{\partial \mathbf{W}_0} = \mathbf{x}_0 \boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_1}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_3} \frac{\partial \mathbf{x}_3}{\partial \mathbf{W}_1} = \mathbf{x}_2 \boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}}$$

Backpropagation (1989)

Parameter Values



Gradient Values

Backpropagation (1989)

Algorithm 5.2: Back-Propagation for fully connected networks

Input: a sample (x, y) and parameter values $\theta = (\theta^{[1]}, \dots, \theta^{[L]})$

Output: derivatives of the cost $(g_W^{[1]}, g_b^{[1]}), \dots, (g_W^{[L]}, g_b^{[L]})$

- 1 Compute $a^{[\ell]}$ and $z^{[\ell]}$ for $\ell = 1, \dots, L$ (Forward pass)
 - 2 Compute $\delta^{[L]} = \frac{\partial C}{\partial z^{[L]}}$
 - 3 Set $g_W^{[L]} = \delta^{[L]} a^{[L-1]^\top}$ and set $g_b^{[L]} = \delta^{[L]}$
 - 4 **for** $\ell = L - 1, \dots, 1$ **do**
 - 5 Compute $\delta^{[\ell]} = \text{Diag}(\dot{\sigma}^{[\ell]}(z^{[\ell]})) W^{[\ell+1]^\top} \delta^{[\ell+1]}$
 - 6 Set $g_W^{[\ell]} = \delta^{[\ell]} a^{[\ell-1]^\top}$ and set $g_b^{[\ell]} = \delta^{[\ell]}$
-

Teorema da Aproximação Universal (1989)

Lembrando: o objetivo de uma rede *feedforward* é aproximar alguma função $g: \mathbb{R}^p \rightarrow \mathbb{R}^q$ por uma aplicação $f_\theta: \mathbb{R}^p \rightarrow \mathbb{R}^q$ que aprende o valor dos parâmetros θ e que idealmente resulta em $g(x) \approx f_\theta(x)$. Mas porque isso funciona?

“Aproximação Universal”: Seja σ qualquer função sigmoidal contínua. Então, somas finitas da forma $G(x) = \sum_{j=1}^N \alpha_j \sigma(y_j^T x + \theta_j)$ são densas em $C(I_n)$. Em outras palavras, dada qualquer $f \in C(I_n)$ e $\varepsilon > 0$, existe uma soma $G(x)$, da forma acima, para a qual $|G(x) - f(x)| < \varepsilon$ para todo $x \in I_n$.

Como podemos implementar isso?



