

# Treinamento de Machine Learning e Deep Learning

## Do Básico ao Avançado

Salomão Machado Mafalda<sup>1</sup>

<sup>1</sup>Universidade Federal do Acre  
PAVIC

2023



# Agenda

- 1 Perceptron
- 2 Adaline
- 3 Neurônio Sigmoidal
- 4 Funções de Ativação
- 5 Backpropagation
- 6 Se Tornando Expert em Gradientes



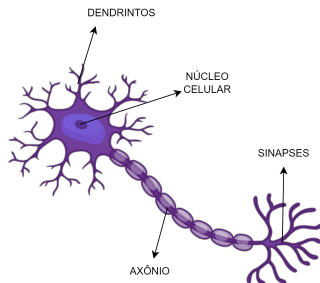


Figure: Neurônio humano

# Perceptron

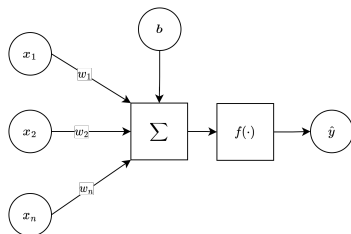
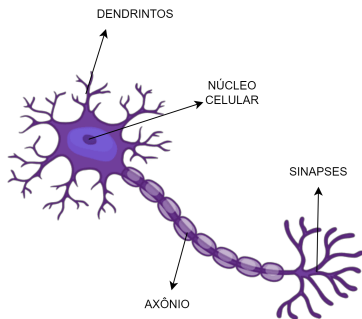
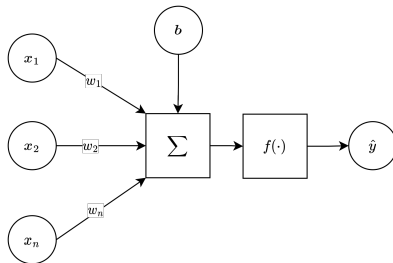


Figure: Neurônio Artificial

# Perceptron



(a) Neurônio Humano



(b) Neurônio Artificial



# Perceptron

- Modelo mais básico de NN
- Um neurônio
- N entradas, Uma saída  $\hat{y}$

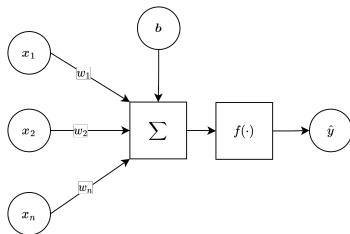


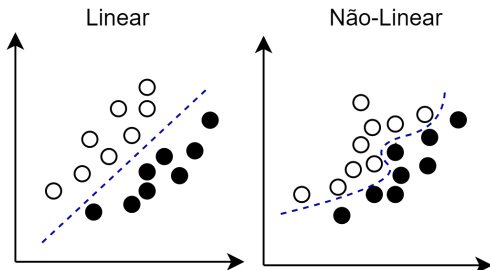
Figure: Neurônio Artificial

$$\hat{y} = f\left(\sum_i w_i x_i + b\right)$$



# Perceptron

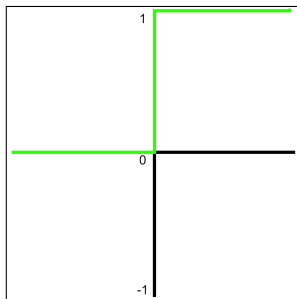
- Modelo mais básico de NN
- Um neurônio
- N entradas, Uma saída  $\hat{y}$
- Classificador binário linear
- Pode ser usado para Regressão
- Perceptron Rule
- Aprendizado Online
  - Atualiza os pesos por amostra



## Função de ativação do perceptron

$$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

- 0 se for negativo
- 1 se maior ou igual a 0





# Perceptron

## Perceptron Rule

O perceptron atualiza seus pesos utilizando a perceptron rule, não com o gradiente

Atualização dos pesos:

$$w_i = w_i + \lambda * (y_i - \hat{y}_i) * x_i$$

Atualização do *bias*:

$$b_i = b_i + \lambda * (y_i - \hat{y}_i)$$

## Observação importante

“Quando a diferença  $y_i - \hat{y}_i$  for 0 então não ocorrerá a atualização dos pesos”

## Ponto de partida diferente

Com diferentes pontos de partida, o algoritmo encontra quase a mesma solução, embora com diferentes taxas de convergência.

