

REDES NEURAIS

80 anos de história

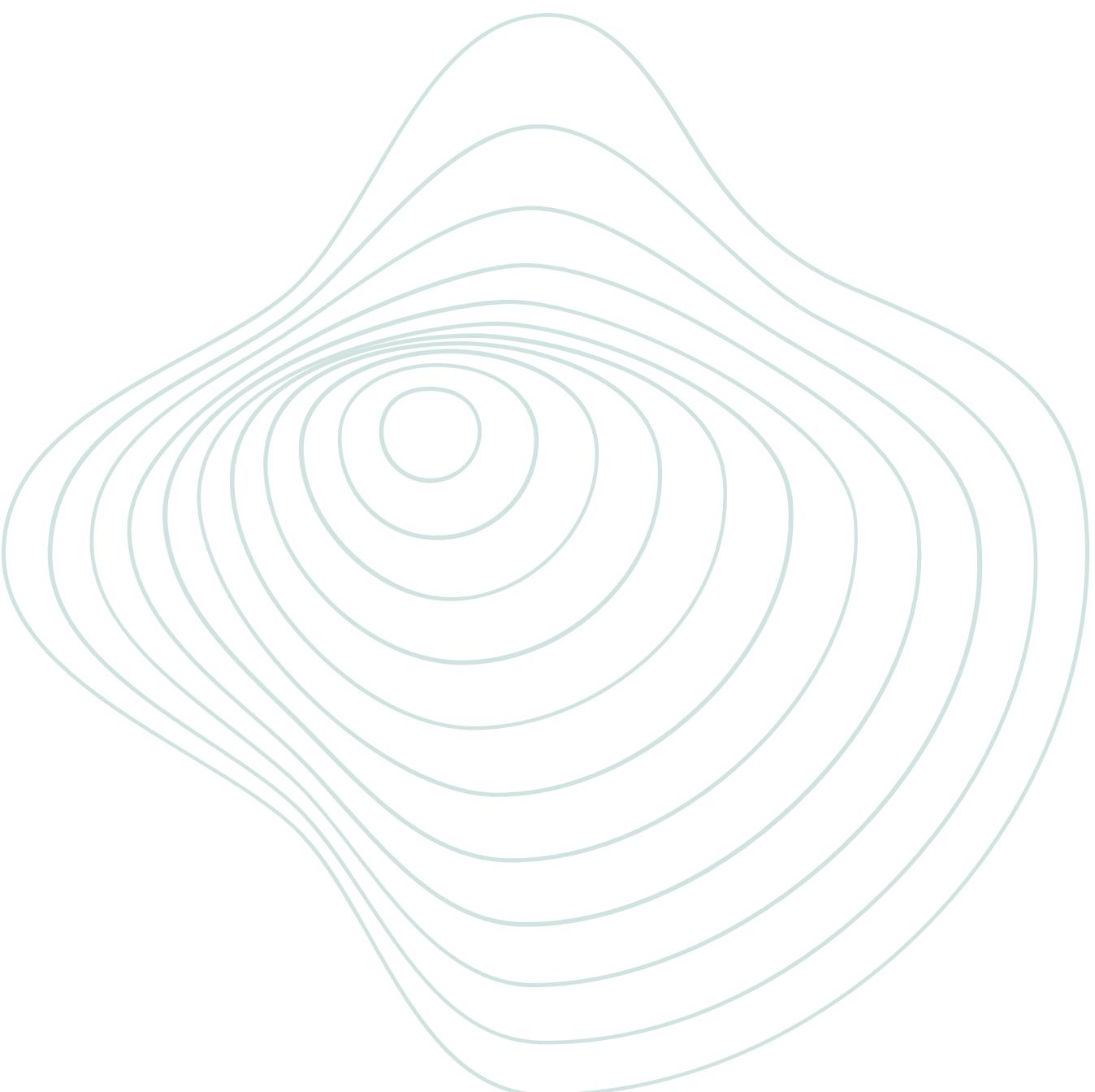
Walkiria Resende - walkiriaresende@gmail.com

