

# Machine Learning:

## uma introdução prática em Python

Otacílio “Minho” Neto, [minhotmog@gmail.com](mailto:minhotmog@gmail.com)

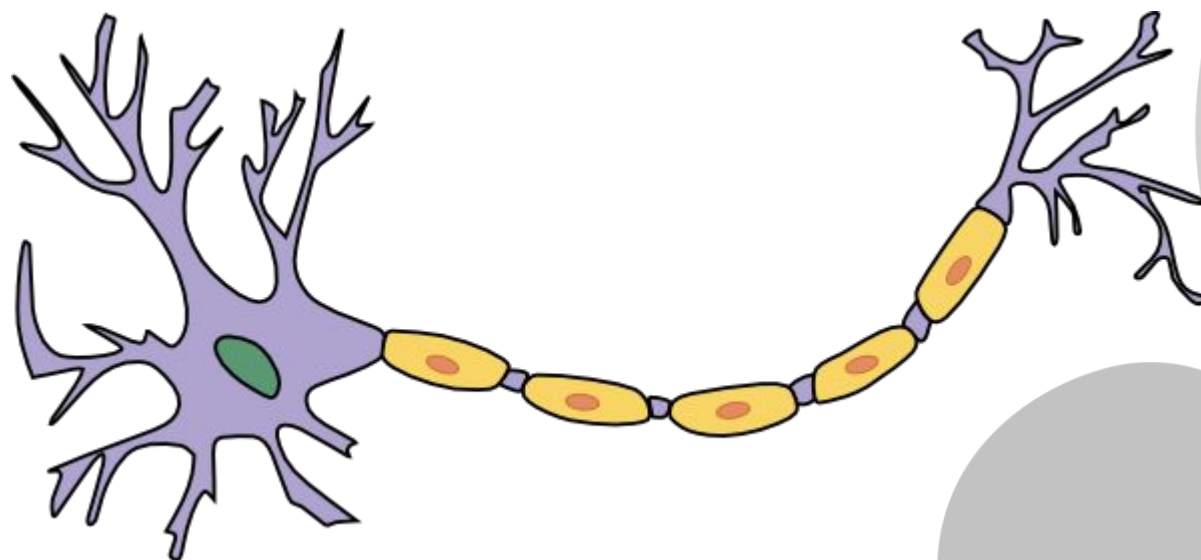


# Redes Neurais: Single-Layer Perceptron

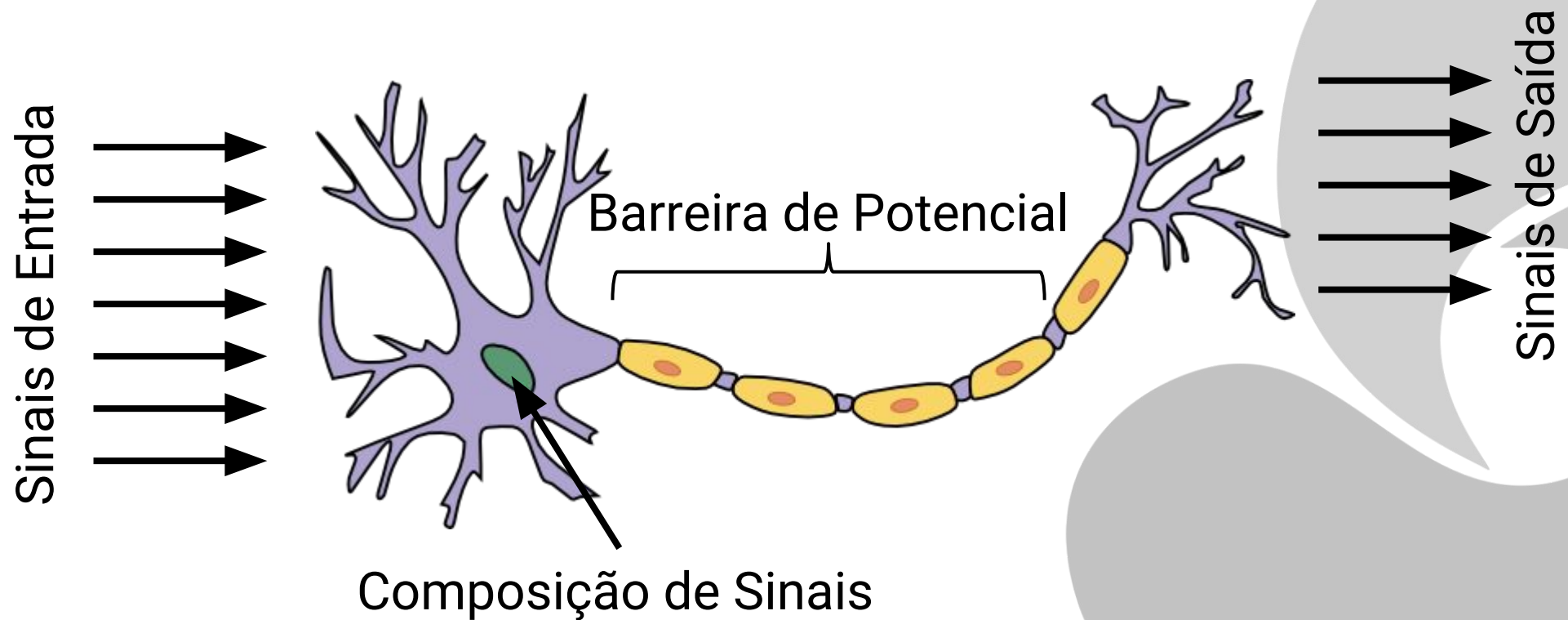
A Rede Neural mais simples que existe!



