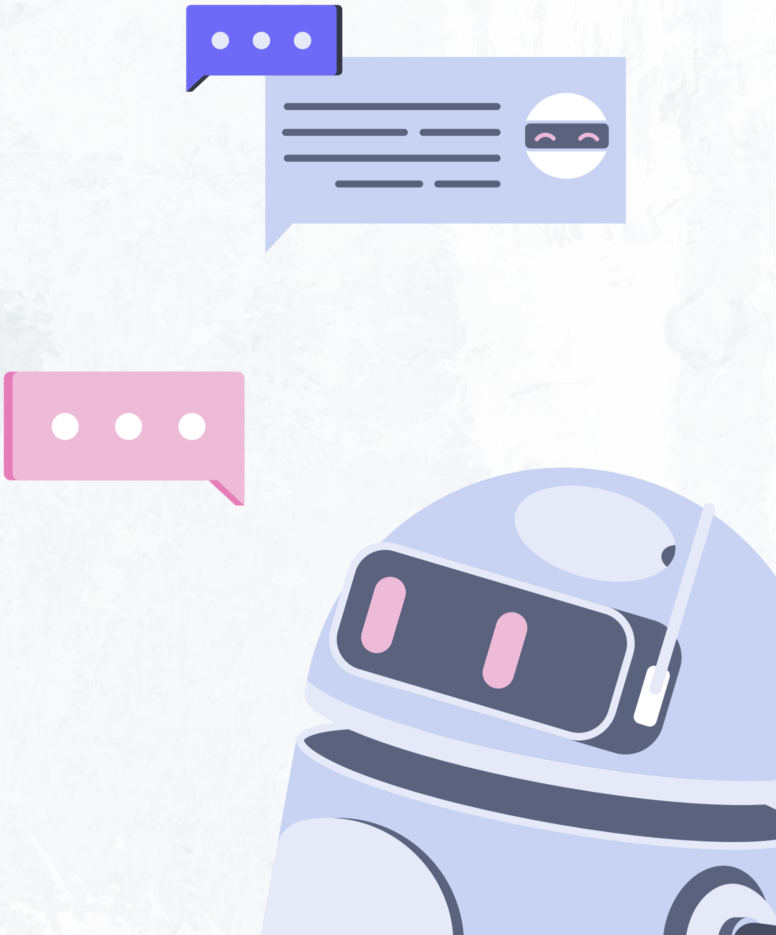
Setrem

# Introdução à Inteligência Artificial

João Paulo Aires

(IA)



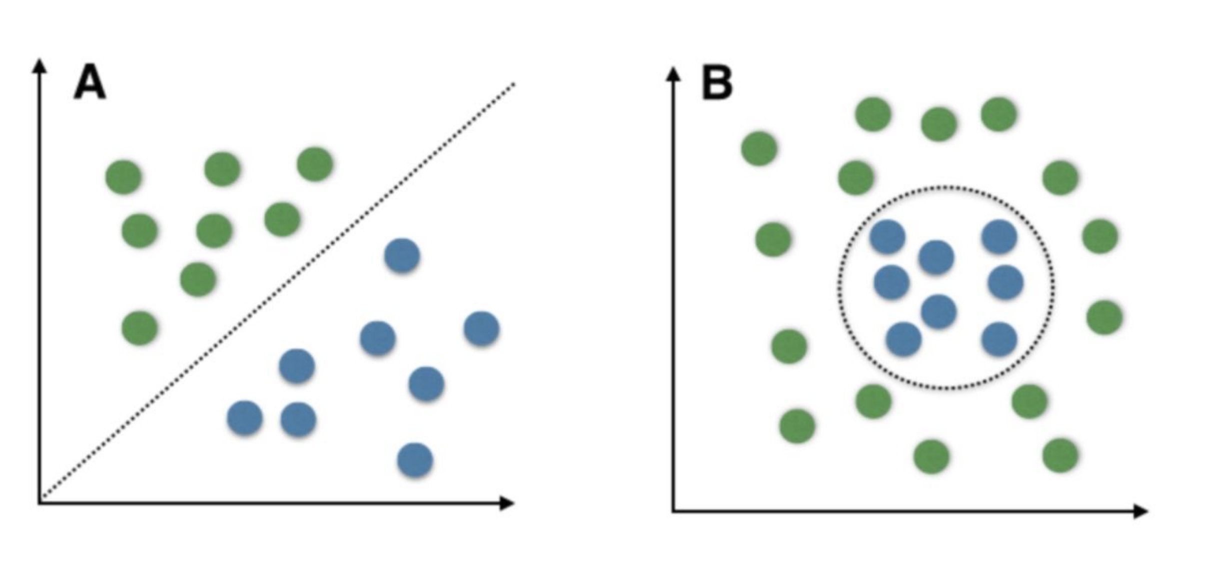
# Índice

01 →

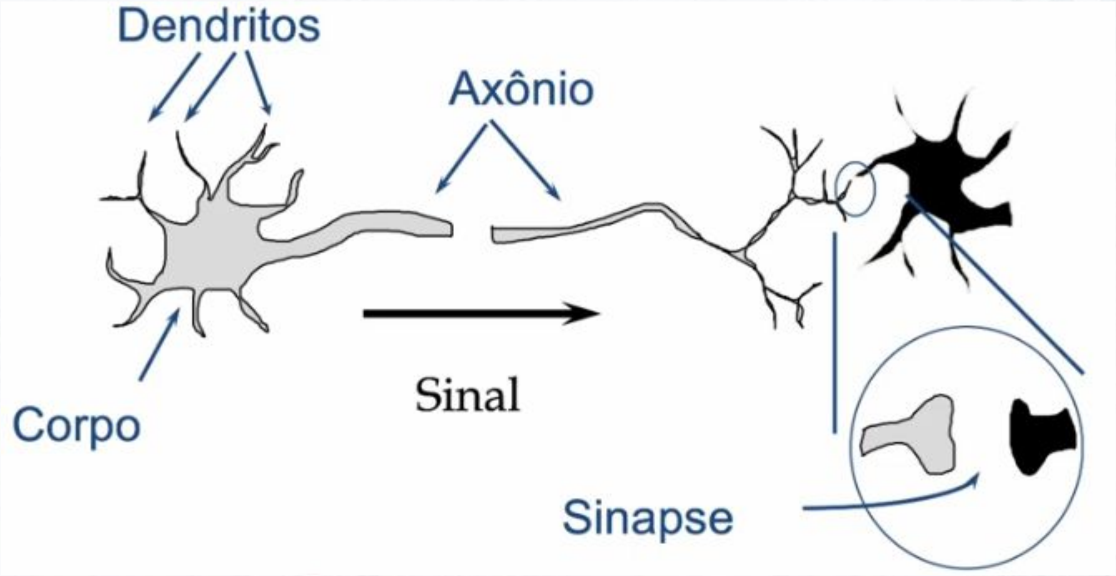
# Redes Neurais Artificiais

(IA)