▶ Caso 01

▶ Caso 02

## Learning Rate - Taxa de aprendizagem

- $LR = 0.01$  a velocidade de convergência é muito lenta. Quando o cálculo se torna complicado, uma taxa de aprendizado muito baixa afetará a velocidade do algoritmo, mesmo nunca atingindo o destino.
- $LR = 0.5$ , o algoritmo se aproxima do alvo muito rapidamente após várias iterações. No entanto, o algoritmo falha ao convergir porque o salto é muito grande, fazendo com que ele fique parado no destino.

▶ Caso 01

▶ Caso 02

Vamos ver na prática

Vamos praticar utilizando o notebook 00\_perceptron



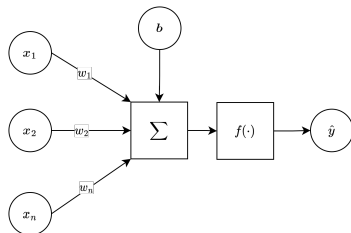


Figure: Neurônio Artificial

- Modelo mais básico de NN
- Um neurônio
- N entradas, Uma saída  $\hat{y}$
- Classificador binário linear
- Pode ser usado para Regressão
- Sabe o quanto 'errou'
- Aplica-se o gradiente descendente
- Aprendizado Online

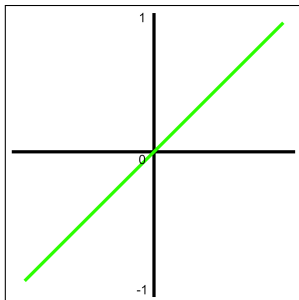
$$\hat{y} = f\left(\sum_i w_i x_i + b\right)$$



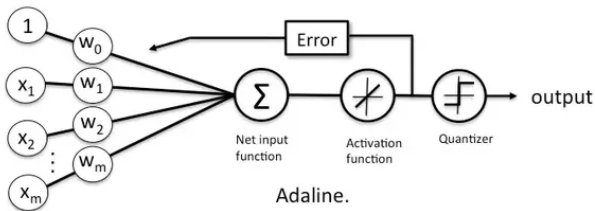
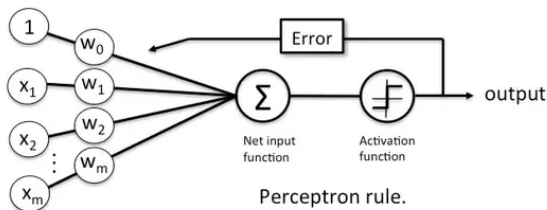
## Função de ativação do Adaline

$$f(x) = x$$

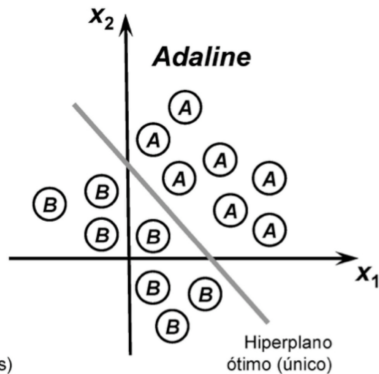
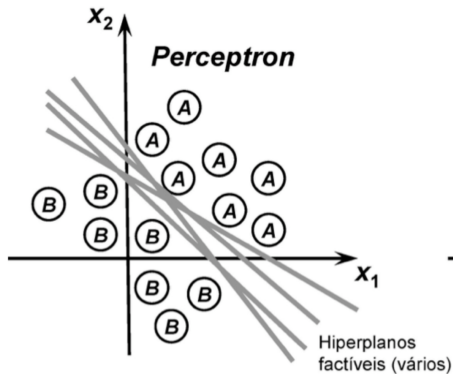
- Possibilita o cálculo da derivada