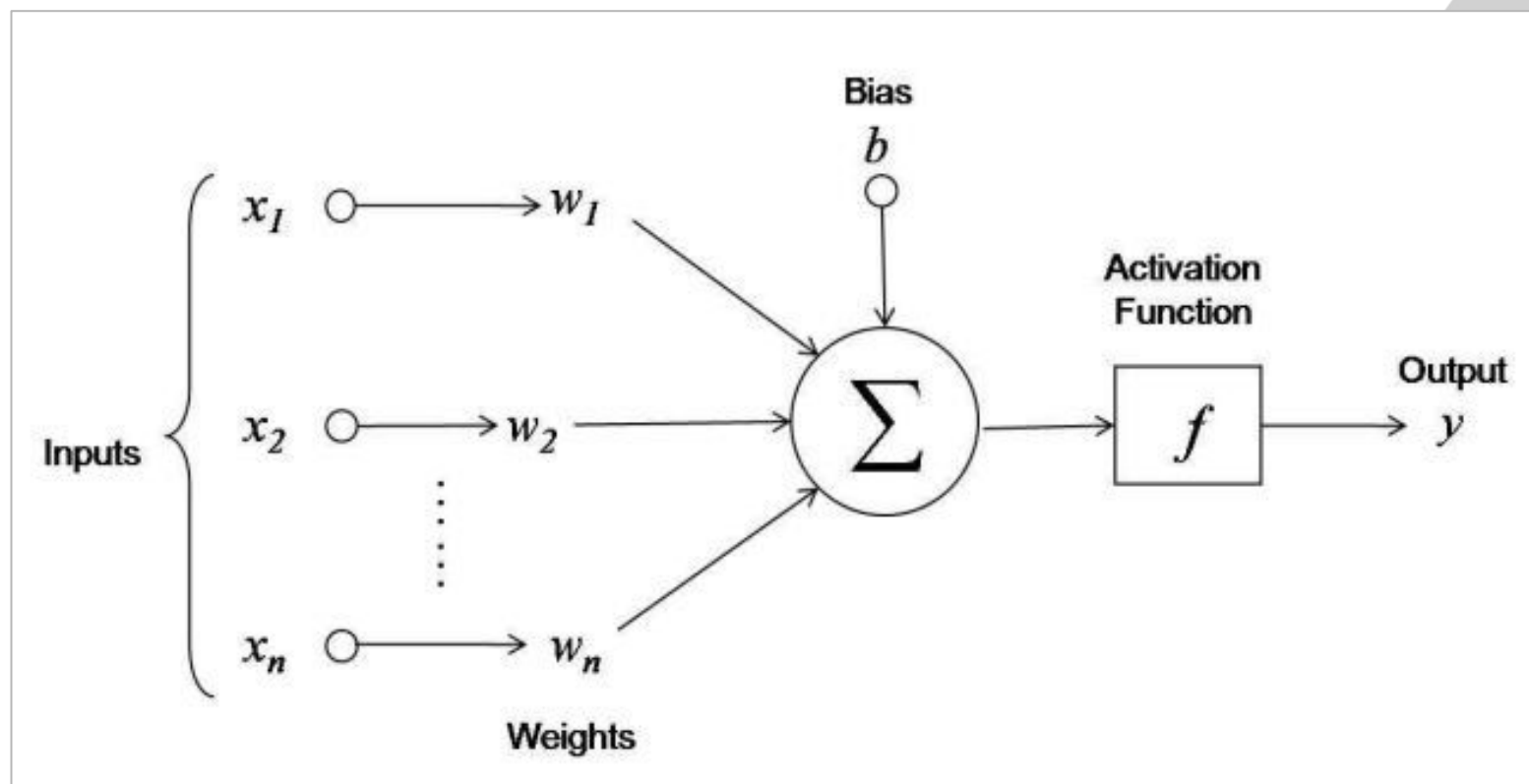
# Motivação Biológica: Neurônio



# Motivação Biológica: Neurônio



# Modelo de McCulloch & Pitts



Fonte: <https://blogs.cornell.edu/>



# Funcionamento de um Neurônio

1. Os sinais de entrada são multiplicados por pesos e somados (*Weighted Sum Function*).