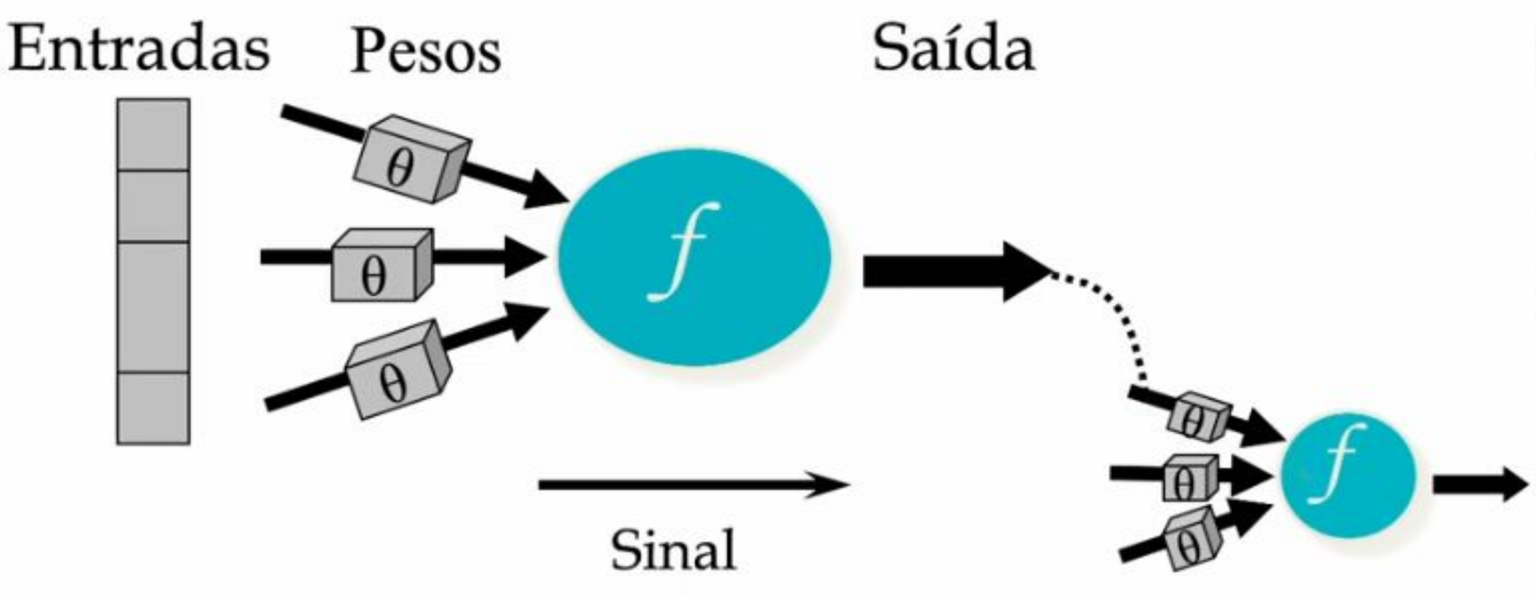
# Problemas Não-Lineares



# Redes Neurais: A inspiração biológica



# Neurônio Artificial



# Conceitos Básicos

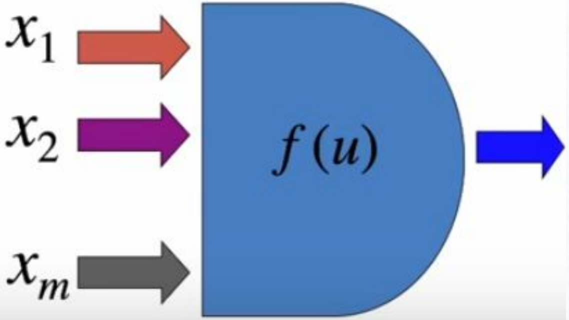
- Aspectos de uma Rede Neural Artificial (RNA)
    - Arquitetura
        - Unidades de Processamento (neurônios)
        - Conexões
        - Topologia
    - Aprendizado
        - Paradigmas
        - Algoritmos

# Unidades de Processamento

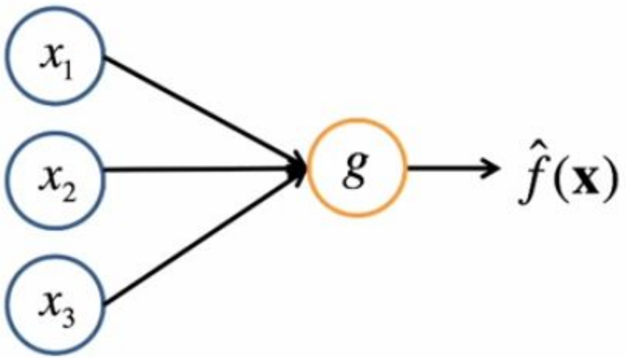
- Funcionamento
  - Recebe entradas a partir de um conjunto de unidades A
  - Aplica função sobre as entradas
  - Envia resultado para conjunto de unidades B

- Entrada total:

$$u = \theta_0 + \sum_{i=1}^{m} \theta_i x_i$$

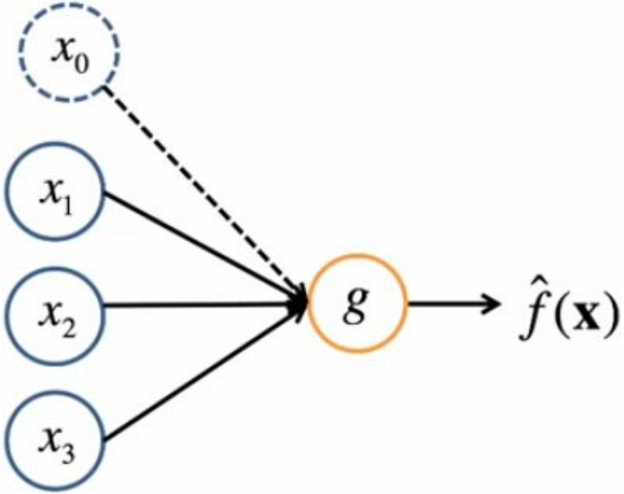


# Unidade Logística



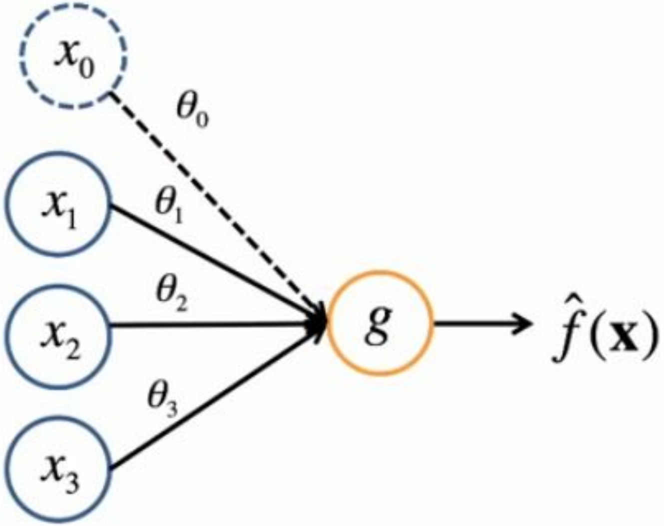

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

# Unidade Logística



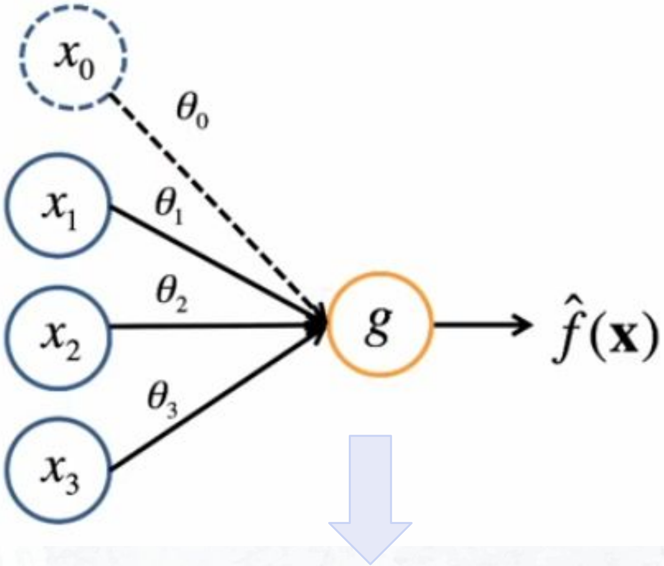

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

# Unidade Logística




$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

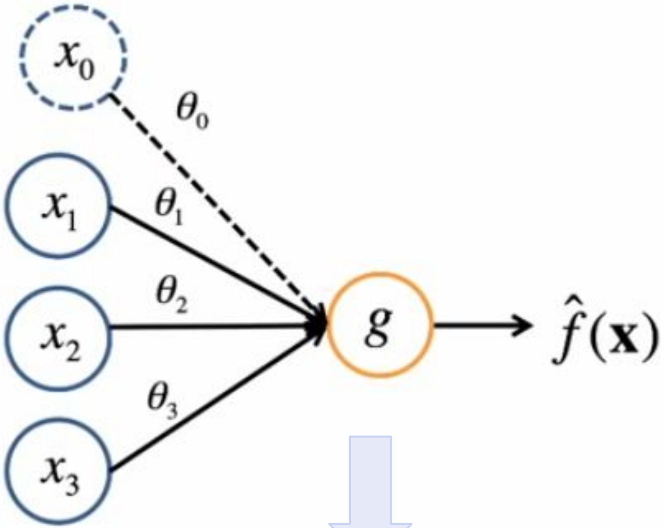
# Unidade Logística




$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

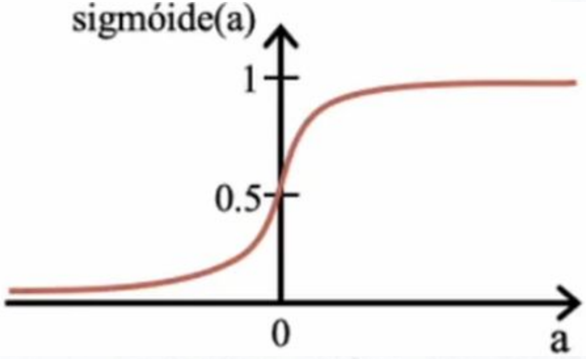
$$g(\Theta^T \mathbf{x}) = \frac{1}{1 + e^{-(\Theta^T \mathbf{x})}}$$

# Unidade Logística




$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$g(\Theta^T \mathbf{x}) = \frac{1}{1 + e^{-(\Theta^T \mathbf{x})}}$$

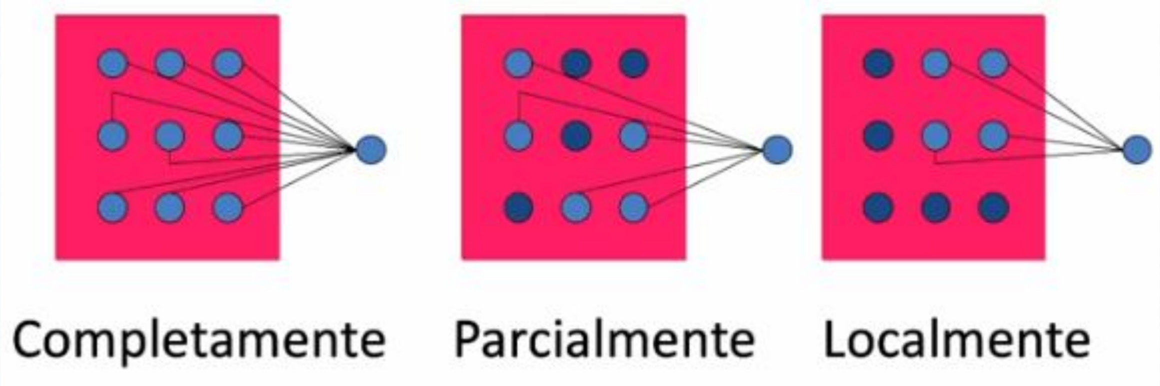


# Conexões

- Definem como neurônios estão interligados
- Codificam o conhecimento da rede
- Tipos de conexões
    - Excitatória: pesos positivos
    - Inibitória: pesos negativos

