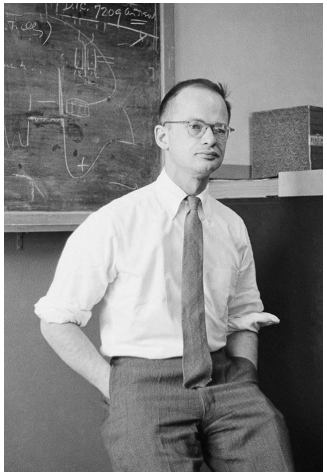
Redes Neurais Artificiais

Redes Neurais Artificiais



Devido ao caráter “tudo ou nada” da atividade nervosa, eventos neurais e as relações entre eles podem ser tratados por meio da lógica proposicional. Descobriu-se que o comportamento de cada rede pode ser descrito nesses termos, com a adição de meios lógicos mais complicados para redes contendo círculos; e que para qualquer expressão lógica que satisfaça certas condições, pode-se encontrar uma rede se comportando da maneira que ela descreve. É mostrado que muitas escolhas particulares entre possíveis suposições neurofisiológicas são equivalentes, no sentido de que para cada rede se comportando sob uma suposição, existe outra rede que se comporta sob a outra e dá os mesmos resultados, embora talvez não no mesmo tempo. Várias aplicações do cálculo são discutidas.

Walter Pitts

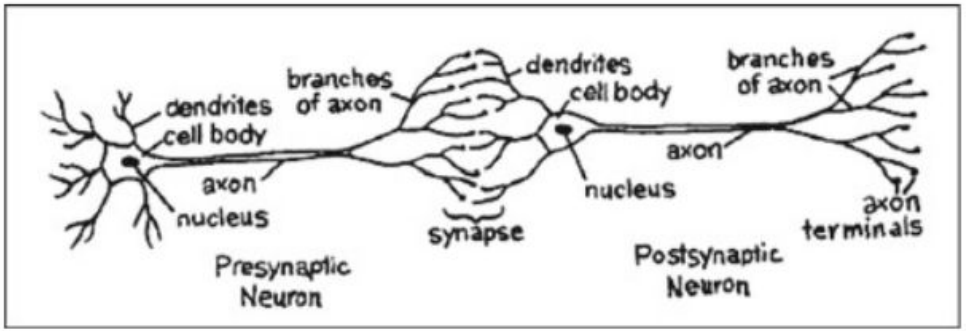
1923 - 1969

Warren McCulloch

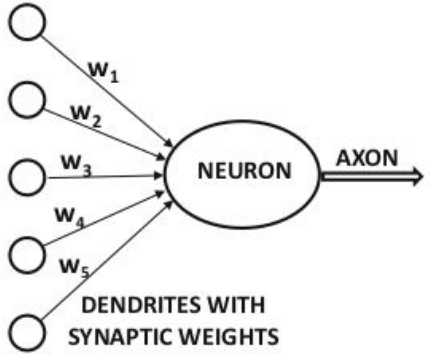
1898 - 1969

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY* (1943)

Modelo Teórico

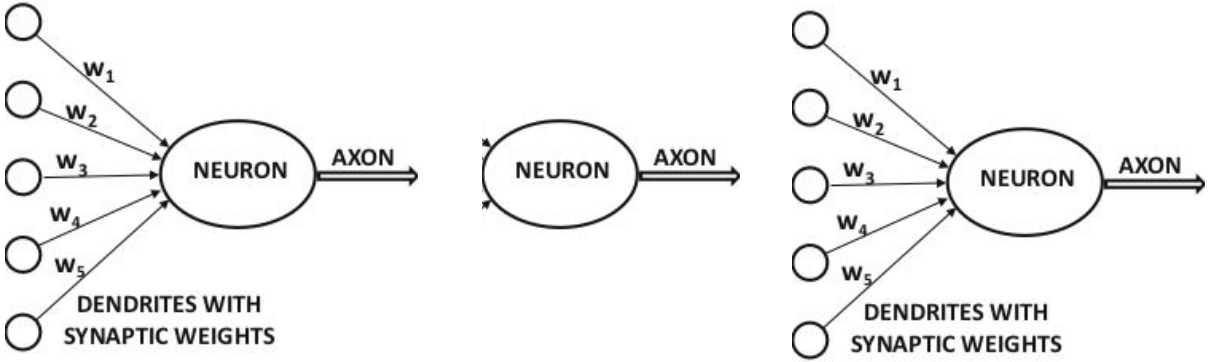


(a) Biological neural network



(b) Artificial neural network

Modelo Teórico



No modelo computacional, as unidades (neurônios) são interconectadas por ‘pesos’, que possuem o objetivo de simular a força das conexões sinápticas. Assim, cada entrada de um neurônio sofre uma transformação de escala que afetará a função de ativação.

Treinamento

Em uma rede neural uma função modifica a entrada, propagando os valores calculados dos neurônios de entrada para o(s) neurônio(s) de saída, utilizando os pesos como parâmetros intermediários.

