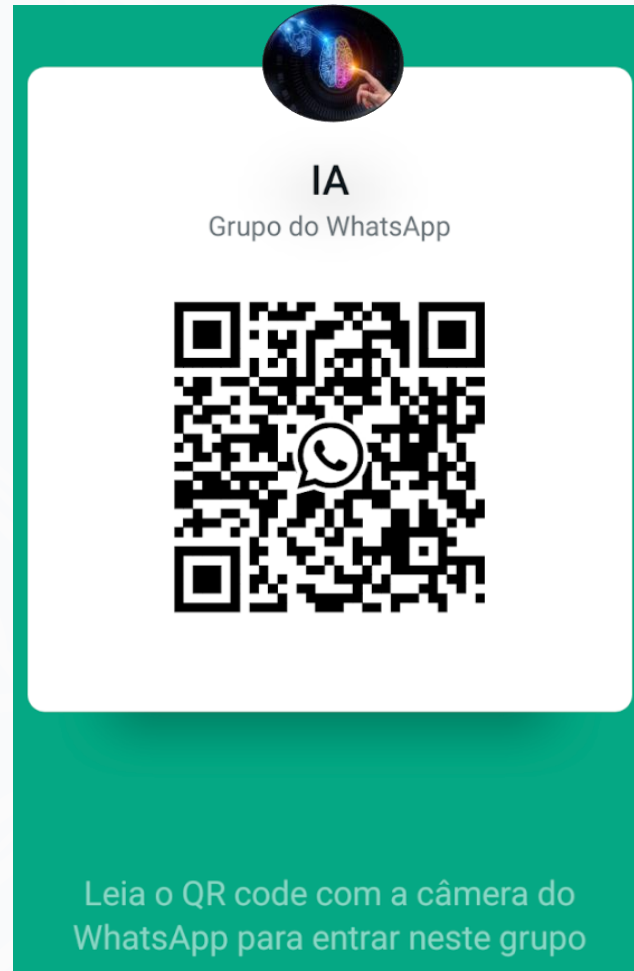


# Inteligência Artificial

Introdução às Redes Neurais Artificiais  
Inspiração biológica e aplicações

Profº - Dr. Thales Levi Azevedo Valente  
[thales.l.a.valente@gmail.com.br](mailto:thales.l.a.valente@gmail.com.br)

# Grupo da turma 2024.2



<https://chat.whatsapp.com/JFB6CgOI7IMCoYmolKEK62>

# Sejam Bem-vindos !



**Os celulares devem  
ficar no silencioso  
ou desligados**

Pode ser utilizado  
apenas em caso  
de emergência



**Boa tarde/noite, por  
favor e com licença  
DEVEM ser usados**

Educação é  
essencial

# Introdução

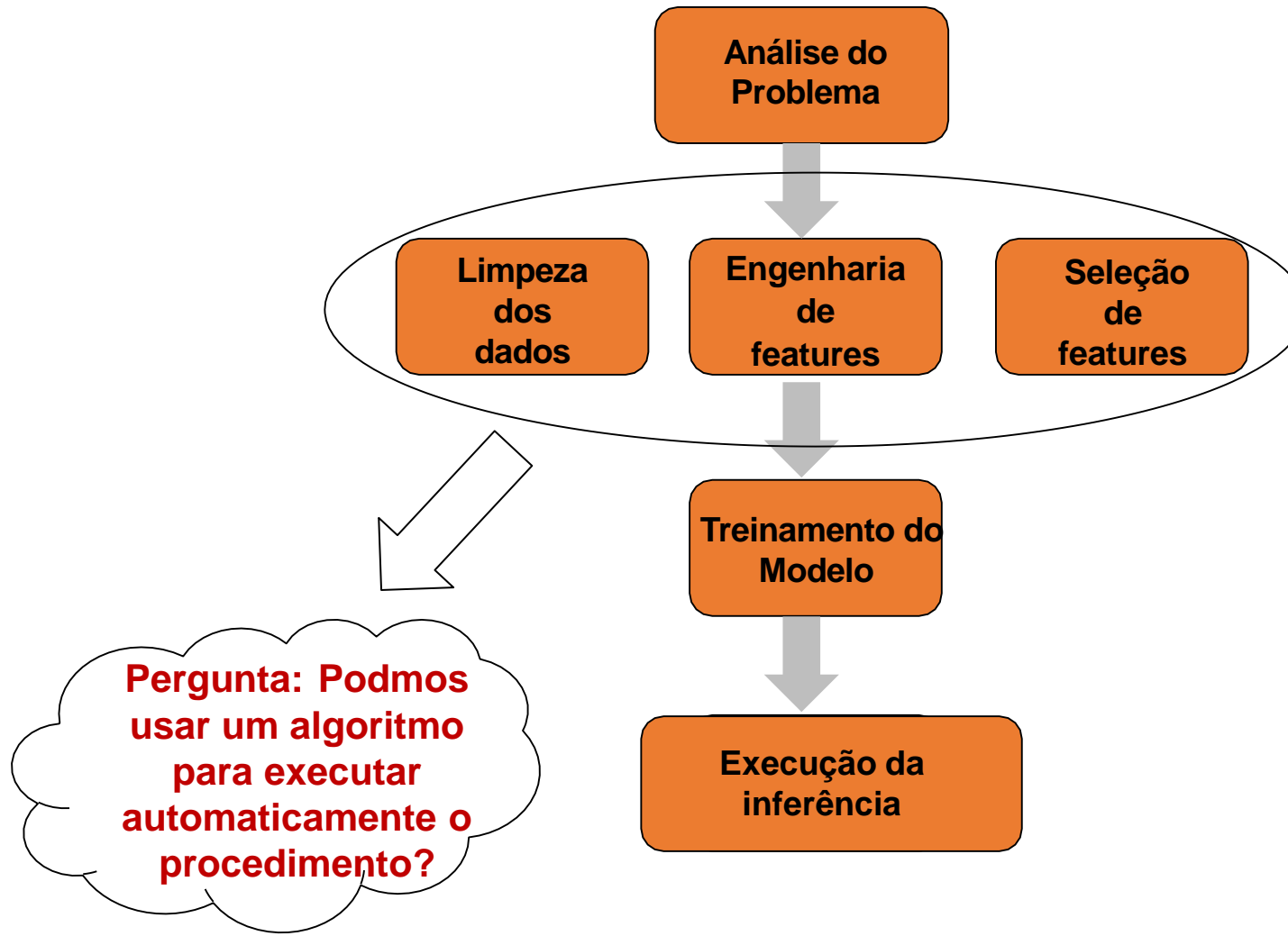
## Tradicional vs Deep Learning

---

Aprendizado de Máquina Tradicional	Deep Learning
Baixos requisitos de hardware no computador: Dado o uso limitado de computação, o computador geralmente não precisa de uma GPU para computação paralela	Altos requisitos de hardware no computador: Para executar operações matriciais em dados massivos, o computador precisa de uma GPU para realizar computação paralela
Aplicável a treinamentos com pequenas quantidades de dados e cujo desempenho não pode ser melhorado continuamente à medida que a quantidade de dados aumenta	O desempenho pode ser alto quando há parâmetros de peso de alta dimensão e grandes quantidades de dados de treinamento disponíveis
Seleção manual de características	Extração automática de características baseada em algoritmos
Manual feature selection	Algorithm-based automatic feature extraction
Características fáceis de explicar	Características difíceis de explicar