$$s_{net} = w_1x_1 + w_2x_2 + \dots + w_nx_n$$

2. A soma é então passada para uma função de ativação (*Activation Function*).

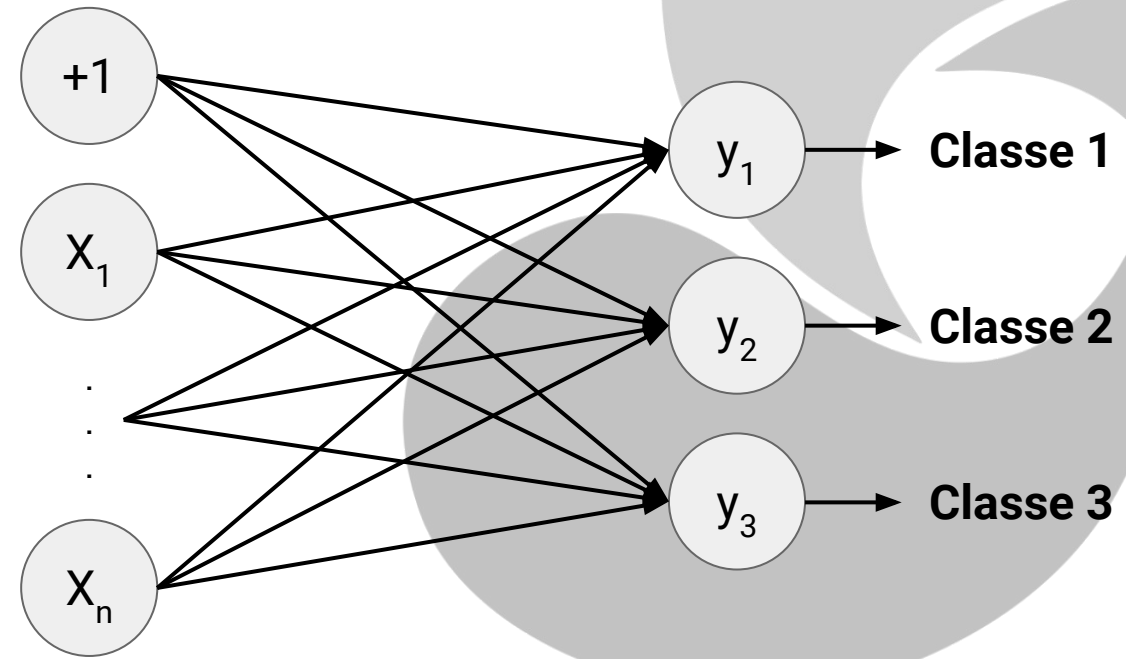
$$\varphi(s_{net}) = \begin{cases} 1 & \text{se } s_{net} \geq \tau \\ 0 & \text{caso contrario} \end{cases}$$