Redes Neurais

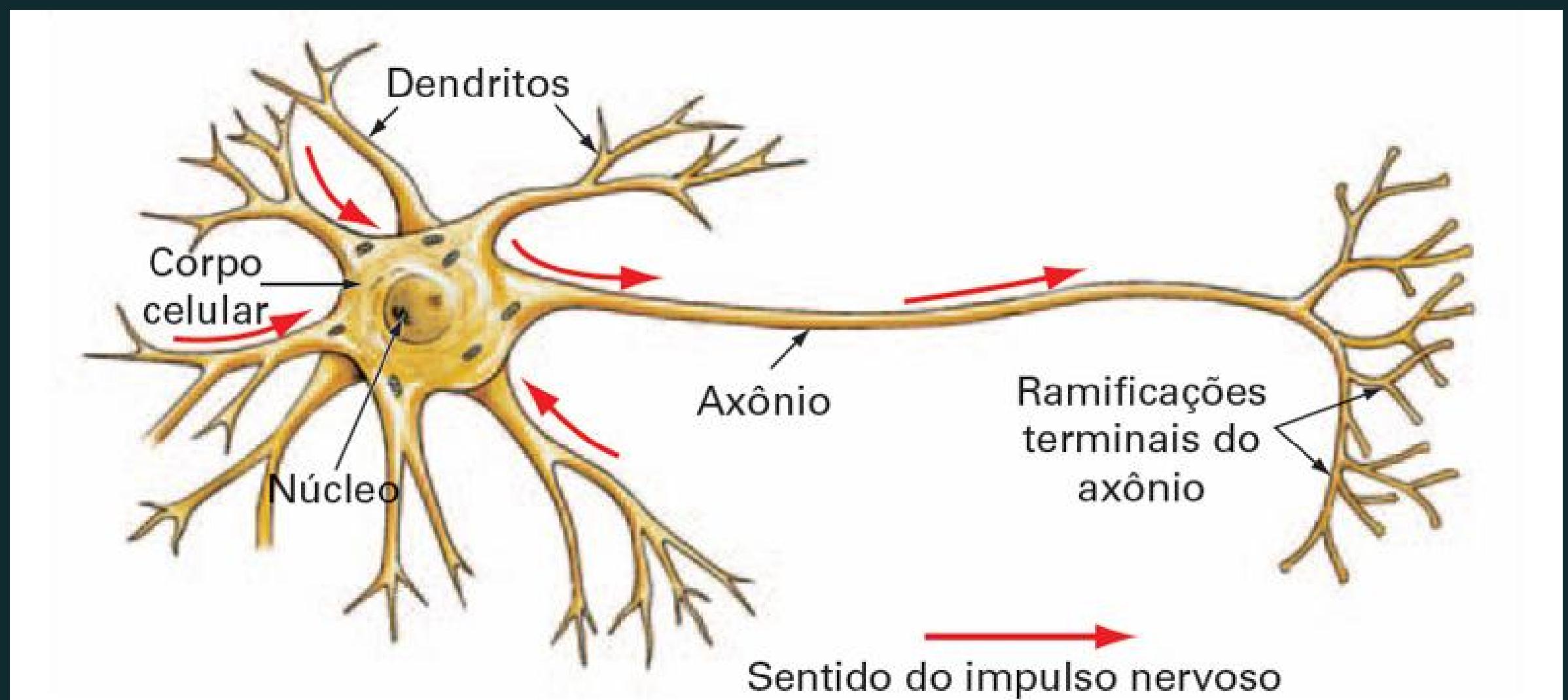
o que você precisa saber

Entenda como as redes neurais funcionam:

- Um pouco de biologia
- Modelo Matemático - McCulloch e Walter Pitts (1943)
- Implementação - Rosenblat (1962)
- Uma dose de realidade - Problemas não lineares
- Um pouco de matemática
- Um pouco de código
- Conclusões



Um pouco de biologia



Modelo Matemático

MCCULLOCH E PITTS - 1943

$$\sigma(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_{k=1}^n x_k > \Theta \text{ and } i = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Um conjunto desses neurônios poderiam computar diversas funções