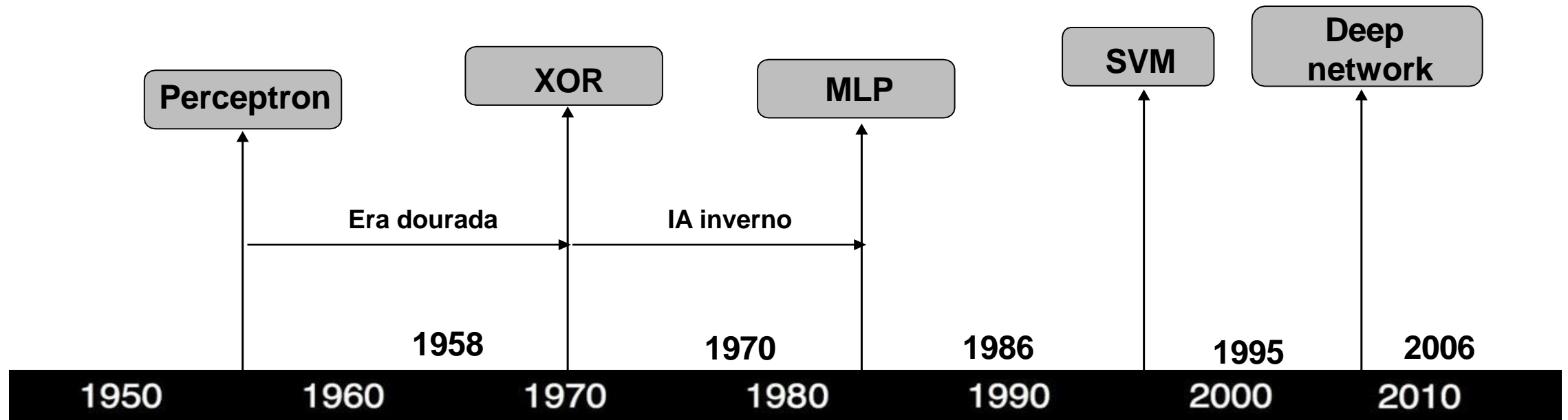
# Introdução

## Tradicional vs Deep Learning



# Introdução

Alguns principais fatos históricos



# Introdução

## Conceitos

---

- **Definição por Hecht Nielsen (pesquisador renomado de redes neurais – EUA)**
  - ✓ Sistema computacional composto por elementos de processamento simples e altamente interconectados, que processam informações por meio de uma resposta dinâmica a entradas externas.
- **Outras definições**
  - ✓ Sistema de processamento de informações projetado para imitar a estrutura e as funções do cérebro humano com base em sua origem, características e explicações.
  - ✓ Formada por neurônios artificiais conectados entre si, a rede neural extrai e simplifica a microestrutura e as funções do cérebro humano.

# Introdução

## Conceitos - Processamento neural

- **Consiste em todo processo em que a rede neural é executada. Divido em duas etapas**
  - ✓ Learning: consiste na atualização dos pesos sinápticos. Sem atualização dos pesos, sem aprendizado
  - ✓ Recall:
    - ✓ resposta da rede mediante a apresentação de um padrão de entrada
    - ✓ Refere-se a recuperação da informação ou inferência
    - ✓ Não há atualização dos pesos, apenas resposta





# Introdução

## Conceitos - Processamento neural

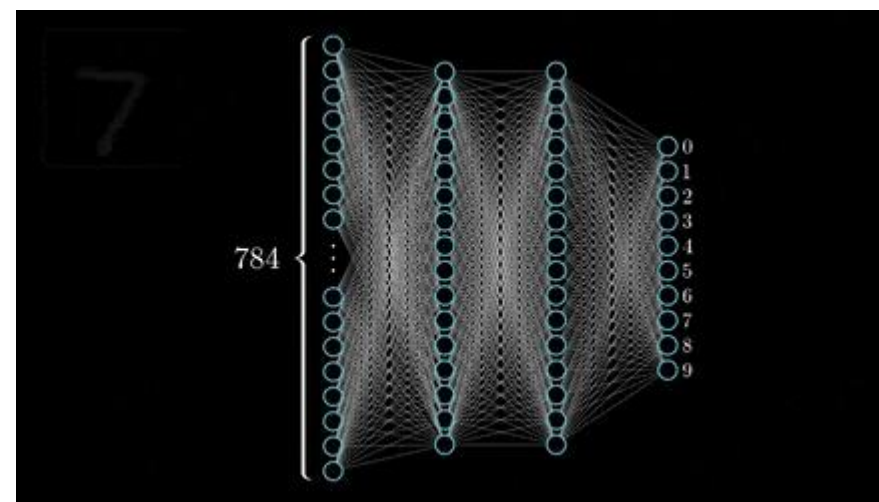
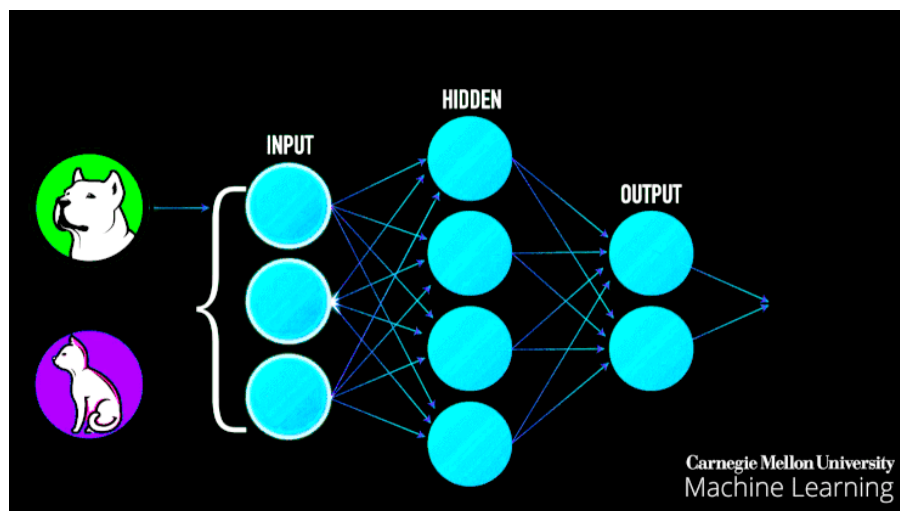
- **Consiste em todo processo em que a rede neural é executada**
- **Divido em duas etapas**
  - ✓ Learning: consiste na atualização dos pesos sinápticos.
    - ✓ Sem atualização dos pesos, sem aprendizado



# Introdução

## Conceitos - Processamento neural

- **Consiste em todo processo em que a rede neural é executada**
- **Divido em duas etapas**
  - ✓ Recall: resposta da rede mediante a apresentação de um padrão de entrada.
    - ✓ Refere-se a recuperação da informação ou inferência
    - ✓ Não há atualização dos pesos, apenas resposta



# Introdução

Conceitos - Processamento neural - Learning

---

- **Corresponde ao treinamento da rede**
- **Considere uma criança hipotética que nunca viu gatos**
- **Imagine que você irá ensinar a ela o que é um gato. De forma simplificada, este processo poderia consistir em**
  - ✓ Apresentar algum animal a ela
  - ✓ Perguntar se é um gato
    - ✓ Resposta correta: não há ajuste do conhecimento
    - ✓ Resposta errada: ela aprende que o dado exemplo não ou é um gato baseado nas características que ela observou