# Single-Layer Perceptron

- O **Single-Layer Perceptron** é a arquitetura de Rede Neural Artificial mais simples. Consiste apenas em uma camada densa diretamente ligado às classes de saída. Por esse motivo, seu funcionamento é idêntico à **Regressão Logística**.

## SLP para Multiclass Classification

$$\Theta = \begin{bmatrix} \Theta_{00} & \Theta_{10} & \cdots & \Theta_{n0} \\ \Theta_{01} & \Theta_{11} & \cdots & \Theta_{n1} \\ \vdots & \vdots & \ddots & \vdots \\ \Theta_{0n} & \Theta_{1n} & \cdots & \Theta_{nn} \end{bmatrix}$$



# Classificação Multiclass

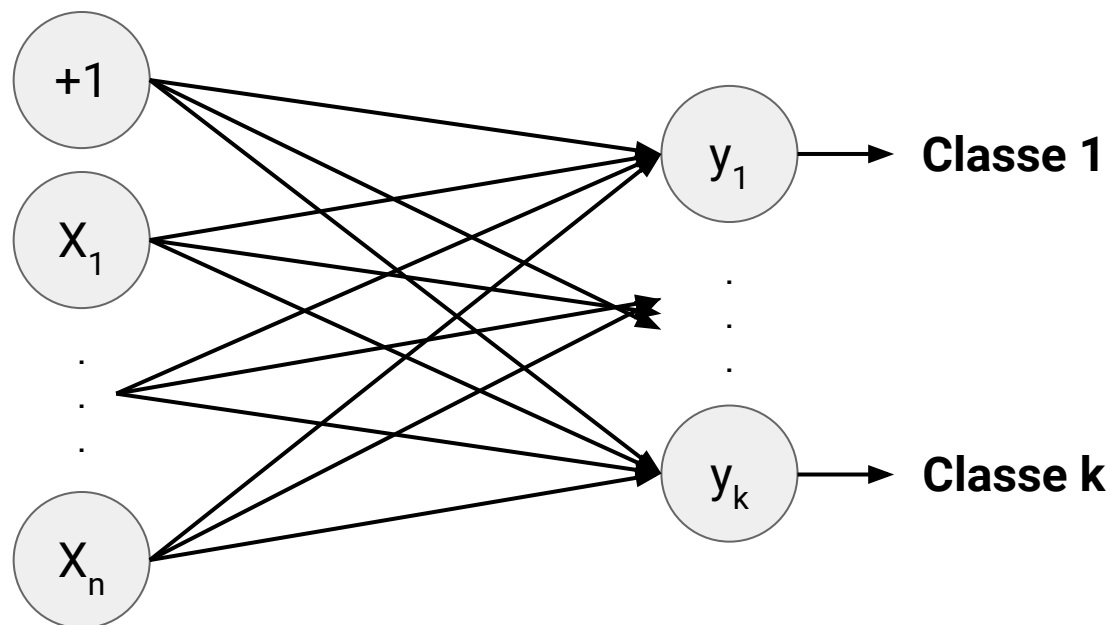
- Nossa Rede Neural irá agregar “*uma Regressão Logística por classe*”. Por isso seus parâmetros são matriciais. Essa técnica chama-se “*one-vs-all*”;
- Classes são enumeradas no Dataset, mas representadas para a Rede Neural utilizando a notação “*one-hot encoding*”:

Exemplar	Classe	Numeração
1	Carro	1
2	Pessoa	2
3	Objeto	3
4	Pessoa	2

Exemplar	Carro	Pessoa	Objeto
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0



# Resumindo: SLP



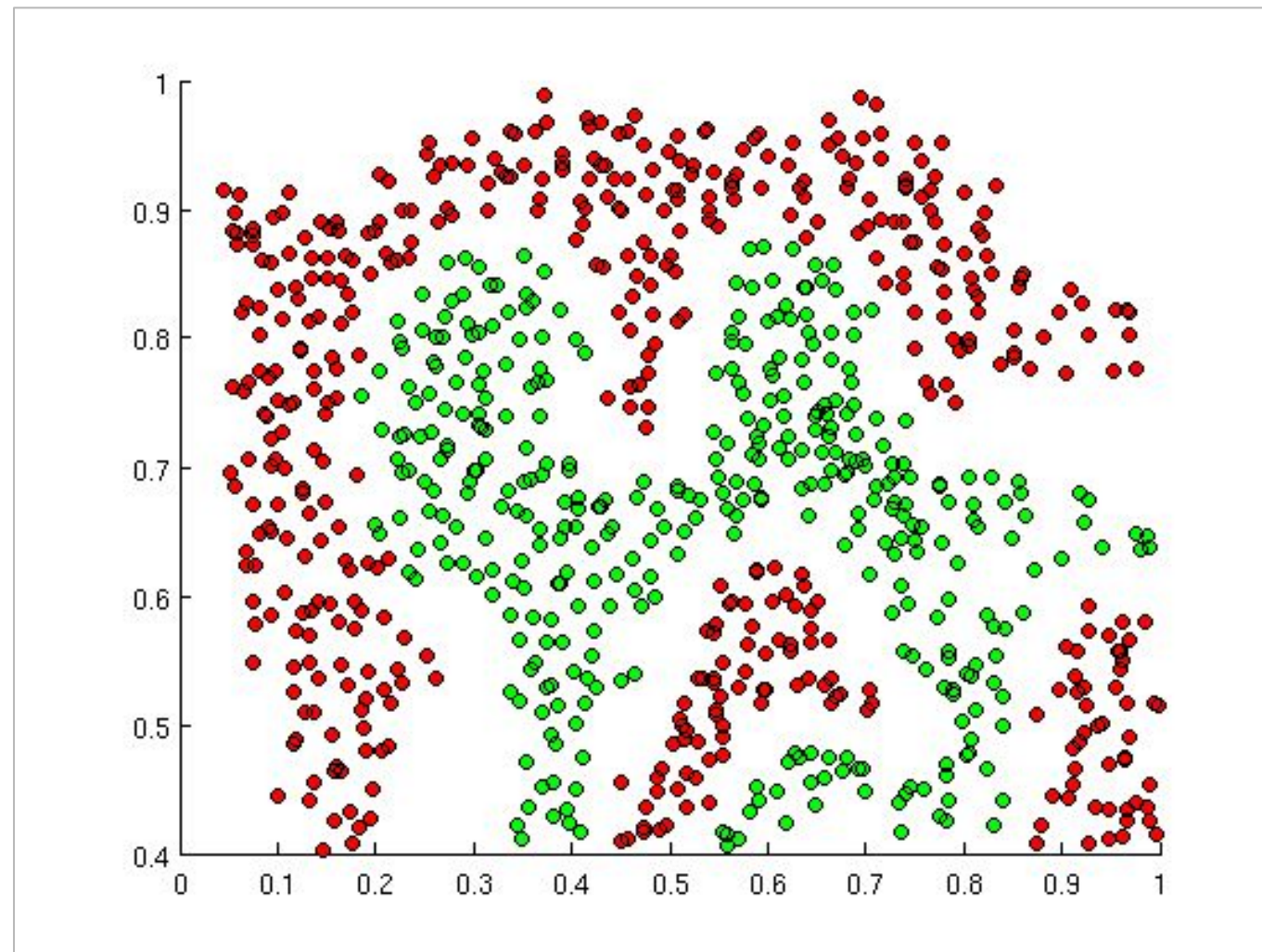
$$X = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ X_1^{(1)} & X_1^{(2)} & \cdots & X_1^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ X_n^{(1)} & X_n^{(2)} & \cdots & X_n^{(m)} \end{bmatrix}; \Theta = \begin{bmatrix} \Theta_{00} & \Theta_{10} & \cdots & \Theta_{k0} \\ \Theta_{01} & \Theta_{11} & \cdots & \Theta_{k1} \\ \vdots & \vdots & \ddots & \vdots \\ \Theta_{0n} & \Theta_{1n} & \cdots & \Theta_{kn} \end{bmatrix} \quad y = \begin{bmatrix} \hat{y}_1^{(1)} & \hat{y}_1^{(2)} & \cdots & \hat{y}_1^{(m)} \\ \hat{y}_2^{(1)} & \hat{y}_2^{(2)} & \cdots & \hat{y}_2^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_k^{(1)} & \hat{y}_k^{(2)} & \cdots & \hat{y}_k^{(m)} \end{bmatrix}$$