Parte 4

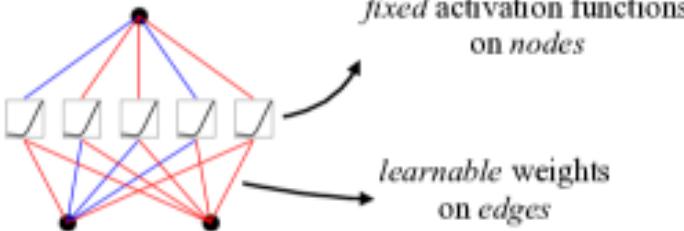
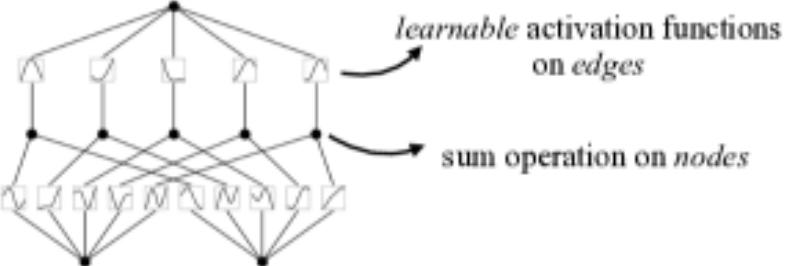
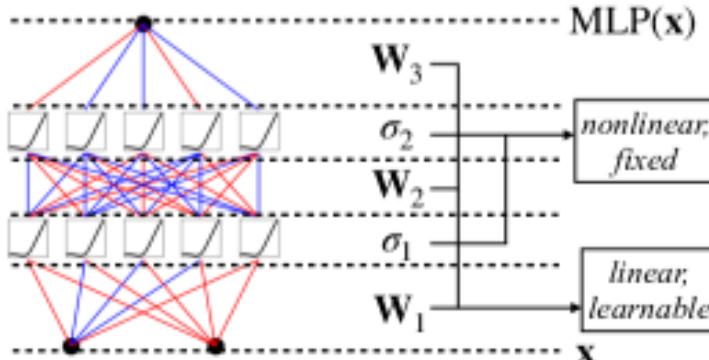
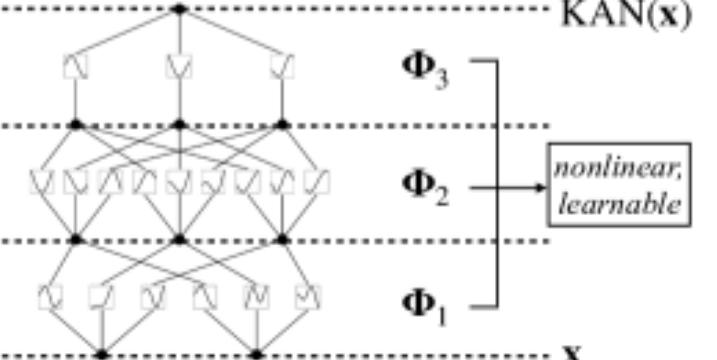
Apesar do poder das redes neurais, nem tudo são flores.

- Necessidade de grande volume de dados (maldição da dimensionalidade)
- Caro para treinar
- Possibilidade de *overfitting*
- Baixa, ou nenhuma, interpretabilidade
- Grande quantidade de hiperparâmetros
- Qualidade dos dados
- Problema de *vanishing gradient*
- ...

Com o intuito de mitigar alguns dos problemas usuais das DNN outras abordagens diferentes de redes neurais podem surgir. Mais recentemente as redes de Kolmogorov-Arnold fizeram um barulho na comunidade da IA.

- Inspiradas no teorema da representação de Kolmogorov-Arnold (qualquer função multivariada pode ser decomposta em uma soma finita de funções univariadas). Sugere-se que isso reduza a maldição da dimensionalidade.
- As funções de ativação são treináveis (B-Splines), o que pode aumentar a interpretabilidade.
- Para alguns casos ela fornece melhor precisão que as DNN.

Kolmogorov-Arnold Networks

Model	Multi-Layer Perceptron (MLP)	Kolmogorov-Arnold Network (KAN)
Theorem	Universal Approximation Theorem	Kolmogorov-Arnold Representation Theorem
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(e)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^n \phi_{q,p}(x_p) \right)$
Model (Shallow)	<p>(a) </p> <p>fixed activation functions on nodes</p> <p>learnable weights on edges</p>	<p>(b) </p> <p>learnable activation functions on edges</p> <p>sum operation on nodes</p>
Formula (Deep)	$\text{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\text{KAN}(\mathbf{x}) = (\Phi_3 \circ \Phi_2 \circ \Phi_1)(\mathbf{x})$
Model (Deep)	<p>(c) </p> <p>MLP(\mathbf{x})</p> <p>\mathbf{W}_3</p> <p>σ_2 <i>nonlinear, fixed</i></p> <p>\mathbf{W}_2</p> <p>σ_1 <i>linear, learnable</i></p> <p>\mathbf{W}_1</p> <p>\mathbf{x}</p>	<p>(d) </p> <p>KAN(\mathbf{x})</p> <p>Φ_3</p> <p>Φ_2 <i>nonlinear, learnable</i></p> <p>Φ_1</p> <p>\mathbf{x}</p>

Por hoje é só, pessoal!

FIM!

Obrigado pela atenção!