# Introdução

Conceitos - Processamento neural - Recall

- **Corresponde a inferência da rede**
- **A criança ensinada a reconhecer gatos identificará corretamente com maior probabilidade futuros gatos apresentados se, dado um conjunto de exemplos**
  - ✓ Mais exemplos diferentes de gatos apresentar a ela
  - ✓ Mais exemplos difíceis apresentar, ou seja, amostras de classes semelhantes a classe gato (exemplo: outros felinos onça-pintada, seval, gato-leopardo)



# Introdução

## Conceitos - Camadas

---

### ■ **Entrada**

- ✓ Função: recebe os sinais de entrada (características, pixels).
- ✓ Número de neurônios: quantidade de sinais de entrada.
- ✓ Número de camadas: 1 camada

### ■ **Saída**

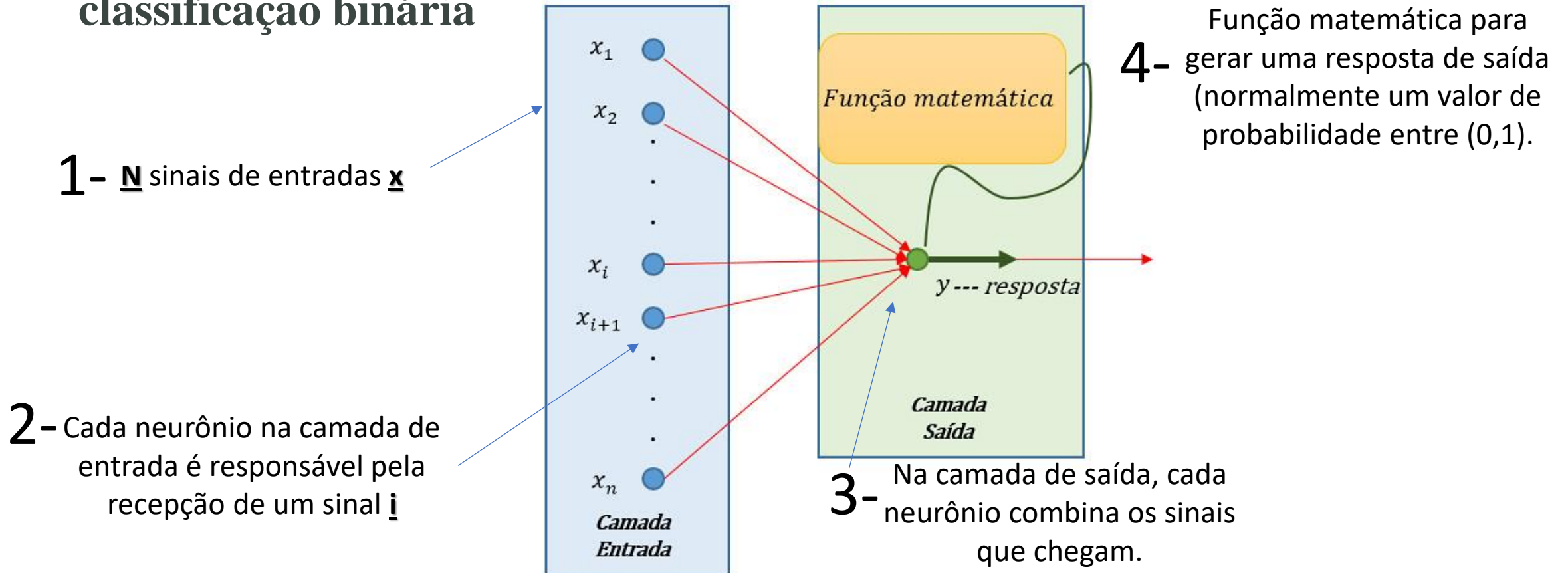
- ✓ Função: fornece a resposta da rede dado as regras de processamento aplicadas desde a camada de entrada.
- ✓ número de neurônios:
  - ✓ classificação binária (2 classes): 1 neurônio
  - ✓ classificação não-binária: quantidade de classes
- ✓ número de camadas: 1 camada geralmente



# Introdução

## Conceitos - Camadas

- Ilustração simplificada de uma rede neural com as camadas de entrada e saída. Por simplificação, consideramos que a camada de saída realiza **classificação binária**



# Introdução

## Conceitos - Camadas

---

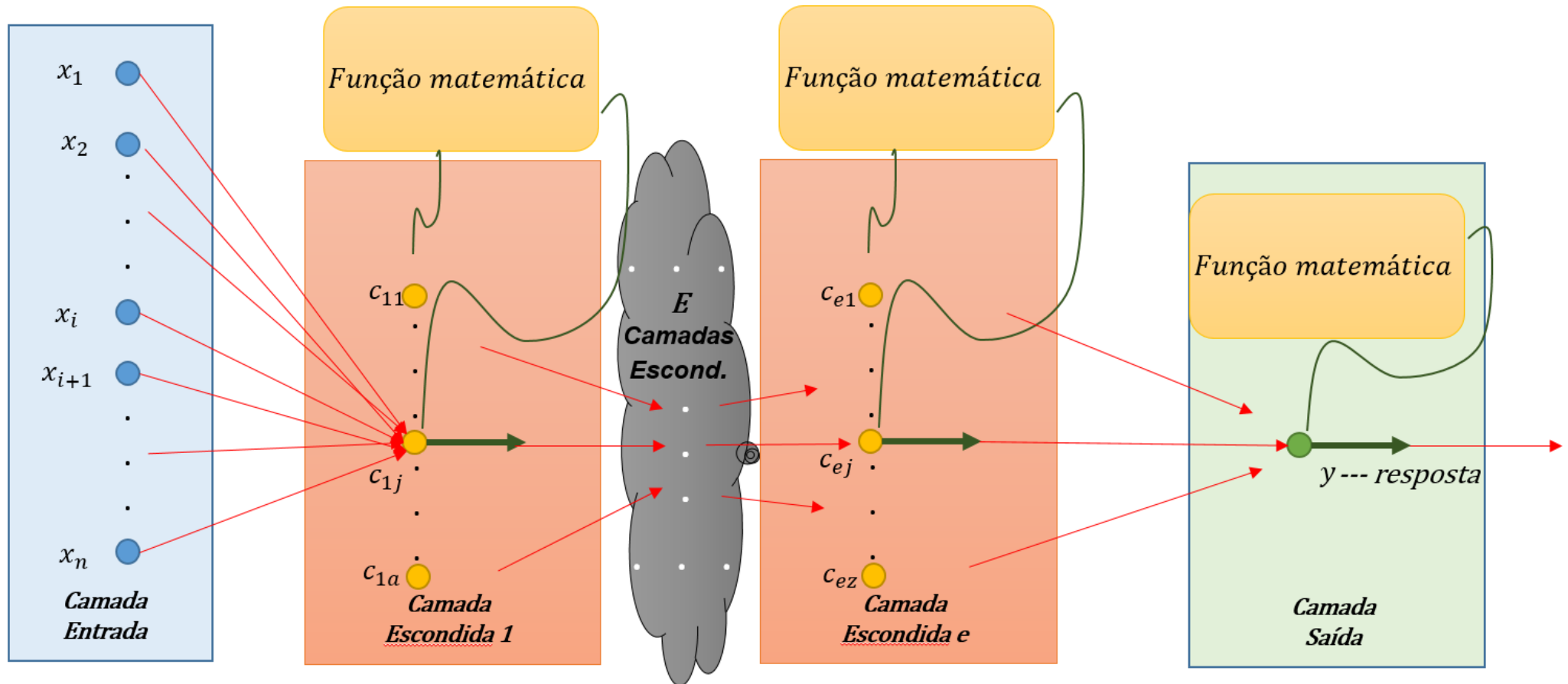
### ■ **Escondida ou intermediária**

- ✓ Função: realizar um mapeamento não-linear dos dados. Nessa camada é realizada a maior parte do processamento da rede
- ✓ Número de neurônios
  - ✓ Depende do problema
  - ✓ Pode ser determinado através de heurísticas
- ✓ Número de camadas
  - ✓ Normalmente 1 ou 2
  - ✓ OBS: já foi provado matematicamente que duas camadas resolve qualquer problema