# Redes Neurais: Multi-Layer Perceptron

Melhorando a qualidade das nossas Linhas de  
Decisão



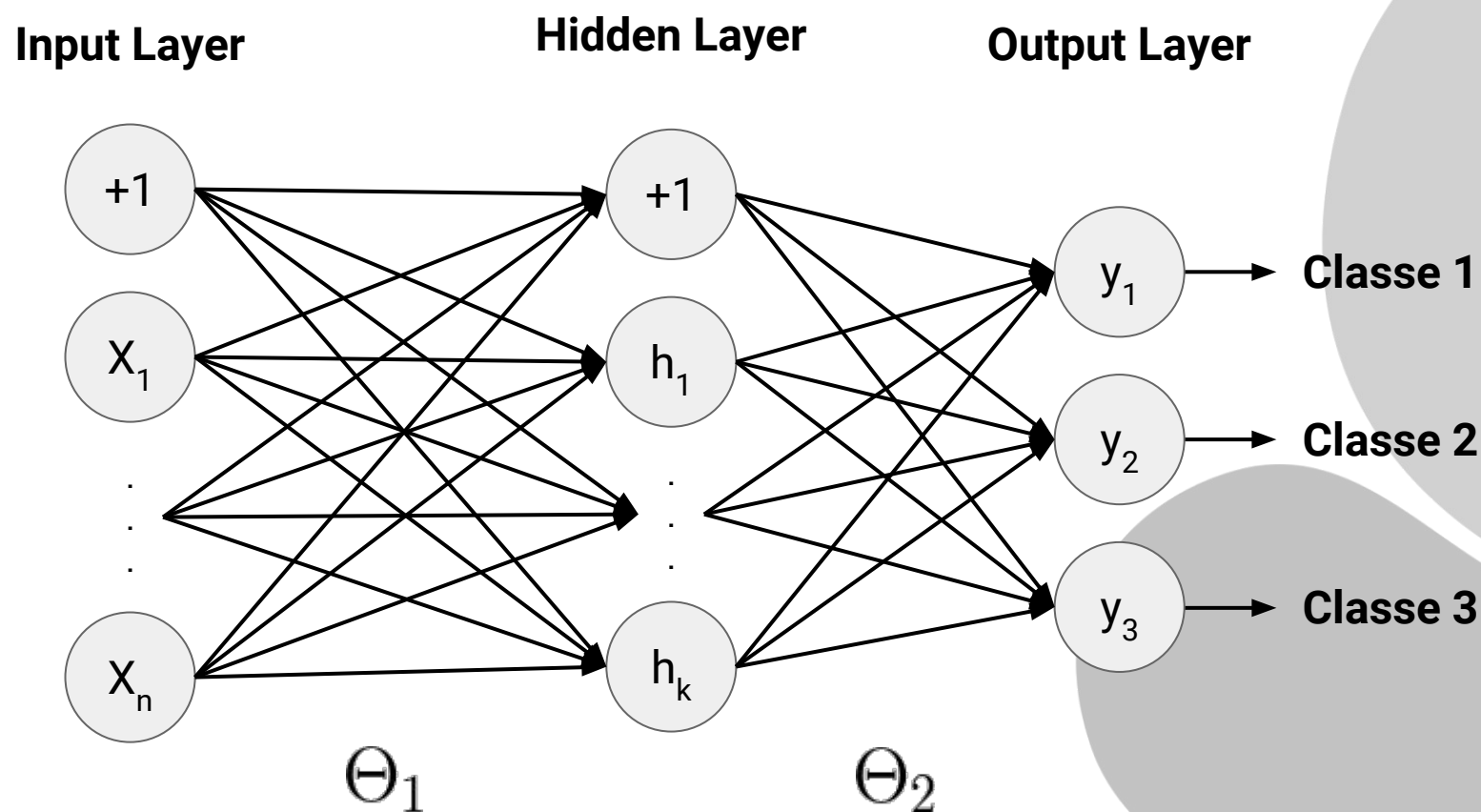
# Desenhe seu Hiperplano Separador!