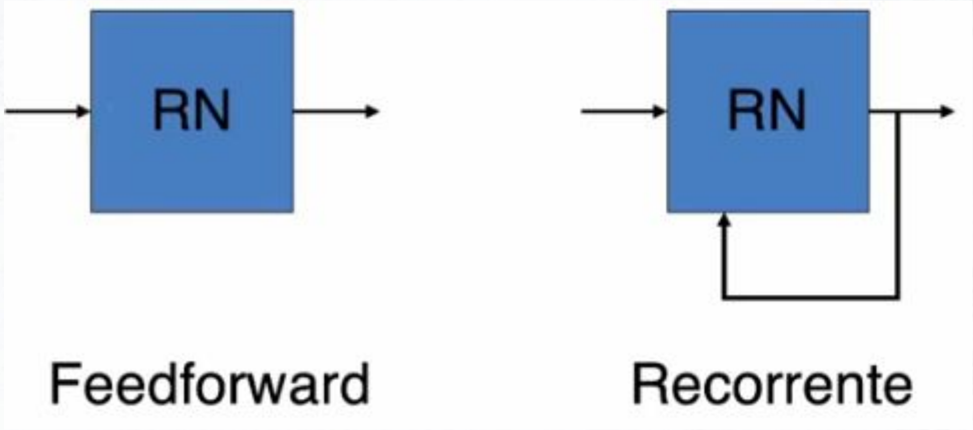
# Topologia

- Número de camadas
  - Uma única camada (e.g., Perceptron, Adaline)
  - Múltiplas camadas (e.g., MLP, RBF)



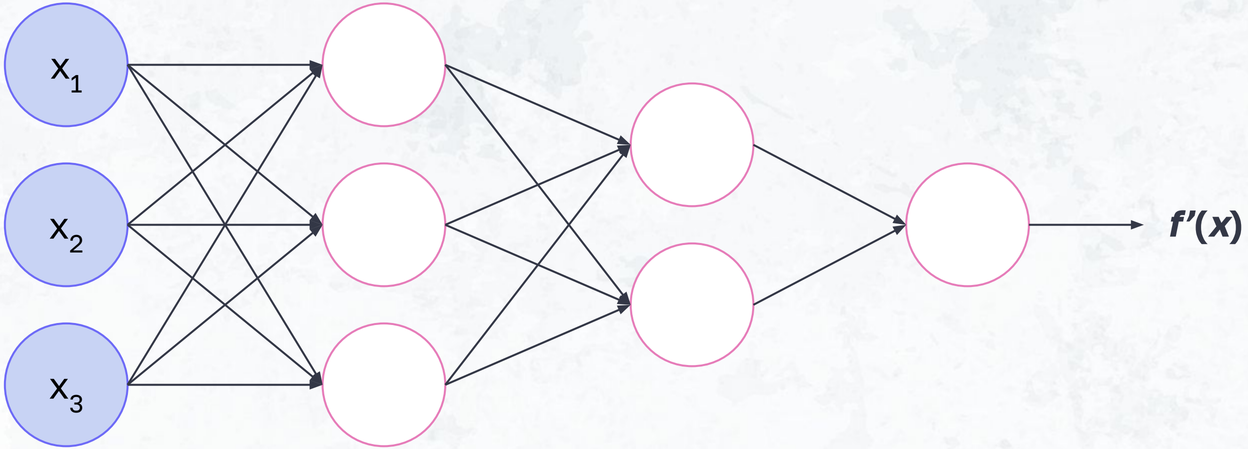
# Topologia

- Arranjo das conexões



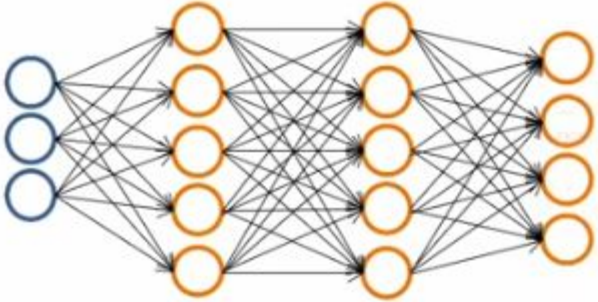
# Exemplo de Topologia

- Rede *feedforward* com múltiplas camadas completamente conectadas



# Problemas com mais de duas classes

Pedestre Carro Moto Caminhão



$$\hat{f}(\mathbf{x}) \in \Re^4$$

Objetivo: $\hat{f}(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ para pedestres $\quad \hat{f}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ para carro

# Aprendizado de Redes Neurais

- **Paradigmas de Aprendizado**
  - Relacionamento da RNA com ambiente externo
  - Principais tipos:
    - Supervisionado
    - Não supervisionado
    - Reforço

# Aprendizado de Redes Neurais

- **Algoritmos de Aprendizado**
  - Correção de erro
  - Hebbiano
  - Competitivo
  - Termodinâmico (Boltzmann)

# Aprendizado de Redes Neurais

- **Atenção!**
  - Assim como regressão linear/logística, redes neurais trabalaham apenas com valores numéricos de atributos
    - **Converta dados categóricos** em numéricos!
    - Utilize alguma estratégia para **imputar valores ausentes**!

# Perceptron

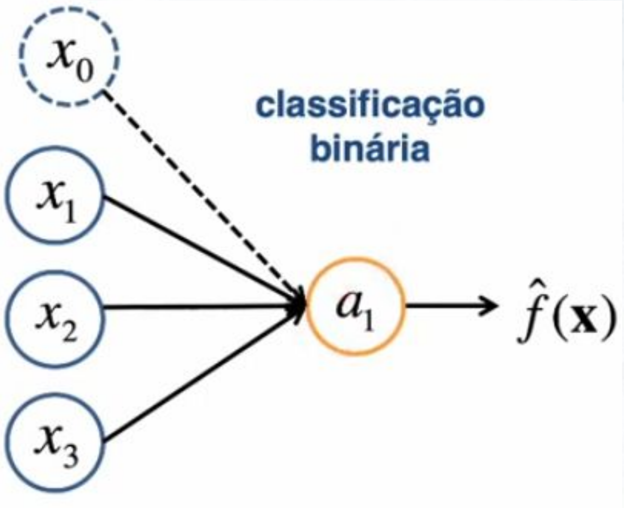
- Desenvolvido por Rosenblatt (1958)
- Utiliza modelo de neurônio de McCulloch-Pitts (1943)
    - Primeiros a formular **matematicamente** neurônios
- Paradigma de Aprendizado
    - **Supervisionado**
- Tipo de aprendizado
    - **Correção de erro**

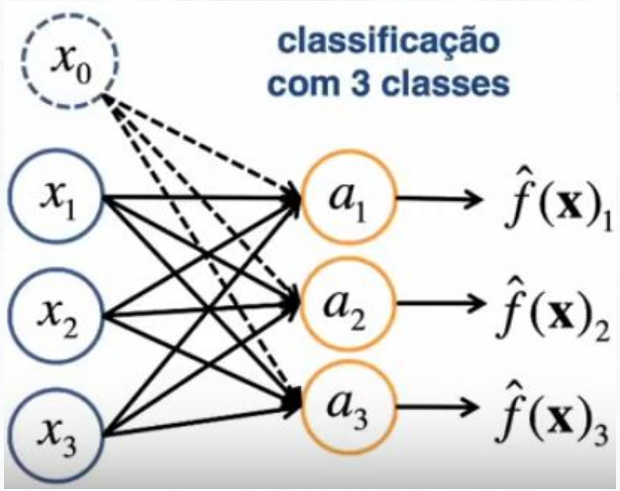
# Perceptron

- Modelo de rede **extremamente simples**
- Apenas uma camada de neurônios
- Saída da rede: {-1, +1}

# Perceptron

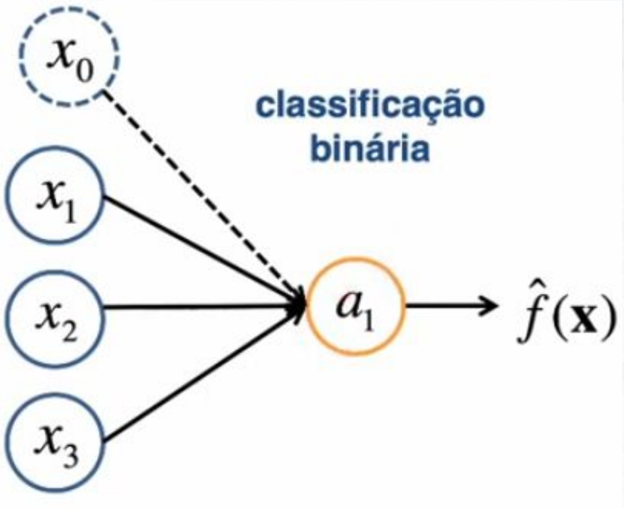
- Modelo de rede **extremamente simples**
- Apenas uma camada de neurônios
- Saída da rede: {-1, +1}