# Introdução

## Conceitos - Camadas

### ■ Escondida ou intermediária



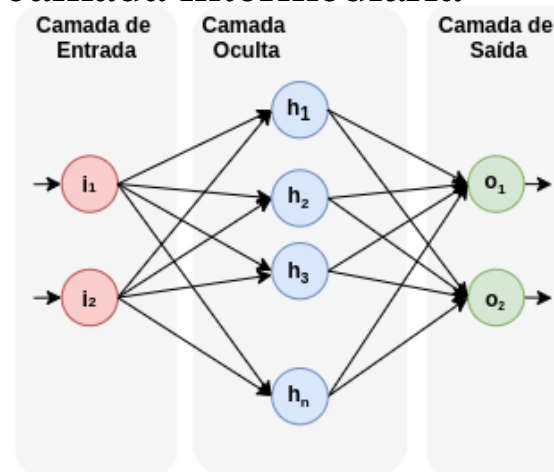
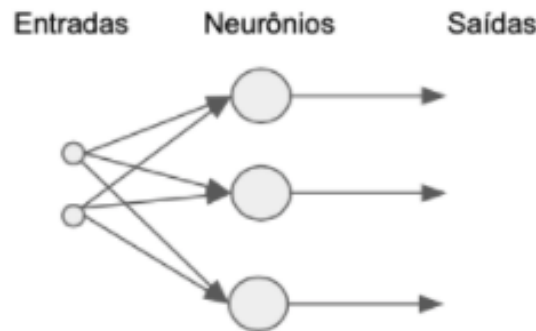


# Introdução

## Conceitos – Tipos de Arquiteturas

### ■ Feedforward

- ✓ Fluxo de dados é sempre em uma única direção ("esquerda para direita")
- ✓ Neurônios de uma camada enviam sinais apenas para neurônios da camada seguinte. "Neurônio da camada  $c$  para  $c+1$ "
- ✓ Tipos
  - ✓ Camada simples / monocamadas: não possui camadas intermediárias
  - ✓ Múltiplas camadas: possui pelo menos 1 camada intermediária

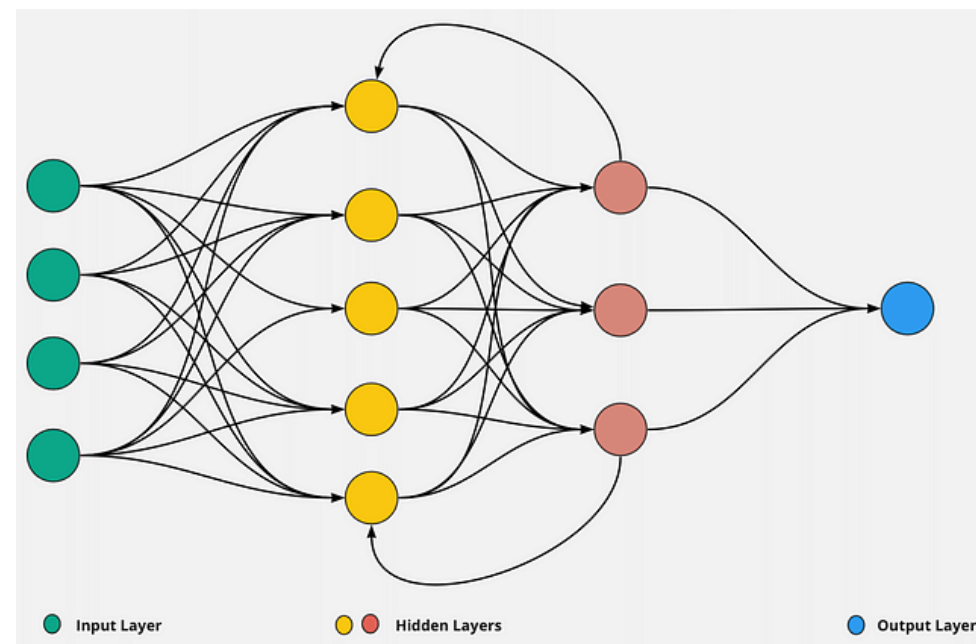
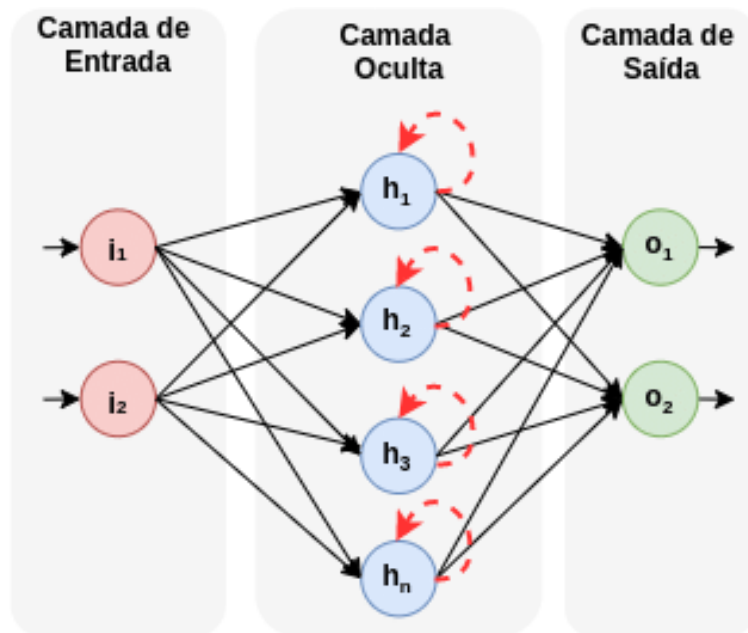


# Introdução

## Conceitos – Tipos de Arquiteturas

### ■ Recorrentes

- ✓ Fluxo de dados em várias direções
- ✓ Neurônios de uma camada podem enviar sinais tanto para neurônios da camada seguinte, quanto da mesma camada ou camada anterior



# Propagação do sinal

---

- **O sinal se propaga na rede de um neurônio para outro através de conexões**
  - ✓ Sinapses ponderadas.
- **O neurônio receptor realiza uma combinação linear das saídas de cada neurônio emissor com os pesos das conexões sinápticas**
  - ✓ Normalmente é utilizado algum mecanismo de inicialização de pesos
  - ✓ A combinação linear determina o potencial de ativação do neurônio receptor e é dada por:

$$p(j) = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix} \times [x_1 \quad x_2 \quad \dots \quad x_n] = [w_1x_1 + w_2x_2 + \dots + w_nx_n]$$

*ou*

$$p(j) = \sum_{i=1}^n (w_i \cdot x_i)$$

# Propagação do sinal

---

- ✓ A combinação linear determina o potencial de ativação do neurônio receptor e é dada por:

$$p(j) = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix} \times [x_1 \quad x_2 \quad \dots \quad x_n] = [w_1x_1 + w_2x_2 + \dots + w_nx_n]$$

ou

$$p(j) = \sum_{i=1}^n (w_i \cdot x_i)$$

- ✓  $x$  representa o sinal de cada neurônio emissor
- ✓  $x_i$  representa o i-ésimo neurônio emissor
- ✓  $n$  é o número de neurônios emissores
- ✓  $w_i$  representa o peso da conexão entre o i-ésimo neurônio emissor e o neurônio receptor
- ✓  $j$  representa o j-ésimo neurônio receptor
- ✓  $p(j)$  representa o potencial de ativação do j-ésimo neurônio receptor