# Adaline vs Perceptron



# Adaline vs Perceptron





## Como atualizar os pesos do Adaline

$$w_i = w_i - \lambda * (y_i - \hat{y}_i) * x_i$$

O erro predito será a saída predita menos a saída desejada multiplicados pela entrada ( $x$ )

$$J(w) = \frac{1}{2} \sum_i^N (y_i - \hat{y}_i)^2$$

$$\frac{\partial J}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_i^N (y_i - \hat{y}_i)^2 = \frac{1}{2} \sum_i^N \frac{\partial}{\partial w_i} (y_i - \hat{y}_i)^2$$

$$= \sum_i^N (y_i - \hat{y}_i) \frac{\partial}{\partial w_i} (y_i - \hat{y}_i) = \sum_i^N (y_i - \hat{y}_i) (x_i) \rightarrow \frac{\partial J}{\partial \vec{w}} = -(\vec{y} - \hat{\vec{y}}) \vec{x}$$

Vamos ver na prática

Vamos praticar utilizando o notebook 01\_adaline



# Neurônio Sigmoide

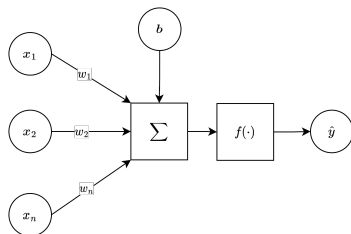


Figure: Neurônio Artificial

# Neurônio Sigmoides

- Modelo mais básico de NN
- Um neurônio
- N entradas, Uma saída  $\hat{y}$
- Custo: Entropia Cruzada
- Classificação binária não-linear
- Pequenas alterações nos parâmetros geram pequenas alterações nas saídas
- Sabe o quanto 'errou'
- Aplica-se o gradiente descendente

$$\hat{y} = f\left(\sum_i w_i x_i + b\right)$$

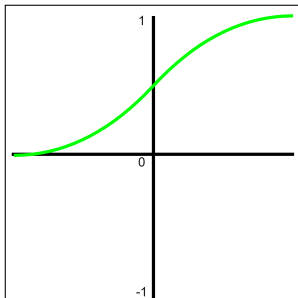


# Neurônio Sigmoid

## Função de ativação do Neurônio Sigmoid

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

- Possibilita o cálculo da derivada em todos pontos
- Aplicado em problemas de regressão logística



## Entropia Cruzada

$p_j$  é o valor a ser predito e  $t_j$  é o valor predito

$$L = -\frac{1}{N} \left[ \sum_{j=1}^N [t_j \log(p_j) + (1 - t_j) \log(1 - p_j)] \right]$$



# Neurônio Sigmoid

Vamos tomar:

$p_j$	$t_j$	Erro	$L$
0	0	$0 - 0 = 0$	0
0	1	$0 - 1 = -1$	$\infty$
1	0	$1 - 0 = 1$	$\infty$
1	1	$1 - 1 = 0$	0

## Entropia Cruzada

Para entrada 0,0 e saída predita 0 e a saída desejada for 0

$$L = -\frac{1}{N} \left[ \sum_{j=1}^N [0 \log(0) + (1 - 0) \log(1 - 0)] \right]$$



# Neurônio Sigmoid

Vamos tomar:

$p_j$	$t_j$	Erro	$L$
0	0	$0 - 0 = 0$	0
0	1	$0 - 1 = -1$	$\infty$
1	0	$1 - 0 = 1$	$\infty$
1	1	$1 - 1 = 0$	0

## Entropia Cruzada

Para entrada 0, 1 e saída predita 0 e a saída desejada for 1

$$L = -\frac{1}{N} \left[ \sum_{j=1}^N [0 \log(1) + (1 - 0) \log(1 - 1)] \right]$$





# Neurônio Sigmoid

Vamos tomar:

$p_j$	$t_j$	Erro	$L$
0	0	$0 - 0 = 0$	0
0	1	$0 - 1 = -1$	$\infty$
1	0	$1 - 0 = 1$	$\infty$
1	1	$1 - 1 = 0$	0

## Entropia Cruzada

Para entrada 1, 0 e saída predita 1 e a saída desejada for 0

$$L = -\frac{1}{N} \left[ \sum_{j=1}^N [1 \log(0) + (1 - 1) \log(1 - 0)] \right]$$



# Neurônio Sigmoid

Vamos tomar:

$p_j$	$t_j$	Erro	$L$
0	0	$0 - 0 = 0$	0
0	1	$0 - 1 = -1$	$\infty$
1	0	$1 - 0 = 1$	$\infty$
1	1	$1 - 1 = 0$	0

## Entropia Cruzada

Para entrada 1, 1 e saída predita 1 e a saída desejada for 1

$$L = -\frac{1}{N} \left[ \sum_{j=1}^N [1 \log(1) + (1 - 1) \log(1 - 1)] \right]$$



# Neurônio Sigmoid

Vamos ver na prática

Vamos praticar utilizando o notebook 02\_neuronio\_sigmoid



# Funções de Ativação

- Localizada a saída de cada neurônio
- Usada para mapear entradas em novas saídas
- Pode alterar o range ex:  $[-100 \ 100]$  para  $[1 \ 0]$

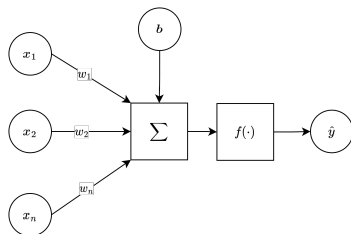


Figure: Neurônio Artificial

$$\hat{y} = f\left(\sum_i w_i x_i + b\right)$$



## Linear

- $y \in [-\infty, +\infty]$
- Função de ativação simples
- Comumente usada em regressão
- Baixa complexidade
- Baixo poder de aprendizagem

Função:

$$f(x) = x$$

Derivada:

$$\frac{\partial y}{\partial x} = 1$$



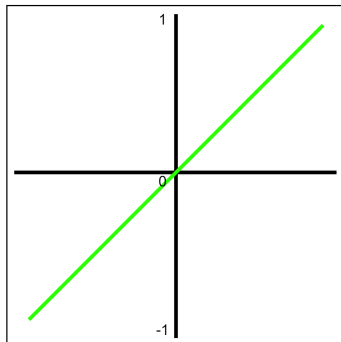
## Atenção

Note este caso: Porque construir modelos apenas com Lineares?

input: “10  $\rightarrow$  100  $\rightarrow$  200  $\rightarrow$  10”

pesos: “10  $\rightarrow$  2  $\rightarrow$  00.5 = 10  $\times$  10 = 10”

Podemos substituir todos pesos por um só



## Sigmoid

- $y \in [0, +1]$
- Regressão Logística
- Geralmente interpretada como probabilidade
- Saída não centrada em 0
- Satura os gradientes
- Não indicada para camadas ocultas
- Converge lentamente

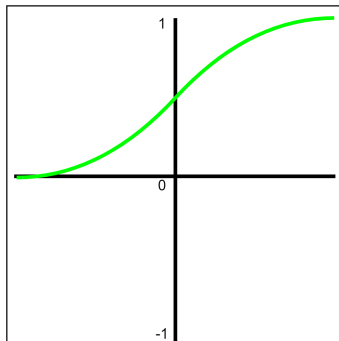
Função:

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

Derivada:

$$\frac{\partial y}{\partial x} = y(1 - y)$$

# Funções de Ativação





## Tanh

- $y \in [-1, +1]$
- Uma versão da Sigmoid
- Saída centrada em 0
- Satura os gradientes. um pouco menos que a Sigmoid
- Converge lentamente

Função:

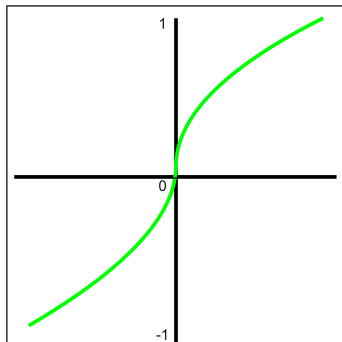
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Derivada:

$$\frac{\partial y}{\partial x} = 1 - y^2$$



# Funções de Ativação



## Relu

- $y \in [0, +\infty]$
- Não tem derivada para valores  $< 0$
- Simples e Eficiente
- Evita a saturação dos gradientes
- Converge mais rápido
- Usada nas camadas escondidas
- Ela mata neurônio