O aprendizado ocorre pela alteração dos pesos que conectam os neurônios. Assim como estímulos externos são necessários para o aprendizado em organismos biológicos, o estímulo externo em redes neurais artificiais é fornecido pelos dados de treinamento contendo exemplos de pares de entrada-saída da função a ser aprendida.

Treinamento

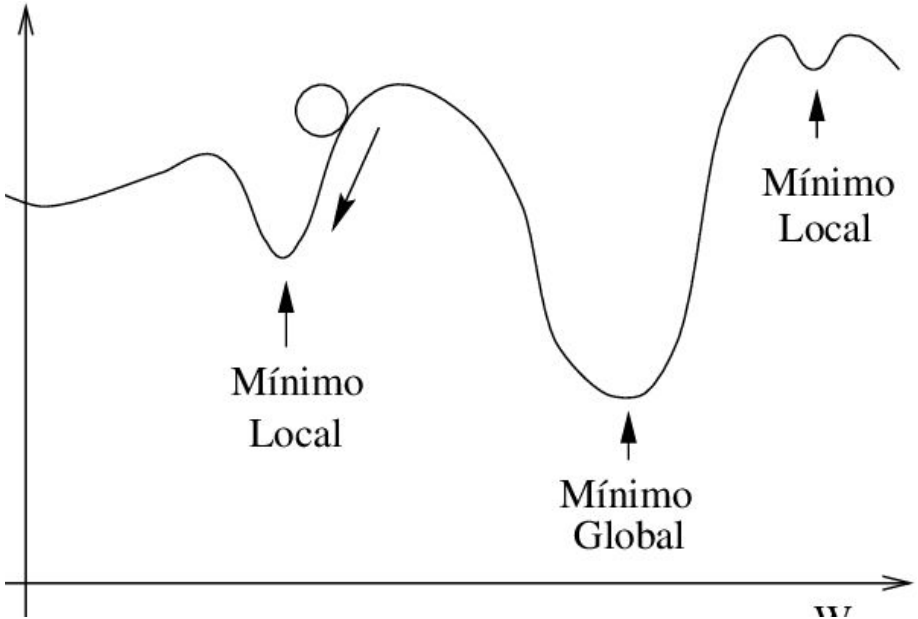
Para o treinamento, são utilizados dois grupos de dados, o primeiro consiste nas entradas e o segundo nas saídas esperadas.

Os dados de treinamento fornecem feedback sobre a correção dos pesos na rede neural, dependendo do erro (diferença entre o esperado e obtido) serão definidos os novos pesos para os neurônios.

O objetivo da alteração dos pesos é modificar a função computada para tornar as previsões mais corretas em iterações futuras. Portanto, os pesos são alterados, utilizando operações matemáticas, para reduzir o erro de cálculo naquele exemplo.

Treinamento

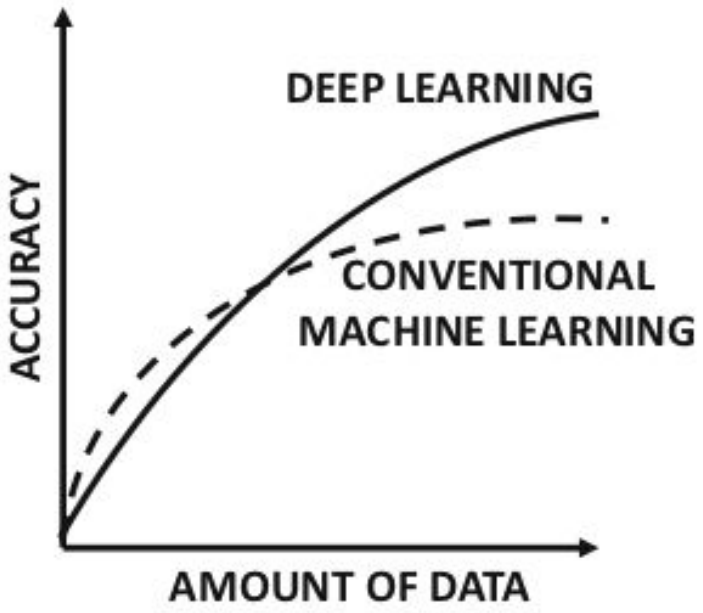
O objetivo do treinamento é definir os pesos de forma a reduzir o erro.



Machine Learning x Deep Learning

Machine Learning é um ramo da inteligência artificial (IA) que ensina computadores a aprenderem com dados e a tomar decisões ou fazer previsões sem serem explicitamente programados para cada tarefa (Ex. Regressão Linear).

Deep Learning é um subtipo avançado de Machine Learning que usa redes neurais artificiais (inspiradas no cérebro humano) para aprender padrões em dados, especialmente em problemas mais complexos.