# Função de Ativação

---

- **Definição:** é uma função matemática aplicada em todos os neurônios de uma rede neural, exceto nos neurônios da camada de entrada.
- Normalizam os valores de saída dentro de um determinado intervalo (exemplo:  $[-1,1]$  ou  $[0,1]$ ).
- Introduzem não linearidade, essencial para aprender relações complexas
- Influenciam a convergência e estabilidade do treinamento da rede neural

# Função de Ativação

## Principais tipos de funções de ativação

### ■ **Funções de Ativação Lineares**

- ✓ São usadas apenas em saídas quando se deseja prever valores contínuos (ex.: regressão).
- ✓ Problema: Se usada em camadas ocultas, a rede neural se comporta como um modelo linear, perdendo sua capacidade de aprender relações complexas

$$f(x) = ax + b$$

# Função de Ativação

## Principais tipos de funções de ativação

### ■ Funções de Ativação Não Lineares

- ✓ Essas funções são essenciais para capturar padrões não lineares nos dados
- ✓ O que significa "Introduzir Não Linearidade" em Redes Neurais?
  - ✓ Funções de ativação introduzirem não linearidade se refere à capacidade da rede neural de aprender relações complexas nos dados
  - ✓ Sem essas funções a rede neural seria apenas uma combinação linear de pesos e entradas
  - ✓ Se a rede tiver várias camadas, a saída ainda será uma combinação linear das entradas iniciais, ou seja

$$y = W_3(W_2(W_1x + b_1) + b_2) + b_3$$

- ✓ Se aplicarmos apenas operações lineares, qualquer número de camadas se reduz a uma única camada, pois a soma e a multiplicação de matrizes continuam sendo transformações lineares

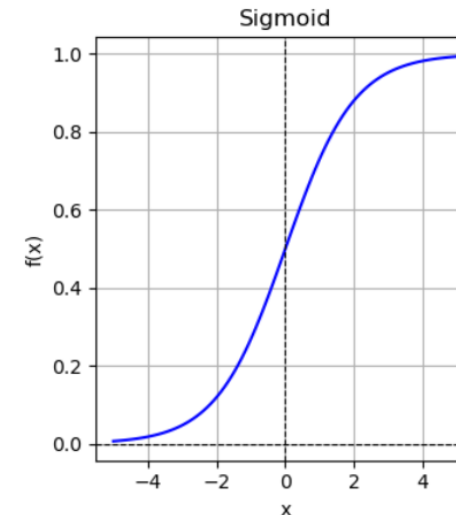
# Função de Ativação

Funções de ativação não lineares mais comuns

## ■ Sigmoid (Logística)

- ✓ Intervalo: (0,1)
- ✓ Vantagem: Suaviza a saída e é útil para problemas de classificação binária.
- ✓ Problema: Pode sofrer de vanishing gradients, onde os gradientes se tornam muito pequenos para atualizar os pesos corretamente.
- ✓ Usada para: Modelos que exigem saída entre 0 e 1, como classificação binária

$$f(x) = \frac{1}{1 + e^{-x}}$$





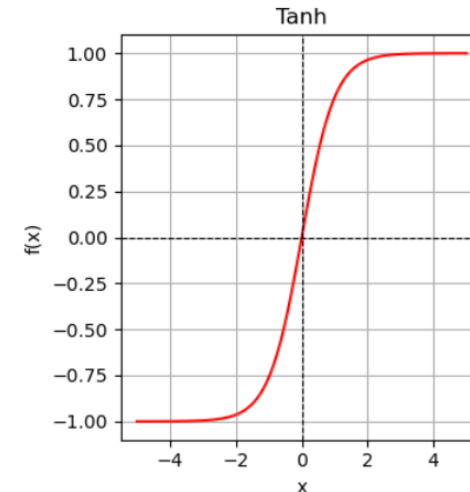
# Função de Ativação

Funções de ativação não lineares mais comuns

## ■ Tanh (Tangente Hiperbólica)

- ✓ Intervalo: (0,1)
- ✓ Vantagem: Mais centrada em zero que a Sigmoid, tornando-a melhor para camadas ocultas.
- ✓ Problema: Ainda pode sofrer com vanishing gradients.
- ✓ Usada para: Modelos onde é importante ter saídas centradas em zero

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



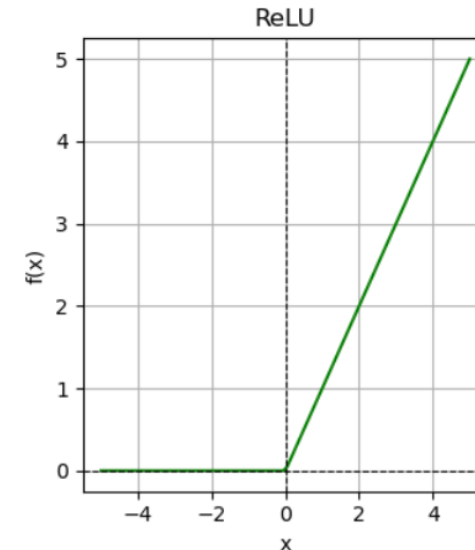
# Função de Ativação

## Funções de ativação não lineares mais comuns

### ■ ReLU (Rectified Linear Unit)

- ✓ Intervalo:  $(0, +\infty)$
- ✓ Vantagem: resolve o problema do vanishing gradient para valores positivos.
- ✓ Problema: Neurônios podem "morrer" quando  $x < 0$  (neurônios mortos).
- ✓ Usada para: A maioria das redes profundas devido à sua eficiência computacional

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$






# Função de Ativação



## Funções de ativação não lineares mais comuns

### ▪ ReLU (Rectified Linear Unit)

#### ✓ Se a entrada ( $x$ ) for positiva ou zero, a saída é $x$ (linear)

- ✓  Evita o problema do vanishing gradient presente em funções como Sigmoid e Tanh.
- ✓  Computacionalmente eficiente (não envolve exponenciais, apenas comparação).
- ✓  Funciona bem para redes profundas, permitindo o aprendizado de representações complexas.