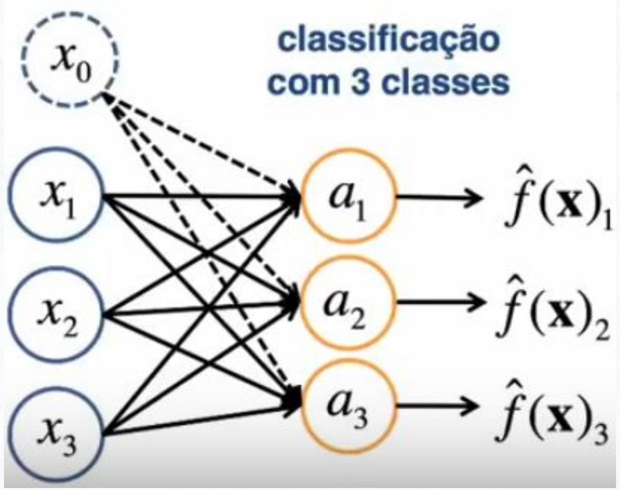


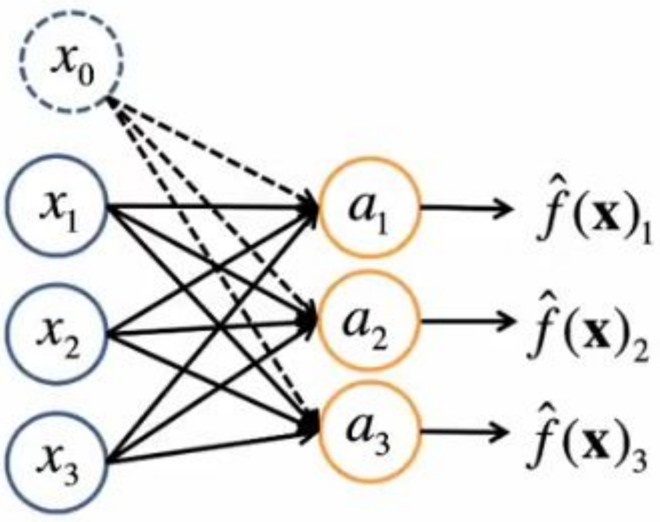
# Perceptron

- Modelo de rede **extremamente simples**
- Apenas uma camada de neurônios
- Saída da rede: {-1, +1}



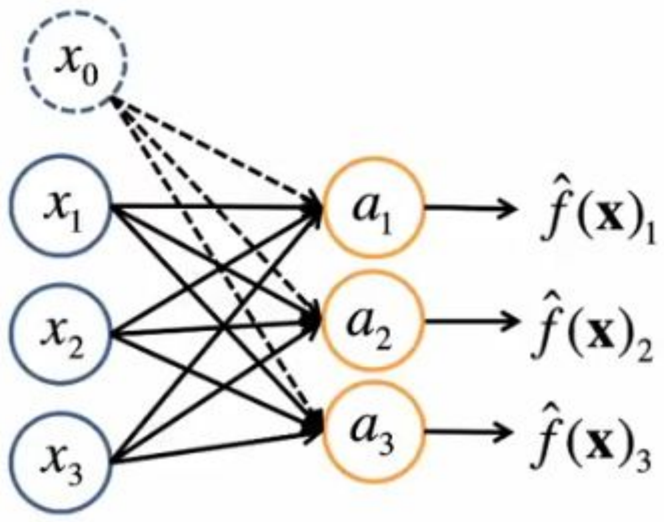



# Perceptron



$x_0$
$x_1$
$x_2$
$x_3$
$a_1$
$a_2$
$a_3$
$\hat{f}(\mathbf{x})_1$
$\hat{f}(\mathbf{x})_2$
$\hat{f}(\mathbf{x})_3$

# Perceptron

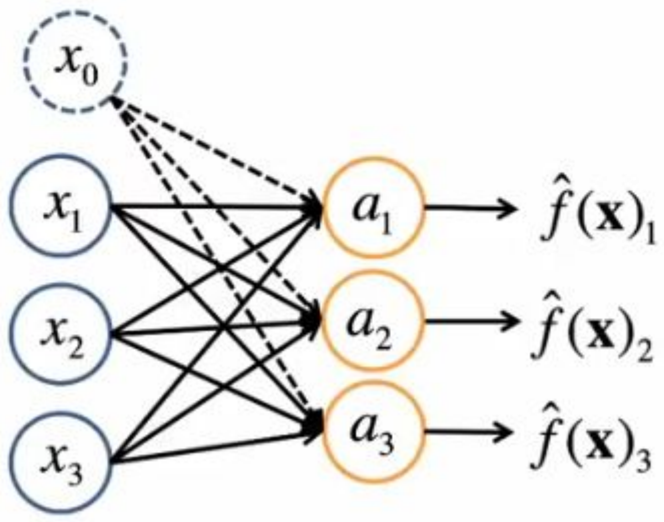



$$a_1 = g(\Theta_1^T \mathbf{x}) = g(\theta_{10}x_0 + \theta_{11}x_1 + \theta_{12}x_2 + \theta_{13}x_3)$$

$$a_2 = g(\Theta_2^T \mathbf{x}) = g(\theta_{20}x_0 + \theta_{21}x_1 + \theta_{22}x_2 + \theta_{23}x_3)$$

$$a_3 = g(\Theta_3^T \mathbf{x}) = g(\theta_{30}x_0 + \theta_{31}x_1 + \theta_{32}x_2 + \theta_{33}x_3)$$

# Perceptron




$$a_1 = g(\Theta_1^T \mathbf{x}) = g(\theta_{10} x_0 + \theta_{11} x_1 + \theta_{12} x_2 + \theta_{13} x_3)$$

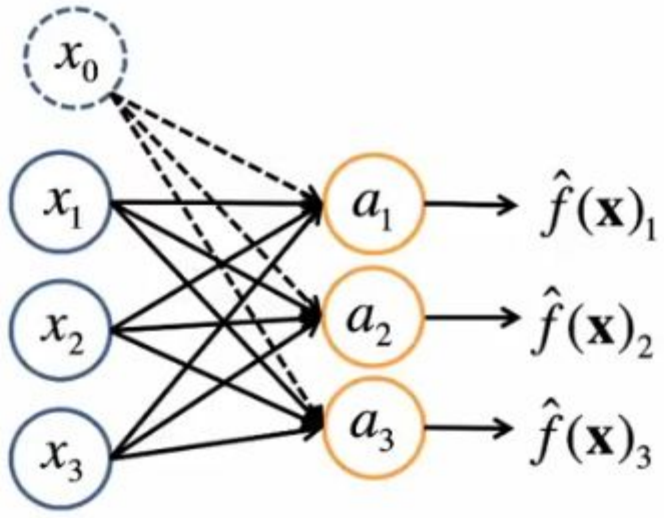
$$a_2 = g(\Theta_2^T \mathbf{x}) = g(\theta_{20} x_0 + \theta_{21} x_1 + \theta_{22} x_2 + \theta_{23} x_3)$$

$$a_3 = g(\Theta_3^T \mathbf{x}) = g(\theta_{30} x_0 + \theta_{31} x_1 + \theta_{32} x_2 + \theta_{33} x_3)$$

**Função limiar (ou sinal):**

$$g(z) = \begin{cases} +1 & se\ z \geq 0 \\ -1 & se\ z < 0 \end{cases}$$

# Perceptron



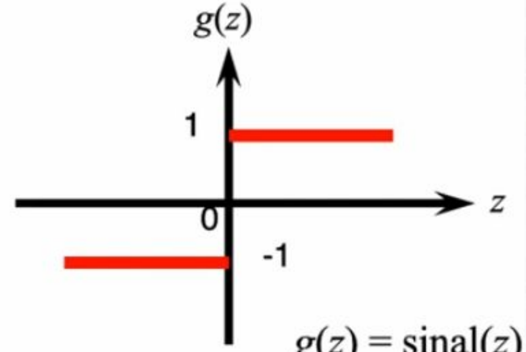

$$a_1 = g(\Theta_1^T \mathbf{x}) = g(\theta_{10}x_0 + \theta_{11}x_1 + \theta_{12}x_2 + \theta_{13}x_3)$$

$$a_2 = g(\Theta_2^T \mathbf{x}) = g(\theta_{20}x_0 + \theta_{21}x_1 + \theta_{22}x_2 + \theta_{23}x_3)$$

$$a_3 = g(\Theta_3^T \mathbf{x}) = g(\theta_{30}x_0 + \theta_{31}x_1 + \theta_{32}x_2 + \theta_{33}x_3)$$