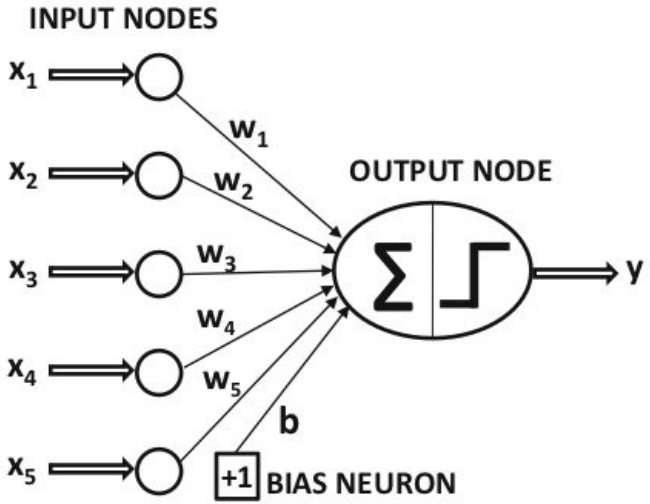
Arquitetura de uma RNA

A arquitetura de uma RNA é definida pelos seguintes aspectos:

- Quantidade de entradas e saídas esperadas.
- Quantidade de camadas da rede (neurônios interconectados).
- Função de ativação.

Arquitetura Perceptron

A RNA mais simples existente é a perceptron de uma camada. Na perceptron (*single-layer)* um conjunto de entradas é mapeado diretamente para uma saída usando uma variação generalizada de uma função linear.



Definições

$(\overline{X}, y)$ Instância de treinamento X -> y

$\overline{X} = [x_1, \ldots x_d]$

Conjunto de variáveis de entrada (*feature)*

$y \in \{-1, +1\}$

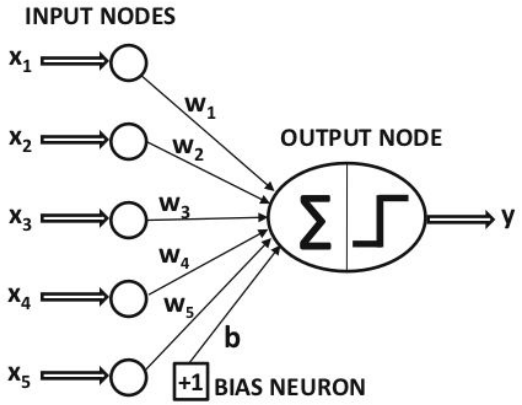
Conjunto de observações (Binária).

$\overline{W} = [w_1 \ldots w_d]$

Conjunto de pesos para um nó de saída.

$\overline{W} \cdot \overline{X} = \sum_{i=1}^{d} w_i x_i$

Função linear para cálculo da saída.



$\hat{y} = \text{sign}\{\overline{W} \cdot \overline{X}\} = \text{sign}\{\sum_{j=1}^{d} w_j x_j\}$

Função para cálculo da predição (variável dependente). A função *sign* mapeia um valor real em +1 ou -1. Para uma saída binária.

$E(\overline{X}) = y - \hat{y}$

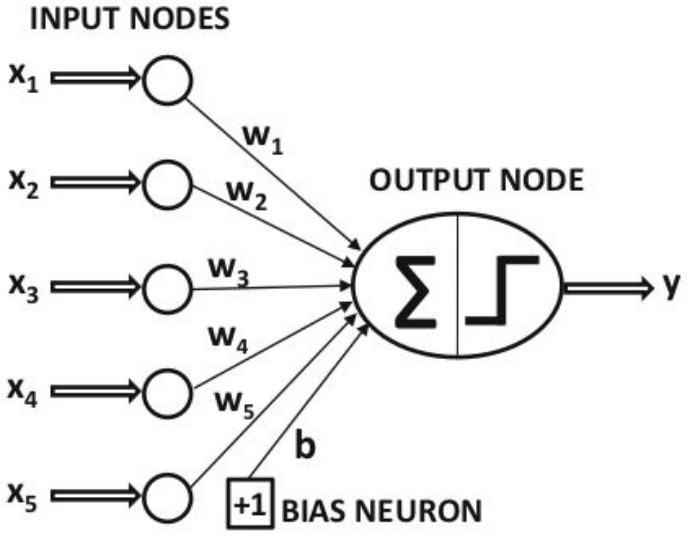
Cálculo do erro

Viés (*Bias)*

Em muitos cenários, há uma parte invariante da previsão, que é chamada de viés (*bias*).

O bias é um parâmetro adicional em redes neurais que ajuda o modelo a ajustar melhor suas previsões, permitindo que ele desloque a função de ativação para a esquerda ou direita, melhorando o aprendizado.

Analogia simples: Pense em uma rede neural como uma pessoa tentando acertar um alvo com dardos.



$$\hat{y} = \text{sign}\{\overline{W} \cdot \overline{X} + b\} = \text{sign}\{\sum_{j=1}^{d} w_j x_j + b\}$$

Os pesos (weights) definem como ela ajusta o lançamento (força, ângulo).
O bias é como um ajuste extra (ex.: dar um passo para frente ou para trás) para que os dardos caiam mais perto do centro.
Sem o bias, a rede neural só pode aprender padrões que passam exatamente pela origem (0,0), o que é muito limitante.