#### ✓ Se a entrada for negativa, a saída é zero.

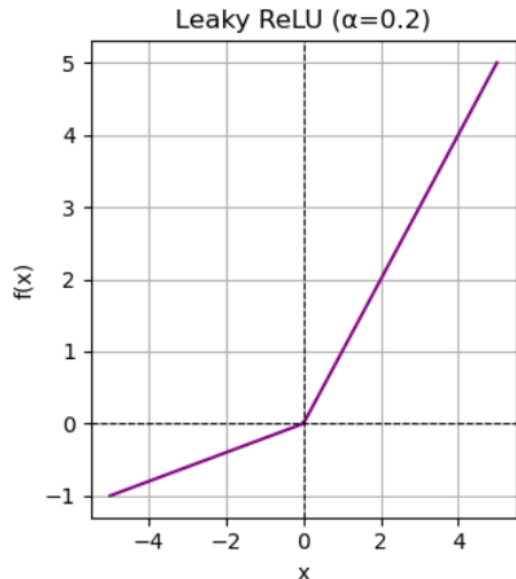
- ✓  Isso significa que, para alguns neurônios, os pesos podem ser ajustados de forma que eles nunca mais ativem durante o treinamento.
- ✓  Como o gradiente da ReLU para  $x < 0$  é zero, os pesos desses neurônios não são mais atualizados, tornando-os “neurônios mortos”
- ✓ O problema é crítico se muitos neurônios morrerem no início do treinamento, pois partes inteiras da rede deixam de contribuir

# Função de Ativação

## Funções de ativação não lineares mais comuns

### ▪ **Leaky ReLU (ReLU com vazamento)**

- ✓ Intervalo:  $(-\infty, +\infty)$
- ✓ Vantagem: evita o problema dos neurônios mortos na ReLU pura.
- ✓ Problema: Pode introduzir um pequeno viés na saída da rede.
- ✓ Usada para: Modelos onde ReLU apresentar problemas de neurônios inativos








$$f(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases} \text{ onde } \alpha \text{ é um valor pequeno e positivo (ex: 0.01)}$$

# Função de Ativação

## Funções de ativação não lineares mais comuns

### ▪ **Leaky ReLU (ReLU com vazamento)**

- ✓ Se a entrada ( $x$ ) for positiva, ela funciona exatamente como a ReLU
- ✓ **Para valores negativos, em vez de zerar a saída, ela permite um pequeno valor negativo ( $\alpha x$ ), evitando o problema dos neurônios mortos**
  - ✓  Evita o problema dos neurônios mortos, pois valores negativos ainda contribuem para a atualização dos pesos.
  - ✓  Melhor estabilidade no aprendizado, pois mantém gradientes pequenos para entradas negativas.
  - ✓  Melhor desempenho em algumas redes profundas, pois permite que a rede use mais neurônios de forma eficiente..
  - ✓  O valor de  $\alpha$  precisa ser escolhido com cuidado. Se for muito pequeno, pode não resolver completamente o problema da ReLU.
  - ✓  Não é garantido que seja melhor que ReLU em todos os casos. Em alguns datasets, ReLU ainda pode funcionar melhor.

# Função de Ativação

## Funções de ativação não lineares mais comuns

### ▪ **Softmax**

- ✓ Intervalo: (0, 1), garantindo que a soma de todas as saídas seja 1
- ✓ Usada para: Classificação com múltiplas classes (exemplo: reconhecimento de imagens).
- ✓ A função Softmax para um conjunto de entradas  $x_1, x_2, \dots, x_n$  é definida como:







$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

- ✓ Onde:
  - ✓  $e^{x_i}$  é a exponencial do logit  $x_i$  e a soma de todas as exponenciais no denominador garante a normalização dos valores.

# Função de Ativação

## Funções de ativação não lineares mais comuns

### ■ Softmax

- ✓  Conversão de valores brutos em probabilidades: facilita a interpretação dos resultados.
- ✓  Ótima para classificação multiclasse: usada na última camada de redes neurais para prever a classe mais provável.
- ✓  Diferenciação clara entre as classes: amplifica as diferenças entre os logits.
- ✓  Sensível a valores extremos: se um logit for muito grande, ele pode dominar os outros, gerando uma probabilidade muito alta para uma única classe.
- ✓  Não é ideal para classificação binária: para problemas binários, o Sigmoid é mais eficiente.
- ✓  Não lida bem com outliers: se os logits tiverem diferenças muito grandes, a Softmax pode produzir distribuições enviesadas.

# Função de Ativação

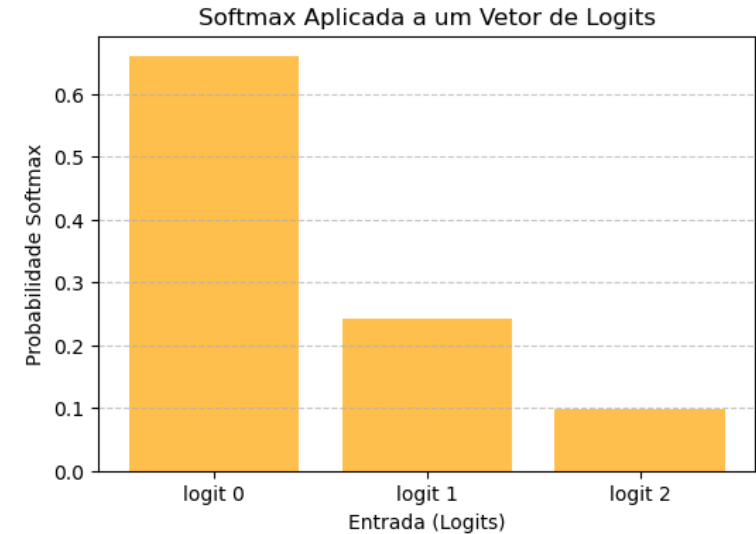
## Funções de ativação não lineares mais comuns

### ■ Softmax – exemplo numérico

- ✓ Se tivermos os logits  $x=[2.0,1.0,0.1]$ .
- ✓ Aplicamos a Softmax
  - ✓ Calculamos as exponenciais  $e^2=7.389, e^1=2.718, e^{0.1}=1.105$
  - ✓ Somamos os valores:  $7.389+2.718+1.105=11.212$
  - ✓ Dividimos cada valor pela soma total:

$$\frac{7.389}{11.212} = 0.659, \quad \frac{2.718}{11.212} = 0.242, \quad \frac{1.105}{11.212} = 0.099$$

- ✓ Resultado:  $[0.659, 0.242, 0.099]$
- ✓ A classe 0 tem o maior logit, ou **maior probabilidade (65.9%)**.





# Função de Ativação

Como escolher as funções?

---

- **Classificação Binária:** Sigmoid (saída)
- **Classificação Multiclasse:** Softmax (saída)
- **Redes Profundas:** ReLU (ocultas), Leaky ReLU se houver neurônios mortos
- **Séries Temporais:** Tanh ou ReLU, dependendo do modelo
- **ReLU vs Leaky ReLU**
  - ✓ Se a rede estiver funcionando bem com ReLU, continue usando.
  - ✓ Se houver neurônios mortos, experimente Leaky ReLU.
  - ✓ Para redes muito profundas, a Leaky ReLU pode ser mais estável.
  - ✓ Algumas variantes como Parametric ReLU (PReLU) e Exponential Linear Unit (ELU) podem ser alternativas

# Função de Ativação

## Comparação das Funções de Ativação

Característica	ReLU	Leaky ReLU
Definição	$f(x) = \max(0, x)$	$f(x) = \max(\alpha x, x)$ com $\alpha > 0$
Saída para $x < 0$	Sempre zero ❌	Pequeno valor negativo ✅
Neurônios Mortos?	Sim ❌	Não ✅
Computacionalmente Eficiente?	Sim ✅	Sim ✅
Melhor para redes profundas?	Às vezes ❌	Às vezes ❌
Usado frequentemente em	CNNs, redes profundas gerais	Quando a ReLU apresenta problemas

# Função de Ativação

## Comparação das Funções de Ativação

Função	Intervalo de Saída	Vantagem	Problema	Uso Ideal
<b>Sigmoid</b>	$(0,1)$	Boa para probabilidade	Vanishing gradient	Classificação binária
<b>Tanh</b>	$(-1,1)$	Melhor que Sigmoid, centrada em zero	Ainda sofre de vanishing gradient	Camadas ocultas
<b>ReLU</b>	$[0, +\infty)$	Simples e eficiente	Neurônios podem morrer	Redes profundas
<b>Leaky ReLU</b>	$(-\infty, +\infty)$	Evita neurônios mortos	Pequeno viés nos valores negativos	Redes profundas com problemas de ReLU
<b>Softmax</b>	$(0,1)$ (soma = 1)	Probabilidades para múltiplas classes	Não usada em camadas ocultas	Classificação multiclasse

# Introdução à Função Custo





Como escolher as funções?