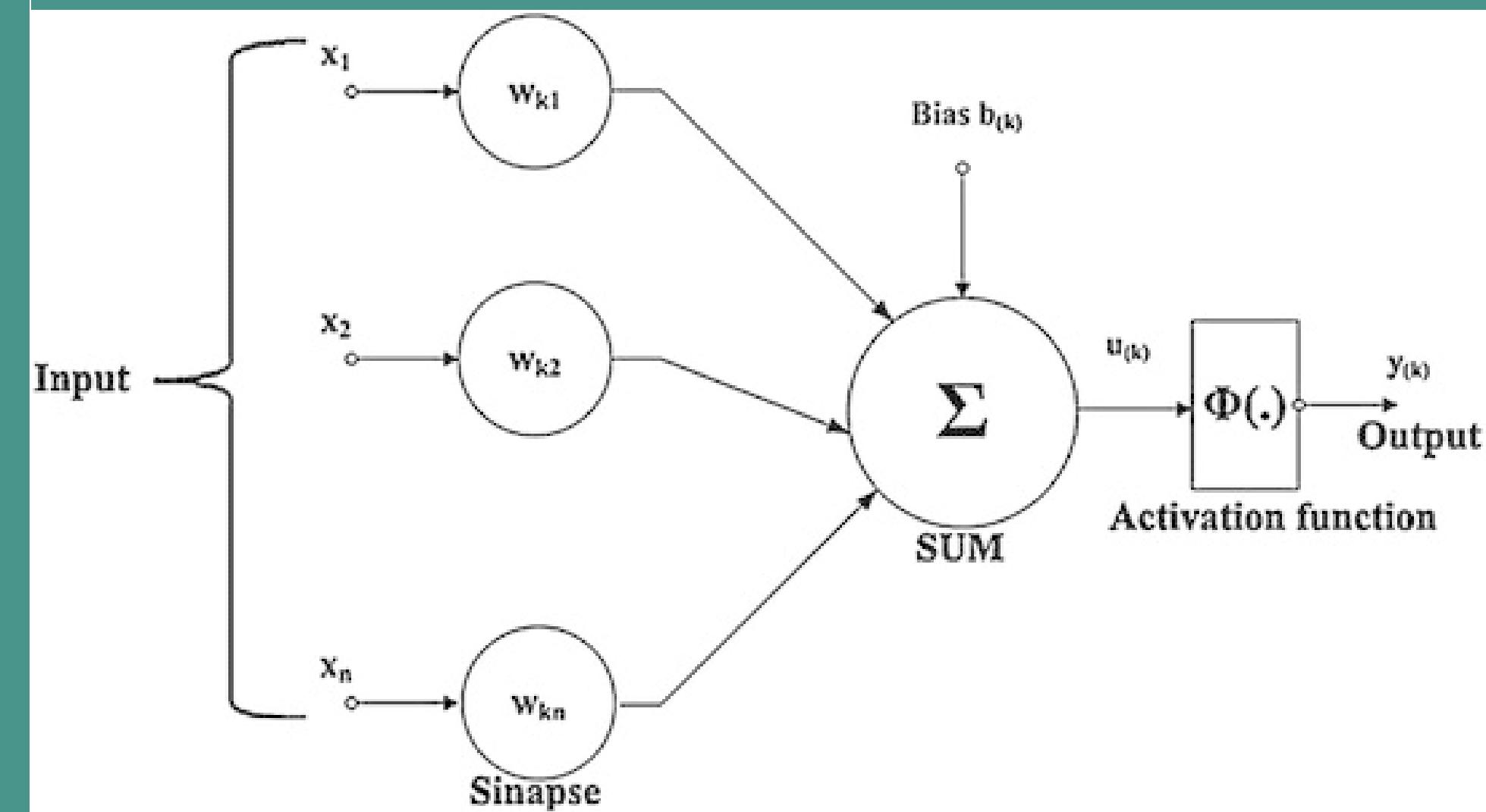
Modelo sem capacidade de aprendizado

Caráter binário inputs e outputs binários

Todos os pesos eram iguais

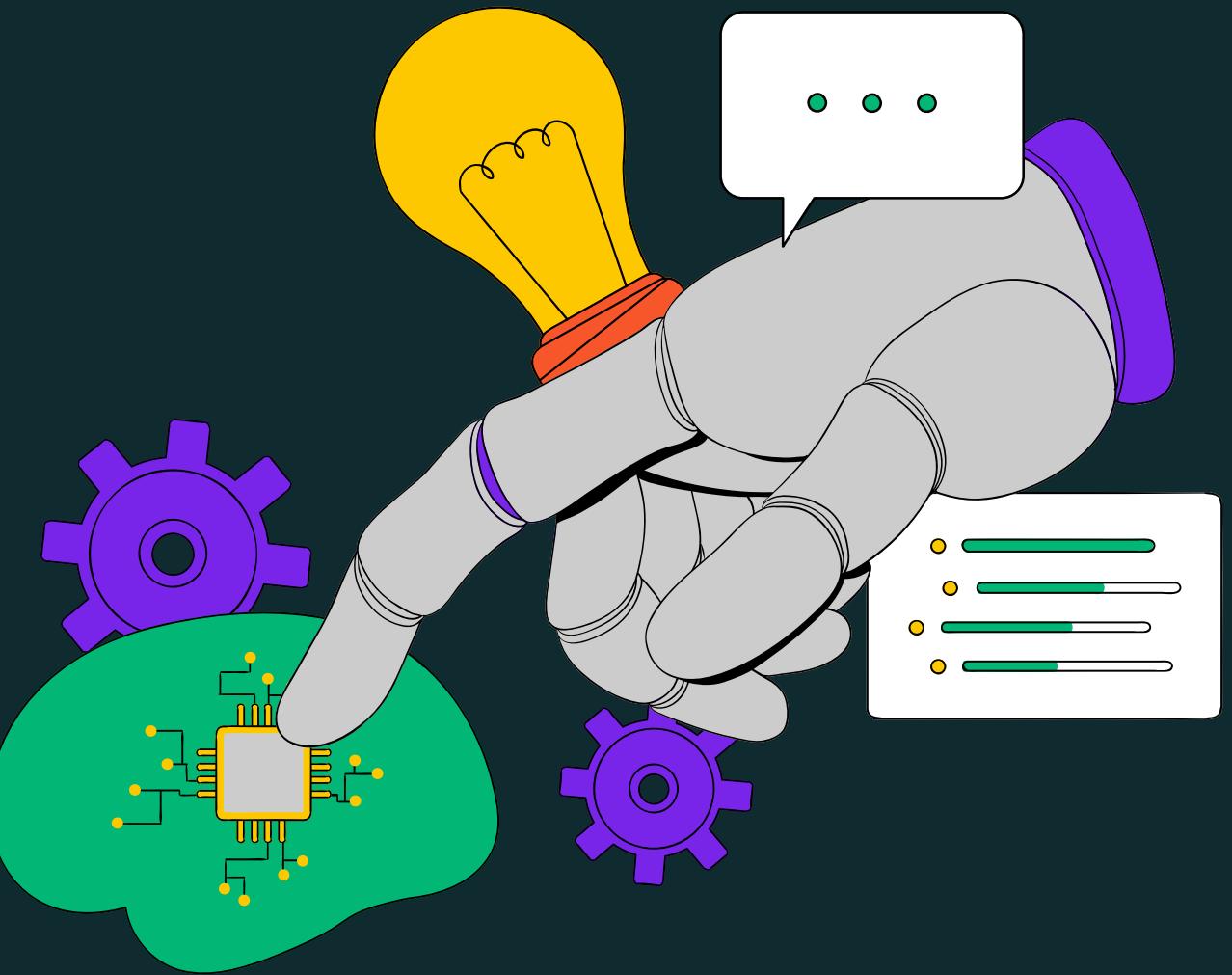
15 anos depois: Implementação

- Relaxamento do conceito de inibição absoluta
- Contribuição de maneira diferente das entradas
- Introduziu o conceito de aprendizagem ao recalcular os pesos