Função de perda (*loss function)*

A Loss Function (Função de Perda ou Função de Custo) é uma medida de erro que diz à rede neural quão longe suas previsões estão dos valores reais.

$$MAE\,(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

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

Função de perda (*loss function)*

Cross Entropy

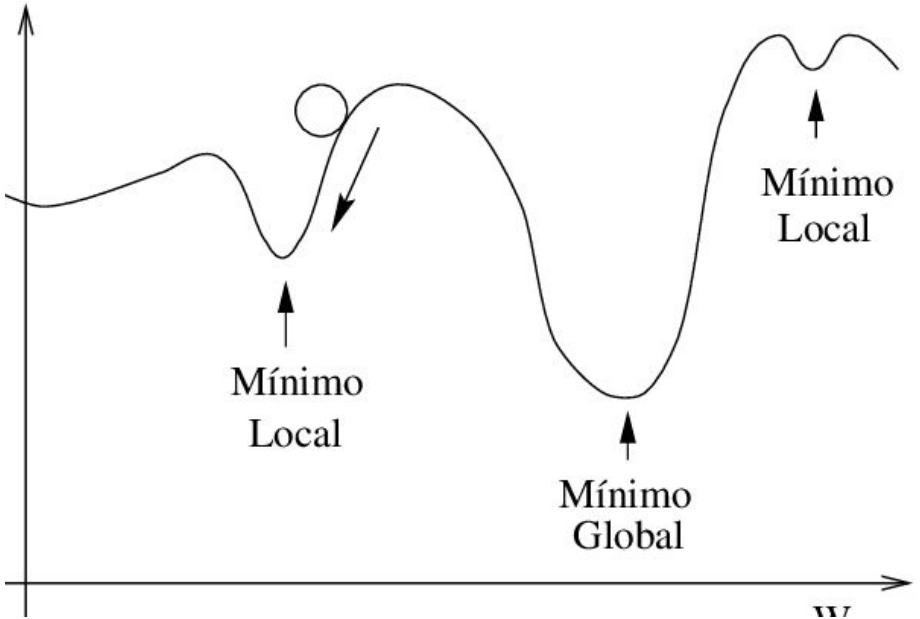
$$\text{Loss} = -[y_{\text{real}} \cdot \log(y_{\text{predito}}) + (1 - y_{\text{real}}) \cdot \log(1 - y_{\text{predito}})]$$

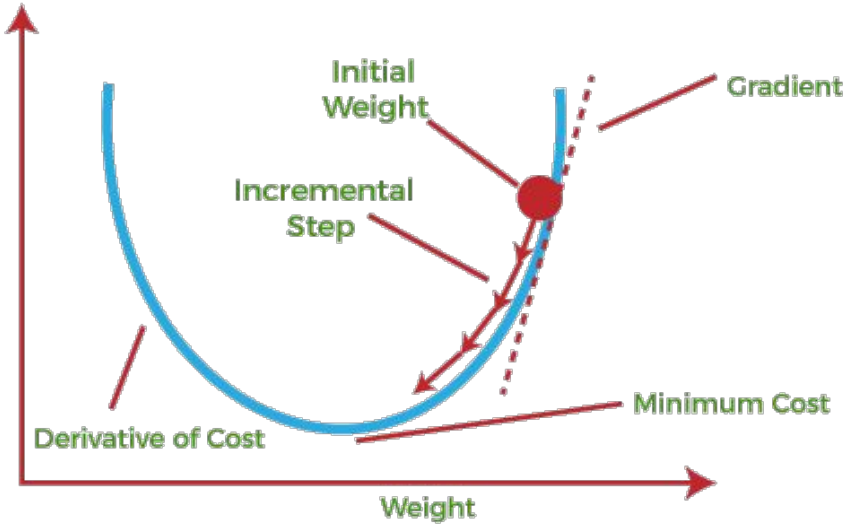
Mede quão distante a probabilidade prevista está do rótulo verdadeiro.

Função de perda (*loss function)*

Por que a Loss Function é Importante?

- Direciona o aprendizado: Sem ela, a rede não saberia como melhorar.
- Ajuda a detectar problemas:
  - Loss muito alta? Modelo está com underfitting (não aprendeu).
  - Loss oscilando? Taxa de aprendizado (learning rate) inadequada.
- Diferentes tarefas, diferentes losses:
  - Regressão → MSE, MAE
  - Classificação → Cross-Entropy
  - Geração de imagens → Perceptual Loss