**Função limiar (ou sinal):**

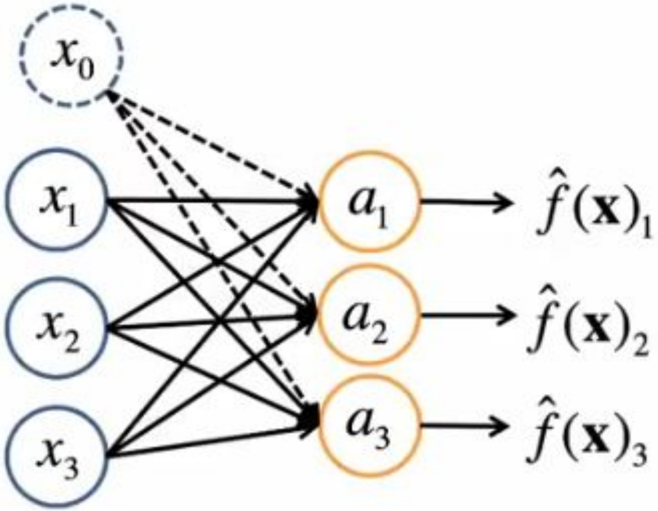
$$g(z) = \begin{cases} +1 & se\ z \geq 0 \\ -1 & se\ z < 0 \end{cases}$$




$$g(z) = \text{sinal}(z)$$

$$\hat{f}(\mathbf{x}) = g(z)$$

# Perceptron




$$a_1 = g(\Theta_1^T\mathbf{x}) = g(\theta_{10}x_0 + \theta_{11}x_1 + \theta_{12}x_2 + \theta_{13}x_3)$$

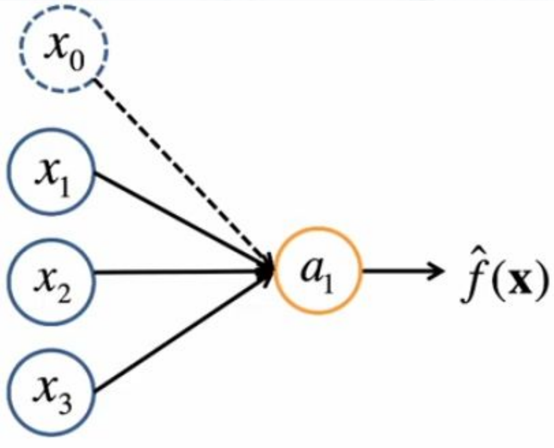
$$a_2 = g(\Theta_2^T\mathbf{x}) = g(\theta_{20}x_0 + \theta_{21}x_1 + \theta_{22}x_2 + \theta_{23}x_3)$$

$$a_3 = g(\Theta_3^T\mathbf{x}) = g(\theta_{30}x_0 + \theta_{31}x_1 + \theta_{32}x_2 + \theta_{33}x_3)$$

*para c =1...k*
    *para j =0...m*
        $\theta_{cj} := rand(-v, +v)$
*repita*
    *para cada par de treinamento* $(\mathbf{x}^{(i)}, f(\mathbf{x}^{(i)}))$
        *para c =1...k*
            $\hat{f}(\mathbf{x}^{(i)})_c := \text{sinal}(\Theta_c^T\mathbf{x}^{(i)})$
        *para c =1...k*
            *para j =0...m*
                $\theta_{cj} := \theta_{cj} - \alpha(\hat{f}(\mathbf{x}^{(i)})_c - f(\mathbf{x}^{(i)}))x_j^{(i)}$
*até convergir (ou erro ser aceitável)*

# Exemplo

- Dada uma rede Perceptron com:
  - 3 atributos de entrada
  - 2 classes (logo, apenas um neurônio)
  - Pesos iniciais:
    - $\theta_0$ = -0.5
    - $\theta_1$ = 0.4
    - $\theta_2$ = -0.6
    - $\theta_3$ = 0.6
  - Taxa de aprendizado $\alpha$ = 0.4




a. Ensinar a rede com os exemplos (001, -1) e (110, +1)
b. Definir a classe dos exemplos de teste 111, 000, 100 e 011

# Exemplo

a. **Treinar a rede**
   i. Instância $x^{(1)} = [1\ 0\ 0\ 1]^T$ $f(\boldsymbol{x}^{(1)}) = -1$

$$\Theta = \begin{bmatrix} \theta_0 = -0.5 \\ \theta_1 = 0.4 \\ \theta_2 = -0.6 \\ \theta_3 = 0.6 \end{bmatrix}$$

# Exemplo

a. **Treinar a rede**
   i. Instância $x^{(1)} = [1\ 0\ 0\ 1]^T$ $f(\boldsymbol{x}^{(1)}) = -1$
   Passo 1: definir a saída da rede

$$\Theta = \begin{bmatrix} \theta_0 = -0.5 \\ \theta_1 = 0.4 \\ \theta_2 = -0.6 \\ \theta_3 = 0.6 \end{bmatrix}$$

# Exemplo

a. **Treinar a rede**

   i. Instância $x^{(1)} = [1\ 0\ 0\ 1]^T$ $f(\boldsymbol{x}^{(1)}) = -1$

   Passo 1: definir a saída da rede

$$\Theta = \begin{bmatrix} \theta_0 = -0.5 \\ \theta_1 = 0.4 \\ \theta_2 = -0.6 \\ \theta_3 = 0.6 \end{bmatrix}$$

$$\Theta^T \mathbf{x}^{(1)} = (1 \times -0.5) + (0 \times 0.4) + (0 \times -0.6) + (1 \times 0.6) = 0.1$$

# Exemplo

$$\Theta = \begin{bmatrix} \theta_0 = -0.5 \\ \theta_1 = 0.4 \\ \theta_2 = -0.6 \\ \theta_3 = 0.6 \end{bmatrix}$$

a. **Treinar a rede**

   i. Instância $x^{(1)} = [1\ 0\ 0\ 1]^T$ $f(\boldsymbol{x}^{(1)}) = -1$

   Passo 1: definir a saída da rede

$$\Theta^T \mathbf{x}^{(1)} = (1 \times -0.5) + (0 \times 0.4) + (0 \times -0.6) + (1 \times 0.6) = 0.1$$

$$\hat{f}(\mathbf{x}^{(1)}) = \text{sinal}(\Theta^T \mathbf{x}^{(1)}) = +1 \text{ (pois } 0.1 \geq 0)$$

# Exemplo

a. **Treinar a rede**

   i. Instância $x^{(1)} = [1\ 0\ 0\ 1]^T$ $f(\boldsymbol{x}^{(1)}) = -1$

   Passo 1: definir a saída da rede

$$\Theta = \begin{bmatrix} \theta_0 = -0.5 \\ \theta_1 = 0.4 \\ \theta_2 = -0.6 \\ \theta_3 = 0.6 \end{bmatrix}$$

$$\Theta^T \mathbf{x}^{(1)} = (1 \times -0.5) + (0 \times 0.4) + (0 \times -0.6) + (1 \times 0.6) = 0.1$$

$$\hat{f}(\mathbf{x}^{(1)}) = \text{sinal}(\Theta^T \mathbf{x}^{(1)}) = +1 \text{ (pois } 0.1 \geq 0)$$

   Passo 2: atualizar pesos (pois $f'(\boldsymbol{x}^{(1)})$ != $f(\boldsymbol{x}^{(1)})$) ⇒

$$\theta_j = \theta_j - \alpha(\hat{f}(\mathbf{x}) - f(\mathbf{x}))x_j$$

# Exemplo

**a. Treinar a rede**

i. Instância $x^{(1)} = [1\ 0\ 0\ 1]^T$ $f(\boldsymbol{x}^{(1)}) = -1$

Passo 1: definir a saída da rede

$$\Theta^T \mathbf{x}^{(1)} = (1 \times -0.5) + (0 \times 0.4) + (0 \times -0.6) + (1 \times 0.6) = 0.1$$

$$\hat{f}(\mathbf{x}^{(1)}) = \text{sinal}(\Theta^T \mathbf{x}^{(1)}) = +1 \text{ (pois } 0.1 \geq 0)$$

Passo 2: atualizar pesos (pois $f'(\boldsymbol{x}^{(1)})$ != $f(\boldsymbol{x}^{(1)})$) $\Rightarrow$ $\theta_j = \theta_j - \alpha(\hat{f}(\mathbf{x}) - f(\mathbf{x}))x_j$

$$\begin{aligned}
\theta_0 &= -0.5 - 0.4(+1 - (-1))1 = -1.3 \\
\theta_1 &= 0.4 - 0.4(+1 - (-1))0 = 0.4 \\
\theta_2 &= -0.6 - 0.4(+1 - (-1))0 = -0.6 \\
\theta_3 &= 0.6 - 0.4(+1 - (-1))1 = -0.2
\end{aligned}$$

$$\Theta = \begin{bmatrix} \theta_0 = -0.5 \\ \theta_1 = 0.4 \\ \theta_2 = -0.6 \\ \theta_3 = 0.6 \end{bmatrix}$$

# Exemplo

a. **Treinar a rede**
   i. Instância $x^{(2)} = [1\ 1\ 1\ 0]^T$ $f(\boldsymbol{x}^{(2)}) = +1$

$$\Theta = \begin{bmatrix} \theta_0 = -1.3 \\ \theta_1 = 0.4 \\ \theta_2 = -0.6 \\ \theta_3 = -0.2 \end{bmatrix}$$

# Exemplo

**a. Treinar a rede**

i. Instância $x^{(2)} = [1\ 1\ 1\ 0]^T$ $f(\boldsymbol{x}^{(2)}) = +1$

Passo 1: definir a saída da rede

$$\Theta^T \mathbf{x}^{(2)} = (1 \times -1.3) + (1 \times 0.4) + (1 \times -0.6) + (0 \times -0.2) = -1.5$$

$$\hat{f}(\mathbf{x}^{(2)}) = \text{sinal}(\Theta^T \mathbf{x}^{(2)}) = -1 \text{ (pois -1.5} < 0)$$

$$\Theta = \begin{bmatrix} \theta_0 = -1.3 \\ \theta_1 = 0.4 \\ \theta_2 = -0.6 \\ \theta_3 = -0.2 \end{bmatrix}$$