Um grande entusiamo

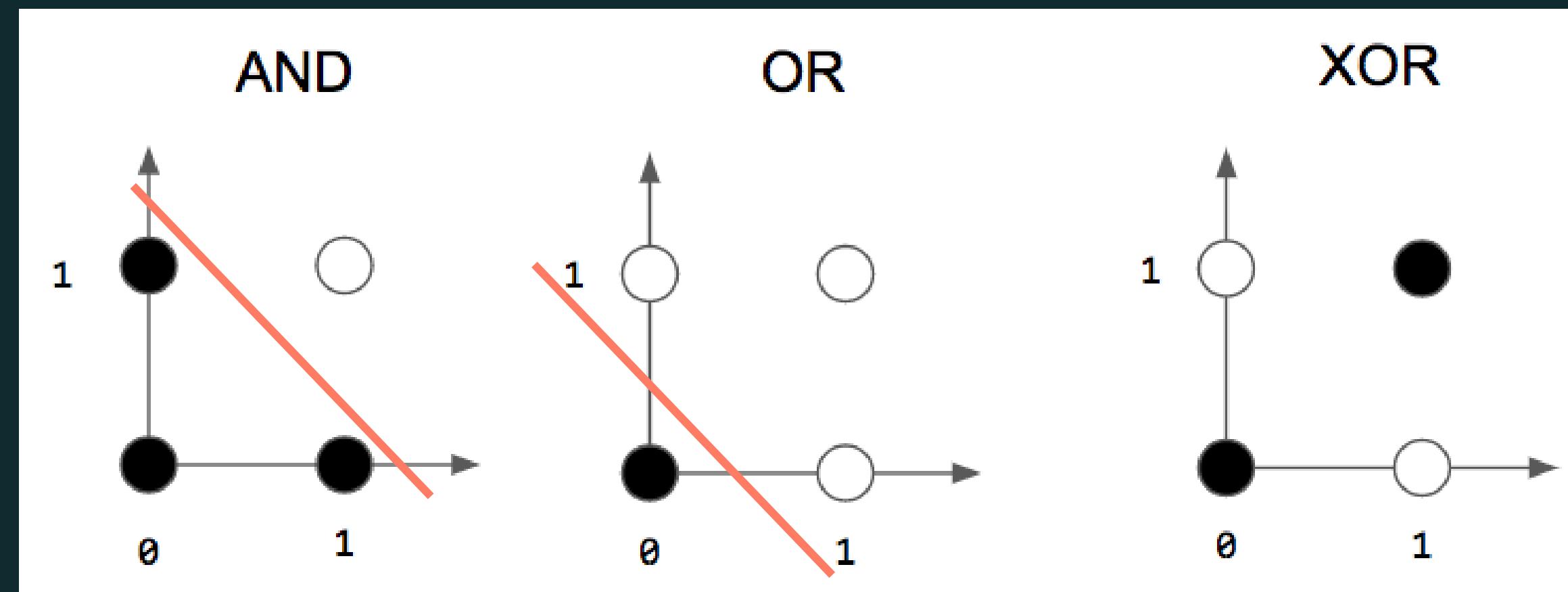
"Olhe mamãe, sem as mãos!"



The winter ...

1966-1973

- Computadores primitivos - Explosão de combinações
- Pesquisas que falharam - Tradutor
- Problemas não lineares