---

- **Em redes neurais, a função custo mede o erro entre as previsões do modelo e os valores reais**
- **O objetivo do treinamento é minimizar essa função, ajustando os pesos da rede**
- **Cada tipo de problema requer uma função custo apropriada, que depende da função de ativação da última camada**

# Introdução à Função Custo



## Função custo da ativação Linear

- **Mede a diferença quadrática entre os valores reais e as previsões**
- **Como os valores podem assumir qualquer número real, a função custo mais comum é o Erro Quadrático Médio (MSE)**
  - ✓  Fácil de calcular e interpretar.
  - ✓  Penaliza erros grandes de forma mais intensa.
  - ✓  Não funciona bem para classificação!
  - ✓  Sensível a outliers, o que pode afetar o treinamento da redes

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

# Introdução à Função Custo

## Função custo da ativação Sigmoid

- **Também chamada de Entropia cruzada binária (BCE)**
- **Ideal para classificação binária, onde a saída representa a probabilidade de pertencer à classe positiva)**
  - ✓  Vantagem: Penaliza fortemente previsões erradas com alta confiança.
  - ✓  Problema: Se  $\hat{y}_i$  for muito próximo de 0 ou 1, pode levar a problemas numéricos.

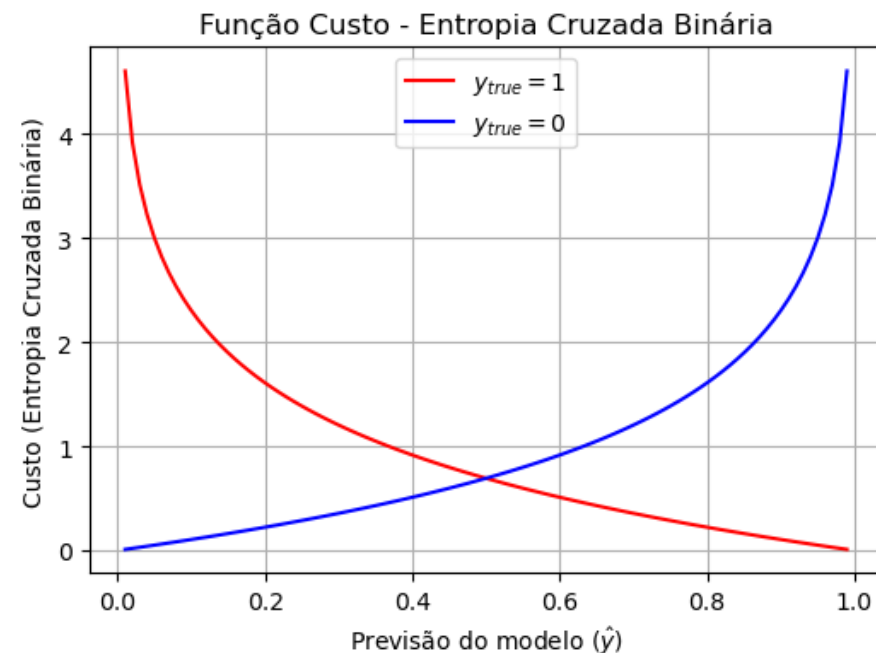
$$J(\theta) = -\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - \hat{y}_i) \log(1 - \hat{y}_i)]$$

# Introdução à Função Custo

## Função custo da ativação Sigmoid

- A ilustração mostra como o custo muda em função da previsão usando BCE.
  - ✓ **Linha Vermelha ( $y_{true}=1$ ):** O custo é alto quando  $\hat{y}$  está próximo de 0, pois o modelo está errando com alta confiança.
  - ✓ **Linha Azul ( $y_{true}=0$ ):** O custo é alto quando  $\hat{y}$  está próximo de 1, pois o modelo está errando com alta confiança.
  - ✓ Interpretação
    - ✓ A entropia cruzada penaliza mais fortemente erros de alta confiança
    - ✓ Isso força a rede neural a ajustar os pesos rapidamente, garantindo um aprendizado mais eficiente.

$$J(\theta) = -\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - \hat{y}_i) \log(1 - \hat{y}_i)]$$



# Introdução à Função Custo

## Função custo da ativação Sigmoid

### ▪ Por que MSE não é ideal para classificação?

#### ✓ Problema 1: Gradientes Pequenos na sigmoid

- ✓ Se usarmos Sigmoid como ativação na saída e MSE como função custo, os gradientes podem ser muito pequenos em algumas regiões.
- ✓ Isso pode levar a aprendizado muito lento (ineficiente), especialmente quando a saída está muito próxima de 0 ou 1

#### ✓ Problema 2: Penalização Simétrica de Erros

- ✓ O MSE trata todos os erros de maneira simétrica, enquanto entropia cruzada foca em previsões muito erradas



# Introdução à Função Custo

## Função custo da ativação Sigmoid

### ■ Por que MSE não é ideal para classificação?

- ✓ Exemplo: Suponha que temos um problema de classificação binária, onde a classe correta é 1 ( $y=1$ ). O modelo faz uma previsão de 90% de certeza de que a classe é 1 ( $\hat{y}=0.9$ ).

- ✓ Agora, calculamos os custos:

$$MSE = (y_i - \hat{y}_i)^2 = (1 - 0.9)^2 = 0.01$$

$$\begin{aligned} BCE &= -[y_i \log(\hat{y}_i) + (1 - \hat{y}_i) \log(1 - \hat{y}_i)] \\ &= -[1 \log(0.9) + (1 - 1) \log(1 - 0.9)] \\ &= -\log(0.9) = 0.105 \end{aligned}$$

- ✓ MSE dá um erro pequeno (0.01), pois trata os erros de forma simétrica. Isso significa que o modelo pode não se esforçar tanto para melhorar essa previsão.
- ✓ BCE penaliza mais o erro (0.105), incentivando ajustes maiores para o mesmo erro. Isso força a rede neural a ajustar mais seus pesos para tornar a predição ainda mais próximo de 1.

# Introdução à Função Custo

## Função custo da ativação Softmax

- **Também chamada de Entropia cruzada categórica (BCC)**

- ✓ Ela mede a diferença entre a distribuição real das classes e as previsões da rede neural

- **Ideal para classificação multiclass, onde a saída representa os valores das probabilidades de pertencer à cada classe**

$n \rightarrow$  Número de exemplos no conjunto de dados.

$C \rightarrow$  Número total de classes possíveis.

$y_{i,c} \rightarrow$  Indica se a amostra  $i$  pertence à classe  $c$  (one-hot encoding).

$\hat{y}_{i,c} \rightarrow$  Probabilidade prevista pelo modelo para a classe  $c$  na amostra  $i$ .





Se a previsão estiver correta e com alta confiança ( $\hat{y}$  próximo de 1), o custo será baixo.