# Exemplo

$$\Theta = \begin{bmatrix} \theta_0 = -1.3 \\ \theta_1 = 0.4 \\ \theta_2 = -0.6 \\ \theta_3 = -0.2 \end{bmatrix}$$

a. **Treinar a rede**

   i. Instância $x^{(2)} = [1\ 1\ 1\ 0]^T$ $f(\boldsymbol{x}^{(2)}) = +1$

   Passo 1: definir a saída da rede

$$\Theta^T \mathbf{x}^{(2)} = (1 \times -1.3) + (1 \times 0.4) + (1 \times -0.6) + (0 \times -0.2) = -1.5$$
$$\hat{f}(\mathbf{x}^{(2)}) = \text{sinal}(\Theta^T \mathbf{x}^{(2)}) = -1 \text{ (pois -1.5} < 0)$$

   Passo 2: atualizar pesos (pois $f'(\boldsymbol{x}^{(2)})$ != $f(\boldsymbol{x}^{(2)})$)

$$\theta_j = \theta_j - \alpha(\hat{f}(\mathbf{x}) - f(\mathbf{x}))x_j$$

$$\theta_0 = -1.3 - 0.4(-1 - (+1))1 = -0.5$$
$$\theta_1 = 0.4 - 0.4(-1 - (+1))1 = 1.2$$
$$\theta_2 = -0.6 - 0.4(-1 - (+1))1 = 0.2$$
$$\theta_3 = -0.2 - 0.4(-1 - (+1))0 = -0.2$$

# Exemplo

a. **Treinar a rede**

    i. Instância $x^{(1)} = [1\ 0\ 0\ 1]^T$ $f(\boldsymbol{x}^{(1)}) = -1$

    Passo 1: definir a saída da rede

$$\Theta = \begin{bmatrix} \theta_0 = -0.5 \\ \theta_1 = 1.2 \\ \theta_2 = 0.2 \\ \theta_3 = -0.2 \end{bmatrix}$$

# Exemplo

a. **Treinar a rede**
   i. Instância $x^{(1)} = [1\ 0\ 0\ 1]^T$ $f(\boldsymbol{x}^{(1)}) = -1$
   Passo 1: definir a saída da rede

$$\Theta = \begin{bmatrix} \theta_0 = -0.5 \\ \theta_1 = 1.2 \\ \theta_2 = 0.2 \\ \theta_3 = -0.2 \end{bmatrix}$$

$$\Theta^T \mathbf{x}^{(1)} = (1 \times -0.5) + (0 \times 1.2) + (0 \times 0.2) + (1 \times -0.2) = -0.7$$

$$\hat{f}(\mathbf{x}^{(1)}) = \text{sinal}(\Theta^T \mathbf{x}^{(1)}) = -1 \text{ (pois -0.7} < 0)$$

# Exemplo

a. **Treinar a rede**

   i. Instância $x^{(1)} = [1\ 0\ 0\ 1]^T$ $f(\boldsymbol{x}^{(1)}) = -1$

   Passo 1: definir a saída da rede

$$\Theta^T \mathbf{x}^{(1)} = (1 \times -0.5) + (0 \times 1.2) + (0 \times 0.2) + (1 \times -0.2) = -0.7$$
$$\hat{f}(\mathbf{x}^{(1)}) = \text{sinal}(\Theta^T \mathbf{x}^{(1)}) = -1 \text{ (pois -0.7} < 0)$$

   Passo 2: atualizar pesos

$$\Theta = \begin{bmatrix} \theta_0 = -0.5 \\ \theta_1 = 1.2 \\ \theta_2 = 0.2 \\ \theta_3 = -0.2 \end{bmatrix}$$

# Exemplo

$$\Theta = \begin{bmatrix} \theta_0 = -0.5 \\ \theta_1 = 1.2 \\ \theta_2 = 0.2 \\ \theta_3 = -0.2 \end{bmatrix}$$

a. **Treinar a rede**
   i. Instância $x^{(1)} = [1\ 0\ 0\ 1]^T$ $f(\boldsymbol{x}^{(1)}) = -1$
      Passo 1: definir a saída da rede

$$\Theta^T \mathbf{x}^{(1)} = (1 \times -0.5) + (0 \times 1.2) + (0 \times 0.2) + (1 \times -0.2) = -0.7$$
$$\hat{f}(\mathbf{x}^{(1)}) = \text{sinal}(\Theta^T \mathbf{x}^{(1)}) = -1 \text{ (pois } -0.7 < 0)$$

Passo 2: atualizar pesos

Não é necessário, pois $f'(\boldsymbol{x}^{(1)}) = f(\boldsymbol{x}^{(1)})$

# Exemplo

a. **Treinar a rede**

   i. Instância $x^{(2)} = [1\ 1\ 1\ 0]^T$ $f(\boldsymbol{x}^{(2)}) = +1$

   Passo 1: definir a saída da rede

$$\Theta = \begin{bmatrix} \theta_0 = -0.5 \\ \theta_1 = 1.2 \\ \theta_2 = 0.2 \\ \theta_3 = -0.2 \end{bmatrix}$$

$$\Theta^T \mathbf{x}^{(2)} = (1 \times -0.5) + (1 \times 1.2) + (1 \times 0.2) + (0 \times -0.2) = 0.9$$
$$\hat{f}(\mathbf{x}^{(2)}) = \text{sinal}(\Theta^T \mathbf{x}^{(2)}) = +1 \text{ (pois } 0.9 \geq 0)$$

   Passo 2: atualizar pesos

   Não é necessário, pois $f'(\boldsymbol{x}^{(2)}) = f(\boldsymbol{x}^{(2)})$

# Exemplo

$$\Theta = \begin{bmatrix} \theta_0 = -0.5 \\ \theta_1 = 1.2 \\ \theta_2 = 0.2 \\ \theta_3 = -0.2 \end{bmatrix}$$

a. **Testar a rede**
   i. Instância $x^{(3)} = [1\ 1\ 1\ 1]^T$

$$\Theta^T \mathbf{x}^{(3)} = (1 \times -0.5) + (1 \times 1.2) + (1 \times 0.2) + (1 \times -0.2) = 0.7$$

$$\hat{f}(\mathbf{x}^{(3)}) = \text{sinal}(\Theta^T \mathbf{x}^{(3)}) = +1 \text{ (pois } 0.7 \geq 0)$$

# Exemplo

a. **Testar a rede**
   i. Instância $x^{(4)} = [1\ 0\ 0\ 0]^T$

$$\Theta = \begin{bmatrix} \theta_0 = -0.5 \\ \theta_1 = 1.2 \\ \theta_2 = 0.2 \\ \theta_3 = -0.2 \end{bmatrix}$$

$$\Theta^T \mathbf{x}^{(4)} = (1 \times -0.5) + (0 \times 1.2) + (0 \times 0.2) + (0 \times -0.2) = -0.5$$

$$\hat{f}(\mathbf{x}^{(4)}) = \text{sinal}(\Theta^T \mathbf{x}^{(4)}) = -1 \text{ (pois } -0.5 < 0)$$

# Exercício

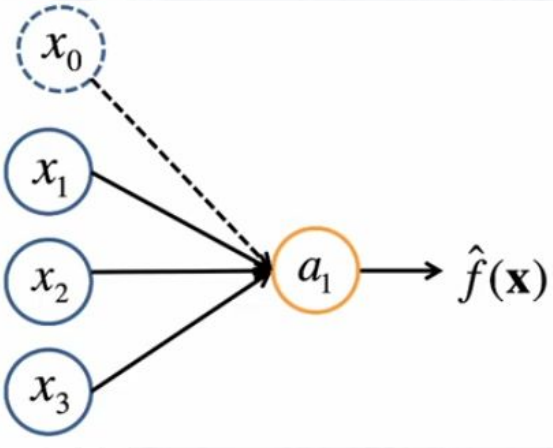
Seja o seguinte cadastro de pacientes:

| Febre | Enjoo | Dores | Diagnóstico |
|---|---|---|---|
| 1 | 1 | 1 | 1 (doente) |
| 0 | 0 | 0 | -1 (saudável) |
| 0 | 1 | 0 | -1 |
| 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | -1 |
| 0 | 0 | 1 | 1 |

1. Treine uma rede do tipo perceptron que seja capaz de distinguir entre pacientes e saudáveis

2. Teste a rede para os seguintes novos cases:
   a. [0 0 1]
   b. [0 1 1]

# Perceptron

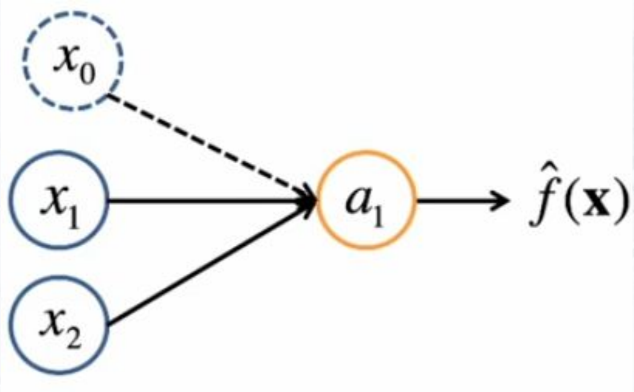



**Perceptron Convergence Theorem:** Rosenblatt provou que o Perceptron converge em número finito de iterações para um vetor de pesos que corretamente classifica todas as instâncias de treino desde que elas sejam linearmente separáveis e que $0 < \alpha < 1$