https://ml4a.github.io/ml4a/how_neural_networks_are_trained/

Exemplo de treinamento

Dados

i	x1	x2	Y
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

Função de Ativação

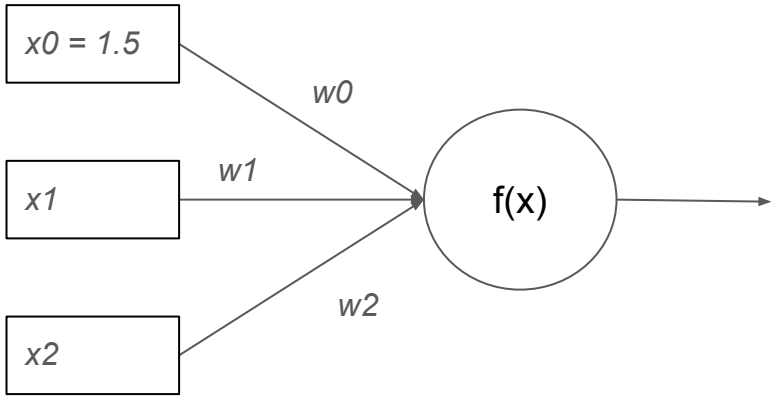
$$f(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

Pesos Iniciais

$w_0 = 0, w_1 = 0, w_2 = 0$

Bias

x0 = 1.5



$$w_i^{\text{novo}} = w_i^{\text{antigo}} + \eta \cdot (y_{\text{desejado}} - y_{\text{atual}}) \cdot x_i$$

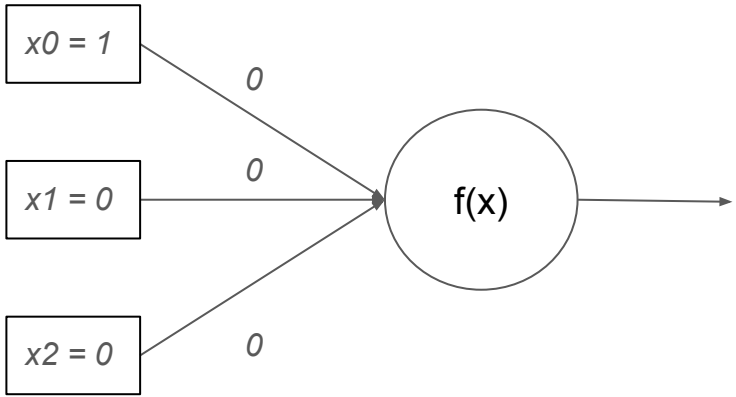
Onde:

- w_i: peso associado à entrada x_i
- η: taxa de aprendizado (learning rate), geralmente $0 < \eta \leq 1$
- y_{desejado}: saída esperada (target)
- y_{atual}: saída produzida pelo Perceptron
- x_i: valor da entrada correspondente ao peso w_i

Learning Rate

n = 0.1

Época 1 - i = 0 - [x0 = 1 | x1 = 0 | x2 = 0]



net = $w_0 \cdot x_0 + w_1 \cdot x_1 + w_2 \cdot x_2 = 0$

saída = f(0) = **1**

erro = desejado - saída = 0 - 1 = **-1**

atualização dos pesos:

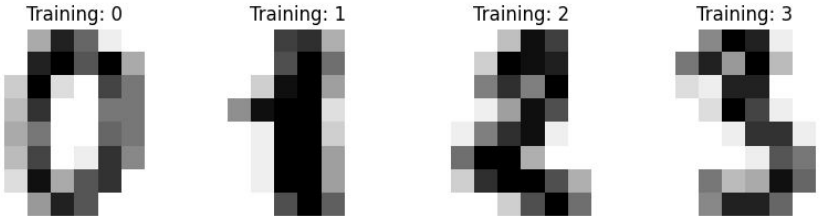
- $w_0 = 0 + 0.1 \cdot (-1) \cdot 1 =$ **-0.1**
- $w_1 = 0 + 0.1 \cdot (-1) \cdot 0 =$ **0**
- $w_2 = 0 + 0.1 \cdot (-1) \cdot 0 =$ **0**

Código da aula

https://github.com/diego91964/ai-uniube/blob/main/code/06%20-%20rna.ipynb

Exemplos de Problemas

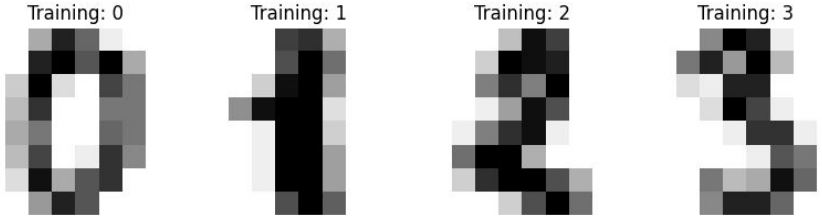
Considere um *dataset* onde uma imagem foi desenhada a mão e a RNA precisa identificar qual número foi desenhado.