# Transformação Não-Linear

- Existem casos interessantes de *Classificação* onde a disposição dos exemplos não permitem uma separação por hiperplanos lineares. Dizemos, então, que os “*dados não são linearmente separáveis*”.
- Semelhante à *Regressão Polinomial*, devemos aumentar a complexidade do modelo. No entanto, só criar versões polinomiais não será suficiente.
- As Redes Neurais permitem não só uma transformação não-linear; elas permitem que o **próprio algoritmo defina os parâmetros da transformação!**

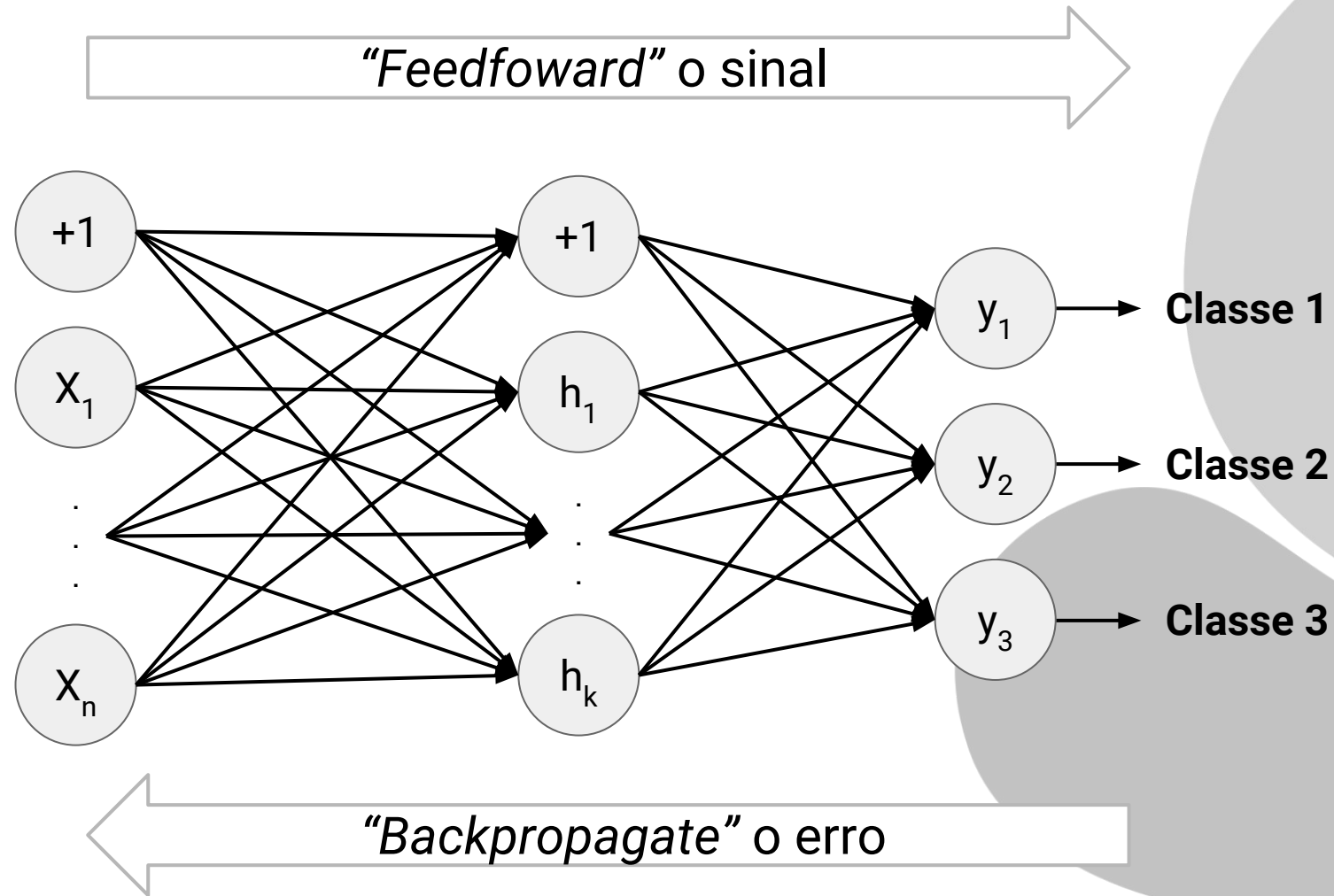
# Multi-Layer Perceptron



# Representação MLP

- A representação do nosso modelo consiste nas duas matrizes de parâmetros:
  - $\Theta_1$  : mapeia os pesos da *Input Layer* para a *Hidden Layer*;
  - $\Theta_2$  : mapeia os pesos da *Hidden Layer* para a *Output Layer*;
- Realizaremos duas ativações para realizar nossa predição. Podemos, então, utilizar o Gradiente Descendente para o treinamento? **Não é tão simples...**
- **Camada de Saída:** erro supervisionado.  
**Camada Oculta:** erro aproximativo.

# Backpropagation





# Backpropagation (Algoritmo)

1. Criar matriz  $\Delta^L$  que representa o gradiente de  $\Theta^L$

2. Realizar Feedforward Propagation:

$$A_1 = \Theta_1^T X \text{ e } A_2 = \Theta_2^T A_1$$

3. Realizar o Backpropagation:

$$\delta^{(2)} = A_2 - y \text{ e } \delta^{(1)} = ((\Theta^2)^T \delta^{(2)}) \cdot (A_2 \cdot (1 - A_2))$$

$$\Delta^1 = \delta^{(1)} A_1^T \text{ e } \Delta^2 = \delta^{(2)} A_2^T$$

4. Realizar o Gradient Descendente:

$$\Theta_j^{(i)} = \Theta_j^{(i)} - \alpha \Delta_j^i$$



# Hands-On!!!



# Obrigado!

minhotmog@gmail.com

www.petcomp.ufc.br