# Perceptron

XOR

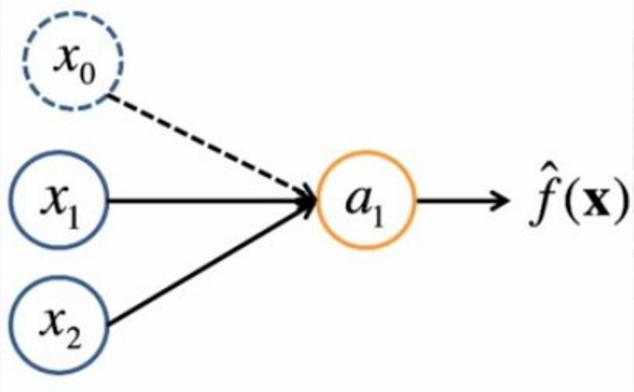
| | |
|---|---|
| 0, 0 | $\rightarrow 0$ |
| 0, 1 | $\rightarrow 1$ |
| 1, 0 | $\rightarrow 1$ |
| 1, 1 | $\rightarrow 0$ |

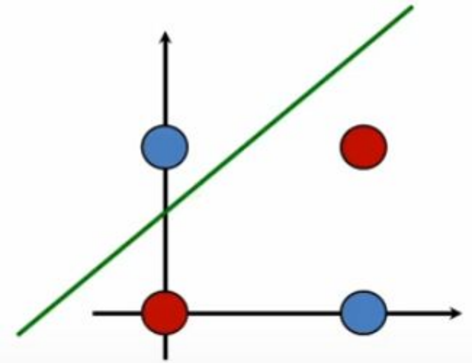


# Perceptron

XOR

$0, 0 \rightarrow 0$
$0, 1 \rightarrow 1$
$1, 0 \rightarrow 1$
$1, 1 \rightarrow 0$

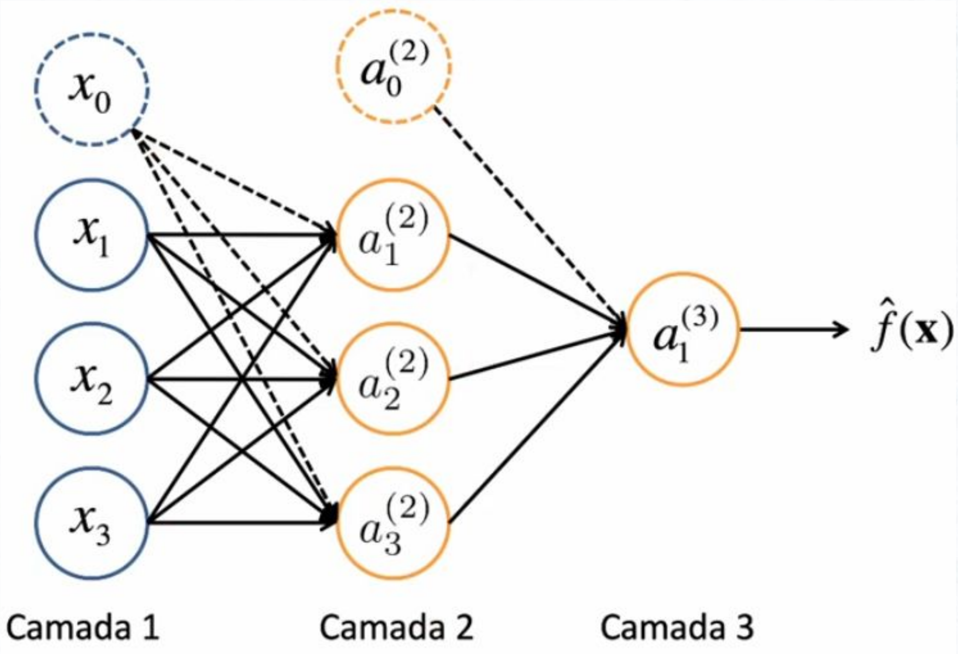


# Perceptron

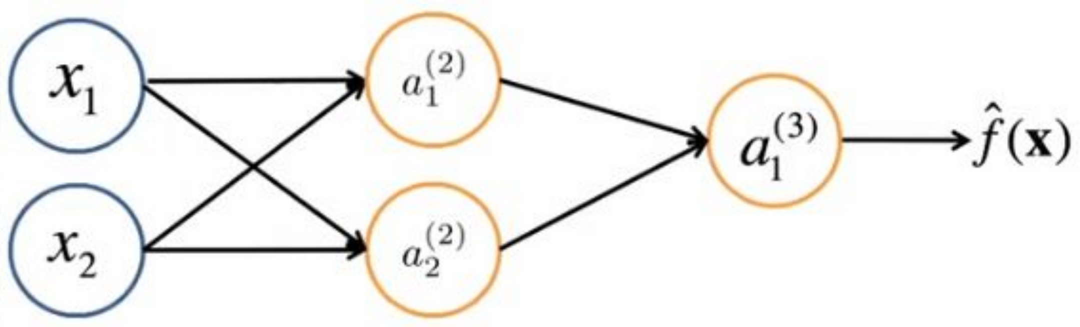
- O que fazer em problemas não linearmente separáveis?
    - Utilizar uma MLP
        - Perceptron Multi-Camada

# Perceptron Multi-Camada



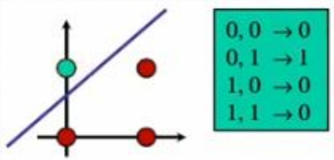
# Perceptron Multi-Camada

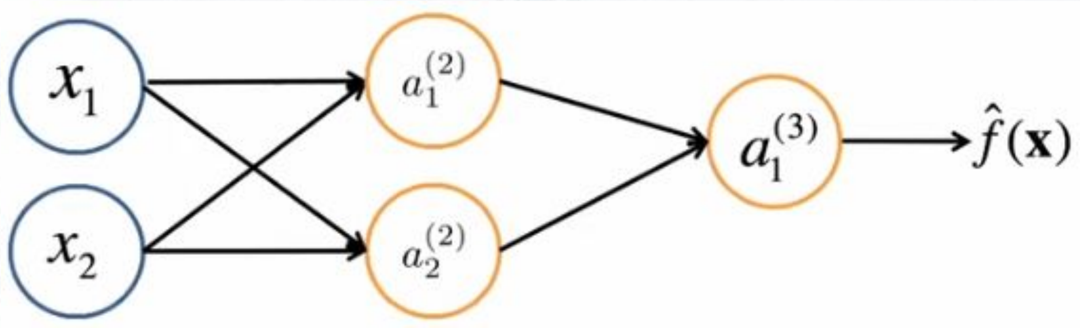
- Por que o Perceptron Multi-Camada consegue resolver o problema não-linear do XOR?



# Perceptron Multi-Camada

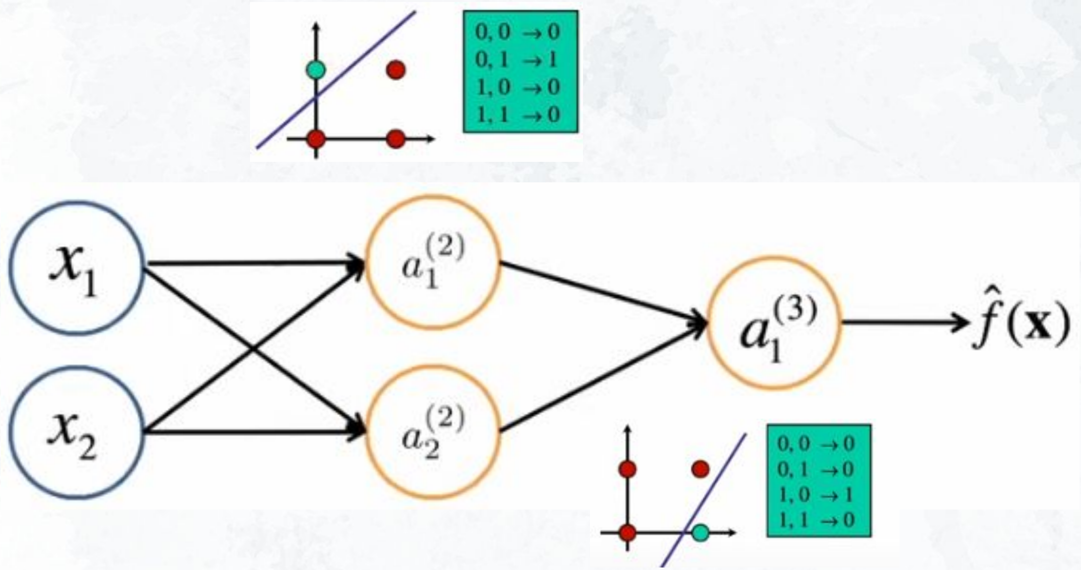
- Por que o Perceptron Multi-Camada consegue resolver o problema não-linear do XOR?