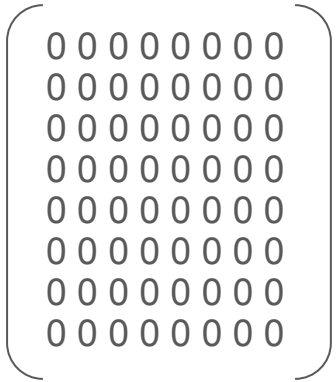


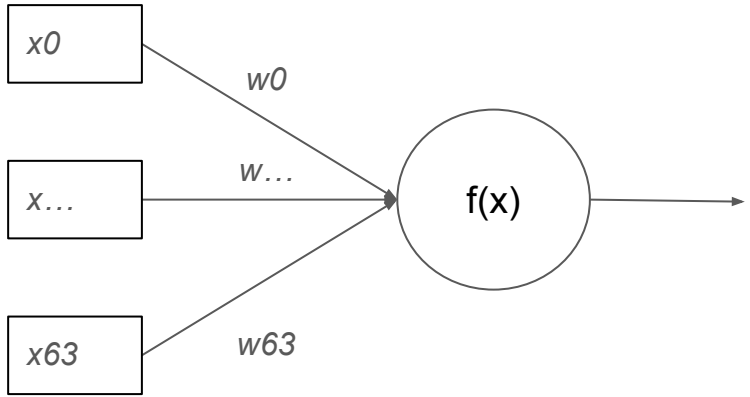
$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

https://scikit-learn.org/1.5/auto_examples/datasets/plot_digits_last_image.html

Dígitos

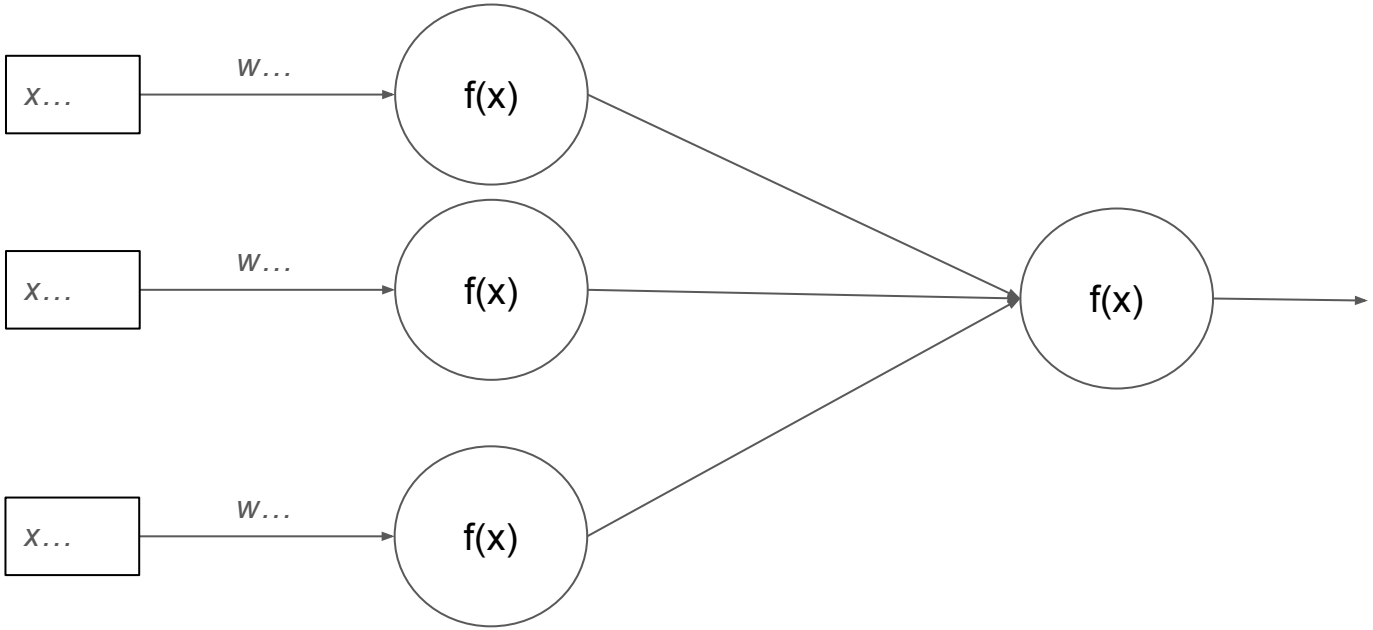






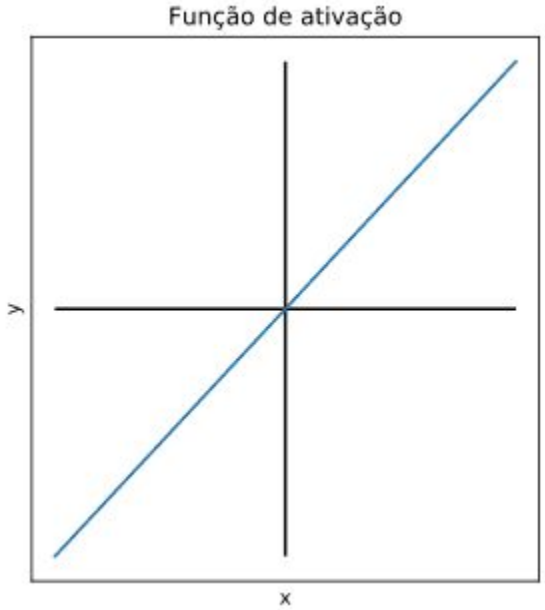
https://scikit-learn.org/1.5/auto_examples/datasets/plot_digits_last_image.html