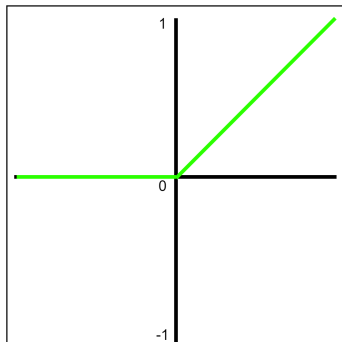
Função:

$$f(x) = \max(0, x)$$

Derivada:

$$\frac{\partial y}{\partial x} = \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x > 0 \end{cases}$$

# Funções de Ativação



## Leaky Relu

- $y \in [-\infty, +\infty]$
- Tem derivada para valores  $< 0$
- Simples e Eficiente
- Evita a saturação dos gradientes
- Converge mais rápido
- Usada nas camadas escondidas
- Diminui as mortes de neurônios

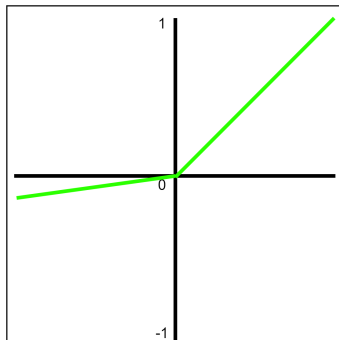
Função:

$$f(x) = \begin{cases} \alpha(e^x - 1), & x \leq 0 \\ x, & x > 0 \end{cases}$$









Derivada:

$$\frac{\partial y}{\partial x} = \begin{cases} \alpha, & x \leq 0 \\ 1, & x > 0 \end{cases}$$

# Funções de Ativação



# Funções de Ativação

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$



## Qual função de ativação usar?

- Evitar Sigmoid nas camadas ocultas, boa na saída
- Tanh usada em modelos generativos
- Relu Muito boa nas camadas ocultas
- Linear não usar em camadas escondidas
- Leaky Relu raramente usadas





## Softmax

- $y \in [0, 1]$  e  $\sum_y = 1$
- Aplicada em dois ou mais neurônios. Pega saída e converte
- A saída é uma confiança
- Nunca nas camadas ocultas
- Multiclasses
- Saída One-hot Encode

Função:

$$S_i = \frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}}$$

Assim para cada  $k$ :  $P^k = S_i^{[k]}$  Derivada:

$$\frac{\partial S}{\partial y} = p^k * (1 - p^k)$$

## Softmax

### Example

$[0.1, 1.3, 2.5] \rightarrow [0.07, 0.22, 0.72]$

$$\frac{e^{0.1}}{e^{0.1} + e^{0.3} + e^{2.5}}$$

Função:

$$S_i = \frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}}$$

Assim para cada  $k$ :  $P^k = S_i^{[k]}$  Derivada:

$$\frac{\partial y}{\partial x} = \begin{cases} \alpha, & x \leq 0 \\ 1, & x > 0 \end{cases}$$

Vamos ver na prática

Vamos praticar utilizando o notebook 03\_funções de ativação



# Backpropagation

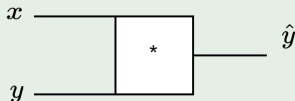
Para que usamos o *backpropagation*?

Utilizamos o Backpropagation no treinamento das redes neurais

## Example

Vamos tomar uma função simples de multiplicação:

$$f(x, y) = x * y$$

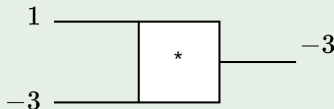


# Backpropagation

## Example

Vamos tomar uma função simples de multiplicação:

$$f(x, y) = x * y$$

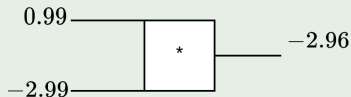


Com que força alterar as entradas desse circuito para de maneira “leve” alteremos a saída?

## Example

Vamos tomar uma função simples de multiplicação:

$$f(x, y) = x * y$$



# Highlighting text

$$W = np.random.randn(5, 10) \quad (1)$$

$$X = np.random.randn(3, 10) \quad (2)$$

$$Y = X.dot(W^T) \quad (3)$$

$$f(x) \quad (4)$$

In this slide, some important text will be **highlighted** because it's important. Please, don't abuse it.

## Remark

Sample text

## Important theorem

Sample text in red box

## Examples

Sample text in green box. The title of the block is "Examples"