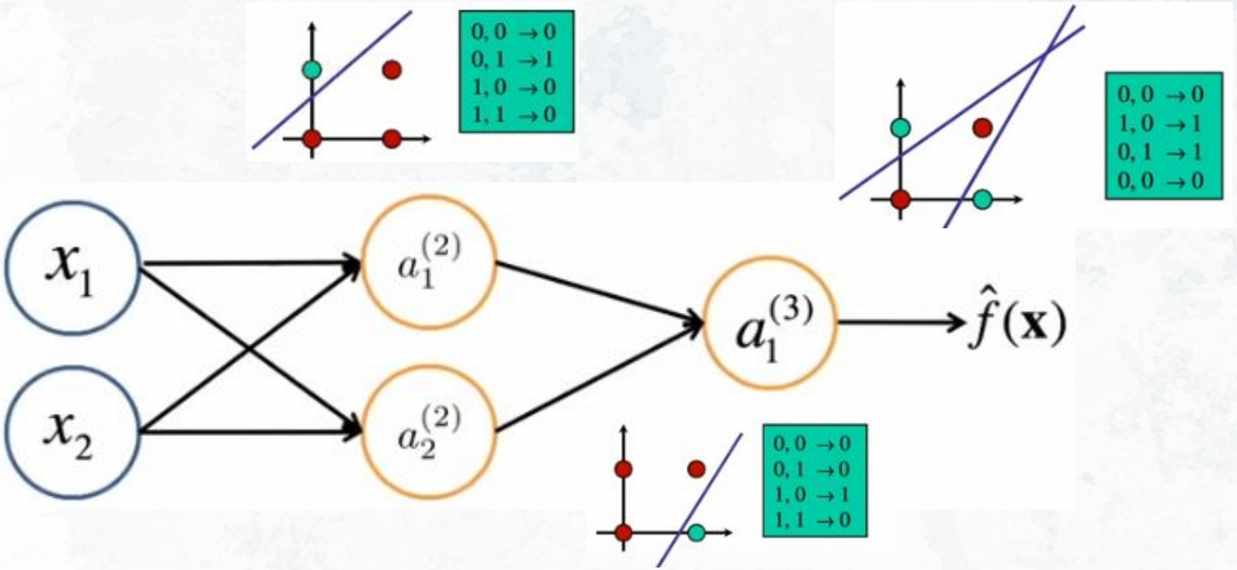
# Perceptron Multi-Camada

- Por que o Perceptron Multi-Camada consegue resolver o problema não-linear do XOR?



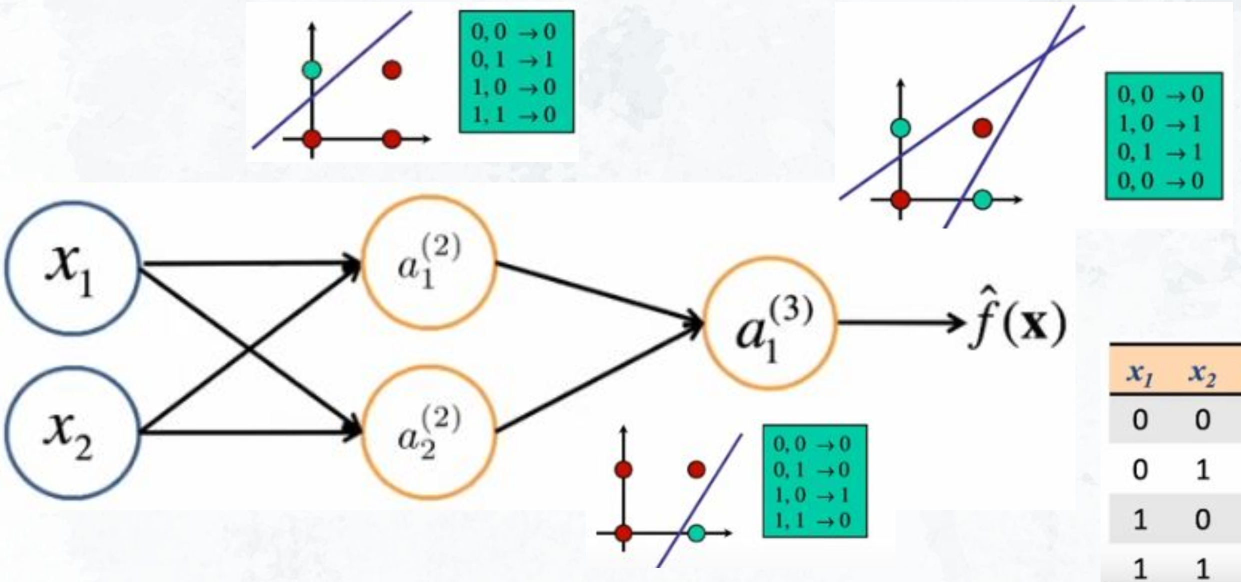
# Perceptron Multi-Camada

- Por que o Perceptron Multi-Camada consegue resolver o problema não-linear do XOR?



# Perceptron Multi-Camada

- Por que o Perceptron Multi-Camada consegue resolver o problema não-linear do XOR?

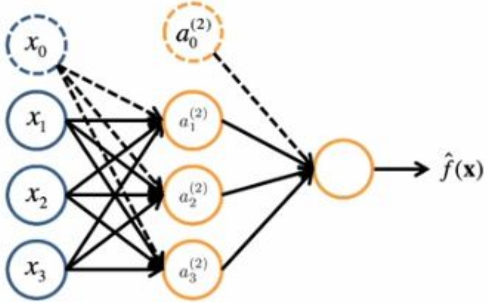



| $x_1$ | $x_2$ | $a_1^{(2)}$ | $a_2^{(2)}$ | | $a_1^{(3)}$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | → | 0 |
| 0 | 1 | 1 | 0 | → | 1 |
| 1 | 0 | 0 | 1 | → | 1 |
| 1 | 1 | 0 | 0 | → | 0 |

# Perceptron Multi-Camada

- É a arquitetura de RNA **mais utilizada**
    - Uma ou mais camadas intermediárias de neurônios
- Funcionalidade
    - Uma camada intermediária: aproxima qualquer **função contínua ou Booleana**
    - Duas camadas intermediárias: aproxima **qualquer função**
- Treinada com o algoritmo “**backpropagation**” (algoritmo de retropropagação)

# Perceptron Multi-Camada




$L =$ número total de camadas

$a_i^{(j)} =$ ativação da unidade *i* na camada *j*

$\Theta^{(j)} =$ matriz de pesos ligando a camada *j* com a camada *j+1*

$\Theta_{i,k}^{(j)} =$ valor do peso que liga a k-ésima unidade da j-ésima camada com a i-ésima unidade da (j+1)-ésima camada

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$\hat{f}(\mathbf{x}) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

# Perceptron Multi-Camada

- Função de custo:
    - Regressão Logística:

$$J(\Theta) = -\frac{1}{N}\left[\sum_{i=1}^{N} f(\mathbf{x}^{(i)})\log(\hat{f}(\mathbf{x}^{(i)})) + (1 - f(\mathbf{x}^{(i)}))\log(1 - \hat{f}(\mathbf{x}^{(i)}))\right]$$

- Rede Neural:

$$J(\Theta) = -\frac{1}{N}\left[\sum_{i=1}^{N}\sum_{k=1}^{K} f(\mathbf{x}^{(i)})_k \log(\hat{f}(\mathbf{x}^{(i)})_k) + (1 - f(\mathbf{x}^{(i)})_k)\log(1 - \hat{f}(\mathbf{x}^{(i)})_k)\right]$$

# Perceptron Multi-Camada

- Número de neurônios por camada
    - Função do número de entradas/saídas?
    - Depedende principalmente
        - Número de **instâncias/atributos**
        - Quantidade de ruído
        - **Complexidade da função desconhecida**
        - Distribuição dos dados
    - Heurística:
        - Número **comparável à quantidade de entradas**
        - Quanto maior, geralmente melhor
            - Porém, mais lento!
            - Possibilidade de overfitting!

# Perceptron Multi-Camada

- Mínimos locais
  - Função custo é **não-convexa**, logo, suscetível a mínimos locais
  - Utilizar **aprendizado online** (gradiente estocástico)

- Problemas Numéricos
  - Normalizar dados
  - Utilizar outros algoritmos de otimização mais robustos

# Perceptron Multi-Camada

- Overfitting
  - Depois de certo ponto de treinamento, a rede começa a sofrer **overfitting**
  - Alternativas:
    - Encerrar treinamento mais cedo (**early stop**)
    - Escolher melhor rede de acordo com conjunto de validação (mesmo sendo proveniente de um ponto intermediário de treinamento)
    - **Poda** de conexões e neurônios irrelevantes (prunning)

# Perceptron Multi-Camada

- Atualização dos Pesos
    - Vimos a atualização online (ou sequencial)
    - Pode ser feita a atualização em batch (após apresentação de todas as instâncias de treinamento)
    - Atualmente (*big data*), se usa atualização em mini-batches (gradiente estocástico)
        - Ex: batches de 32, 64 ou 128 instâncias
    - Qual a melhor abordagem?
        - Depende da aplicação!! **(no free-lunch)**

# Perceptron Multi-Camada

- Online (Sequencial - Gradiente Estocástico)
    - Pesos atualizados após apresentação de **cada instância** em ordem aleatória
    - Requer **menos memória**
    - **Mais rápido!**
    - Menos suscetível a mínimos locais
    - Desvantagem: pode se tornar **instável**
        - Geralmente exige mecanismos de controle da taxa de aprendizado

# Perceptron Multi-Camada

- Batch
  - Pesos atualizados após apresentação de **todas as instâncias**
  - Estimativa mais **precisa** do vetor gradiente
  - **Mais estável!**
  - Mais lento!
  - Mínimos locais!