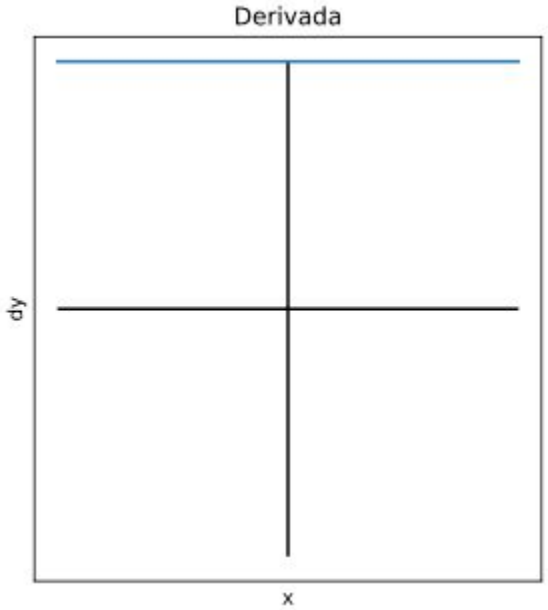
Redes Neurais Multicamadas



Funções de Ativação - Linear

A função linear apenas aplica um fator de multiplicação ao valor que recebe.

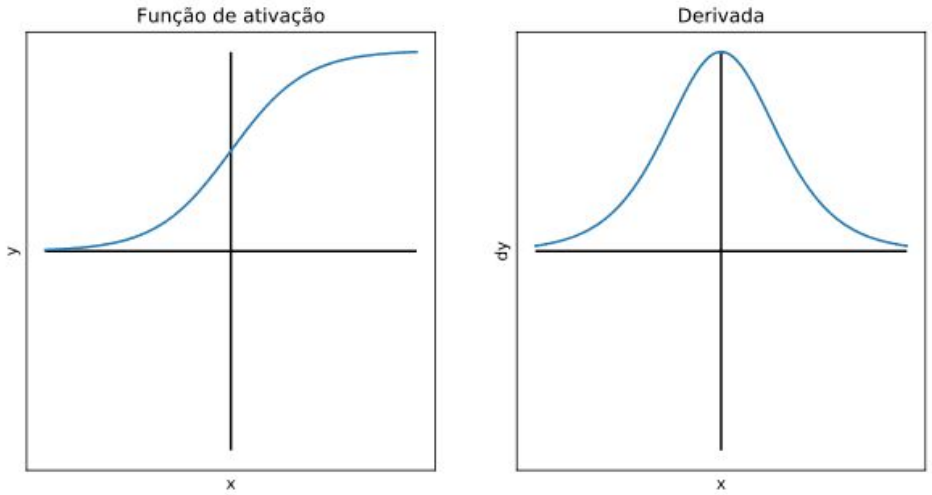




Limitada para relação complexa entre os dados.

Funções de Ativação - Sigmoide

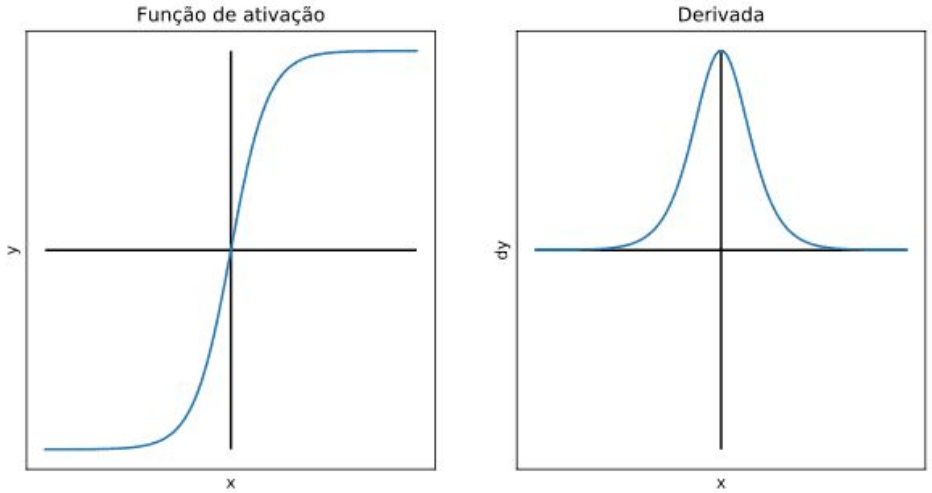
A função logística ou sigmoide produz valores no intervalo [0, 1].