Se o modelo prever com alta confiança a classe errada, o custo é alto.

$$J(\theta) = -\frac{1}{n} \sum_{i=1}^n \sum_{c=1}^C y_{i,c} \log(\hat{y}_{i,c})$$

# Introdução à Função Custo

## Função custo da ativação Softmax

- ✓  Penaliza previsões incertas ou erradas com alta confiança.
- ✓  Incentiva a rede a prever com mais certeza a classe correta.
- ✓  Funciona muito bem quando combinada com a Softmax, pois gera probabilidades normalizadas para cada classe.
- ✓  Se um modelo prevê 99% de certeza para a classe errada, o custo será extremamente alto.
  - ✓ Isso pode levar a ajustes bruscos nos pesos durante o treinamento, tornando a convergência instável.
  - ✓ Uma estratégia para mitigar isso é usar label smoothing, que adiciona uma pequena regularização nas probabilidades para evitar certeza extrema.
- ✓ Exemplo: modelo prever a distribuição  $(0.7, 0.2, 0.1)$  onde  $y=1$ , a penalização será pequena, pois a previsão está correta e confiante. Se a previsão for  $(0.1, 0.1, 0.8)$ , a penalização será muito maior.

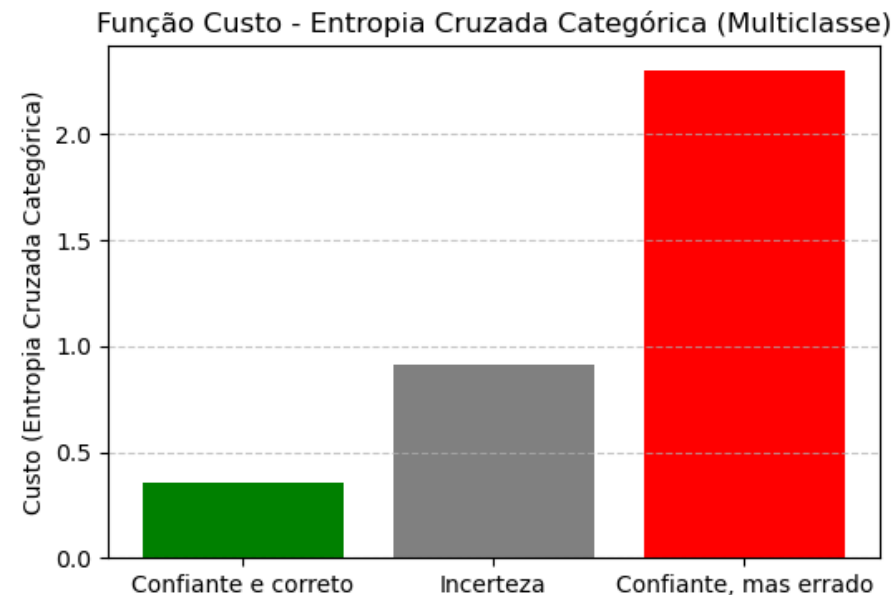
# Introdução à Função Custo

## Função custo da ativação Softmax

- ✓ Exemplo: modelo prever a distribuição  $(0.7, 0.2, 0.1)$  onde  $y=1$ , a penalização será pequena, pois a previsão está correta e confiante. Se a previsão for  $(0.1, 0.1, 0.8)$ , a penalização será muito maior.

```
y_pred_multi = np.array([  
    [0.7, 0.2, 0.1], # Modelo confiante e correto (classe 1)  
    [0.4, 0.3, 0.3], # Modelo incerto  
    [0.1, 0.1, 0.8], # Modelo confiante, mas errado)  
)
```

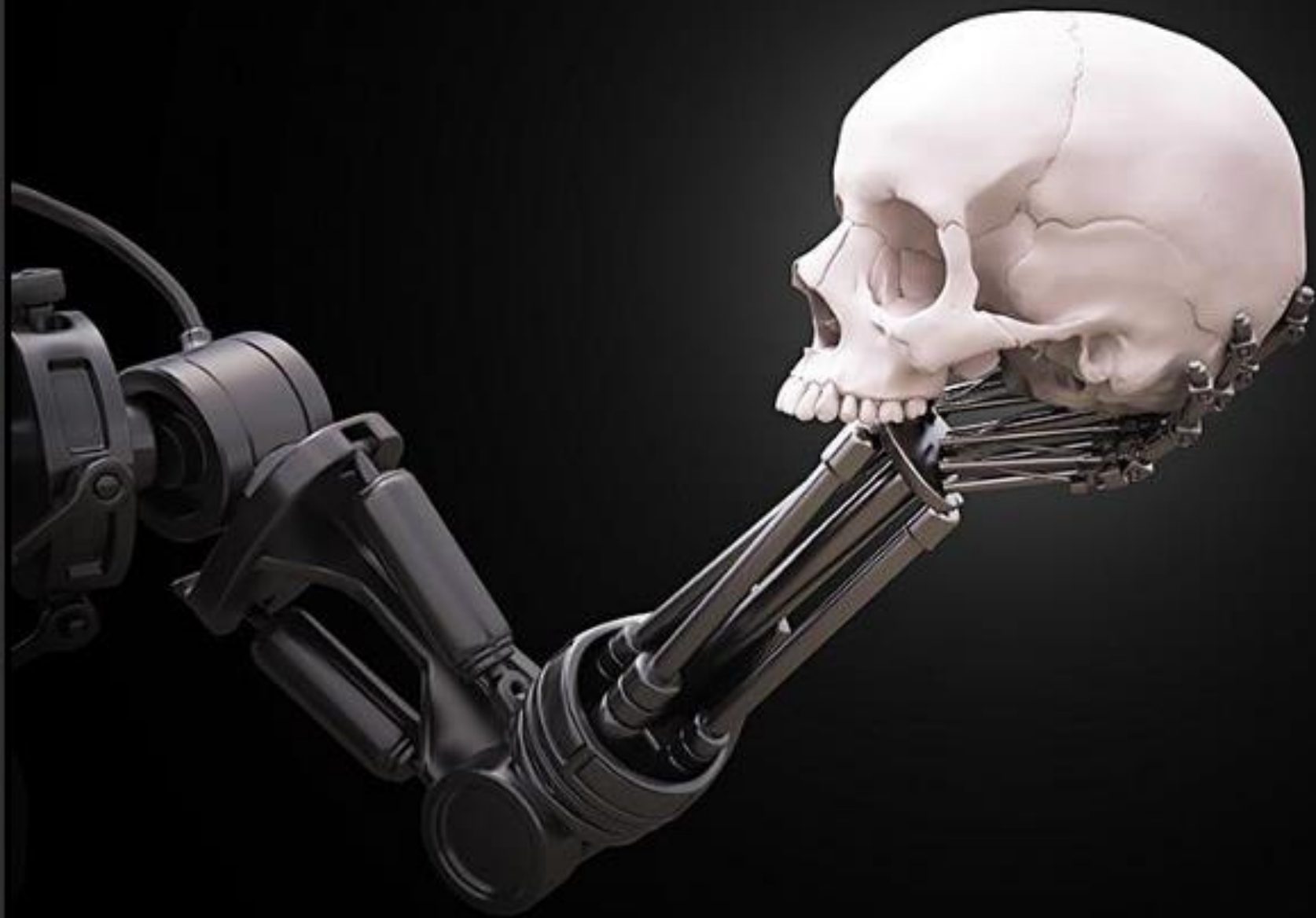
```
✓ y_true_multi = np.array([  
    [1, 0, 0], # Classe 1 correta  
    [1, 0, 0], # Classe 1 correta  
    [1, 0, 0], # Classe 1 correta  
)
```





Dúvidas?

---



Até a próxima...



Apresentador

Thales Levi Azevedo Valente

E-mail:

[thales.l.a.valente@gmail.com](mailto:thales.l.a.valente@gmail.com)