Na camada de saída, a função sigmoide é útil para produzir probabilidades em problemas de classificação binária, já que seus resultados, na faixa de [0, 1], podem ser interpretados como a probabilidade de determinada instância pertencer ou não a determinada classe.

Funções de Ativação - Tangente hiperbólica (tanh)

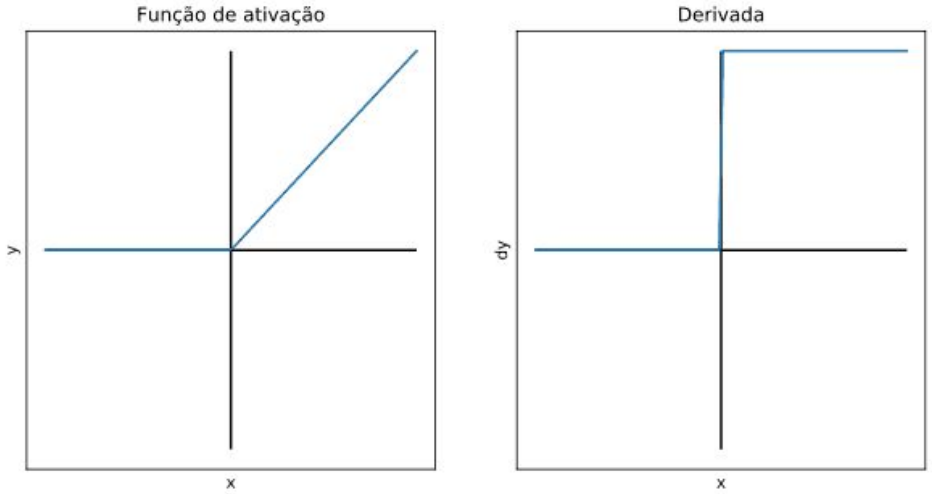
Produz resultados no intervalo [-1, 1].



A tangente hiperbólica tem as mesmas vantagens e ainda resolve um dos problemas da função sigmoide, sendo centrada em zero. Entretanto, sua derivada também converge a zero, e mais rapidamente.

Funções de Ativação - ReLU

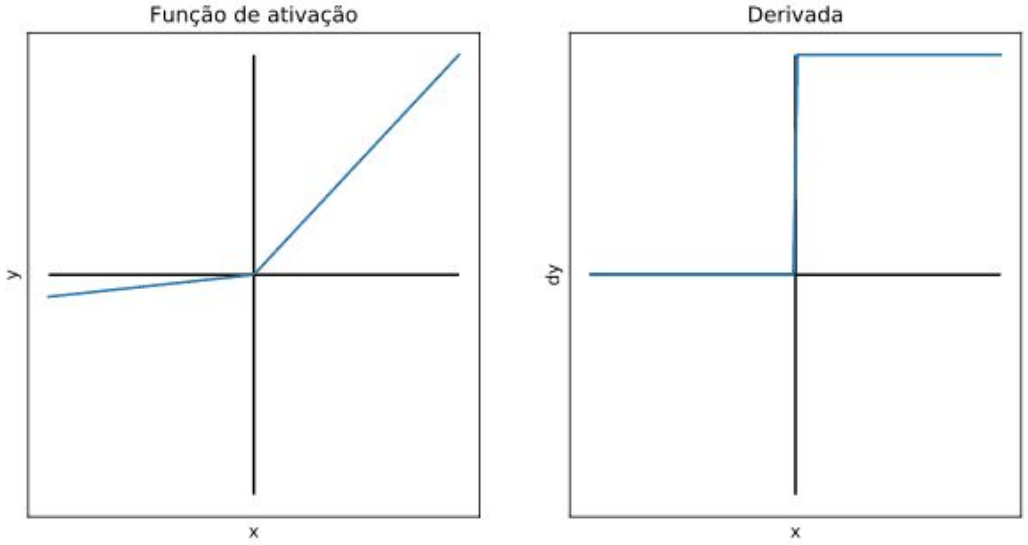
ReLU é uma abreviação para rectified linear unit, ou unidade linear retificada. Ela produz resultados no intervalo [0, ∞ [.



Fonte: https://iaexpert.academy/2020/05/25/funcoes-de-ativacao-definicao-caracteristicas-e-quando-usar-cada-uma

Funções de Ativação - ReLU

Esta é uma modificação da função ReLU, que ao invés de zerar os valores negativos, aplica a eles um fator de divisão (ajustado pelo desenvolvedor), tornando-os ao invés muito pequenos (próximos de zero). Leaky significa “vazando”, que é a ideia por trás dessa função. Esse comportamento tende a resolver alguns dos problemas da função ReLU relacionados aos valores zerados.



Otimizadores

Um otimizador é o algoritmo responsável por ajustar os pesos da rede neural com o objetivo de minimizar a função de perda. Ele usa os gradientes calculados via ***backpropagation*** para atualizar os pesos.

São exemplos de otimizadores:

- SGD (Stochastic Gradient Descent).
- Adam (Adaptive Moment Estimation).
- RMSprop
- Adagrad