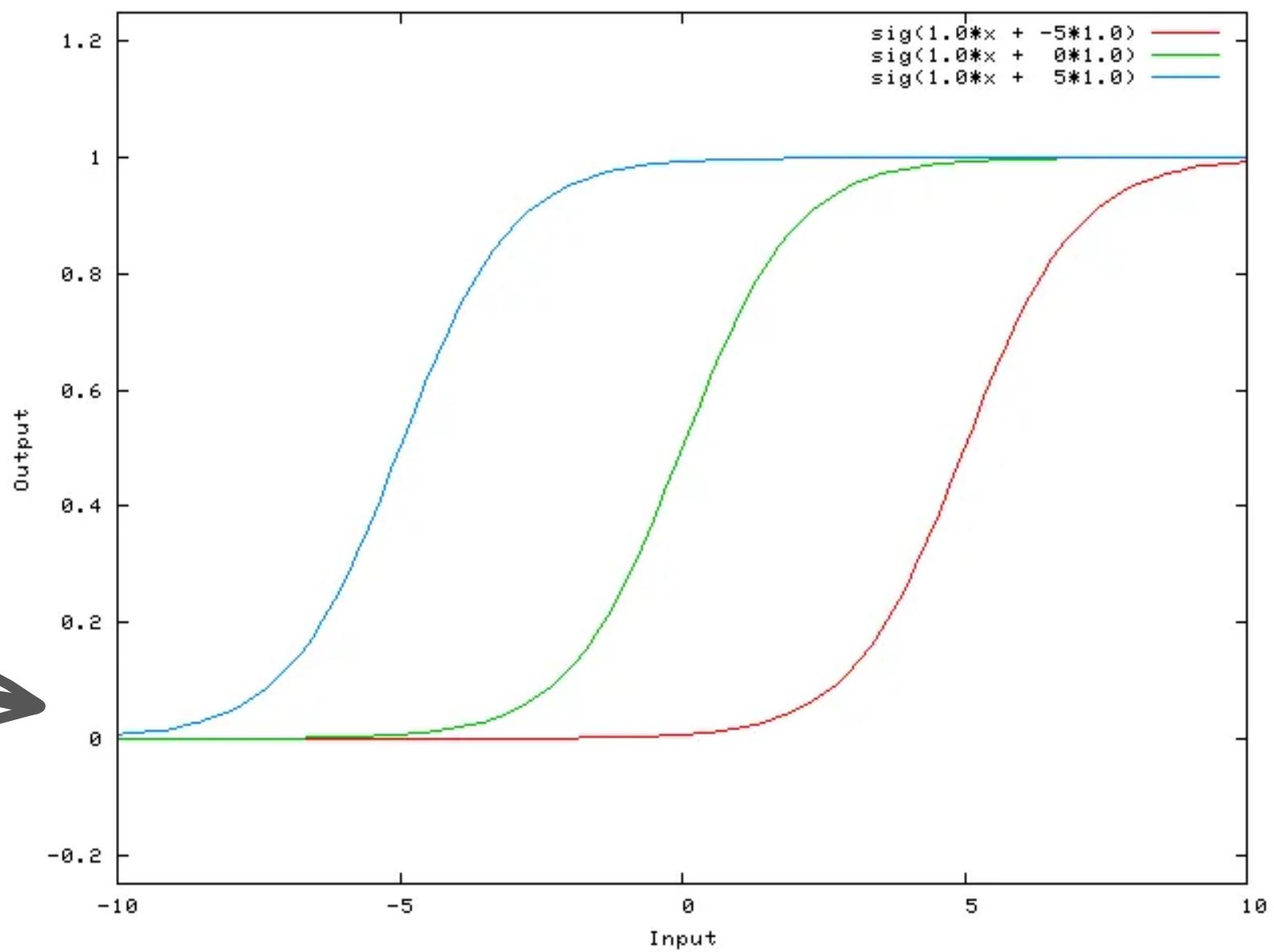
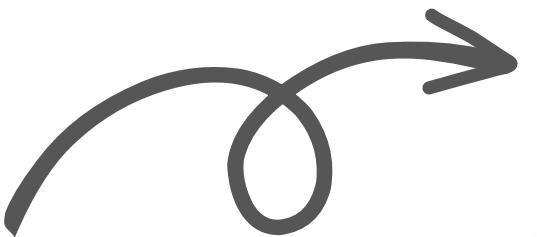
Matemática

1 - Para cada input, multiplique o valor de x_i pelo peso w_i e some todos os valores multiplicados:

$$\sum_{i=1}^n = (x_1, w_1) + (x_2, w_2) + \dots + (x_n, w_n)$$

2 - Bias: aumenta o grau de liberdade da função e possibilita que um neurônio apresente saída não nula

$$\sum_{i=1}^n = (x_1, w_1) + (x_2, w_2) + \dots + (x_n, w_n) + b$$



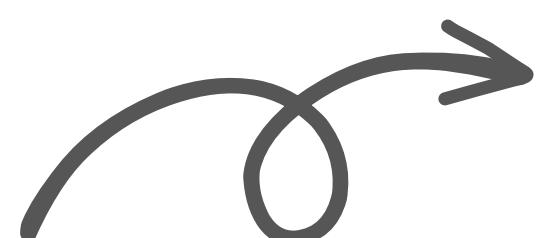
Matemática

3 - Função de Ativação: limita a saída de um neurônio em um intervalo de valores.

Sigmoide: predizer probabilidade

Softmax: multclasse

Relu: fáceis de otimizar



Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) \underset{x=0}{\text{is undefined}} \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
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		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Matemática

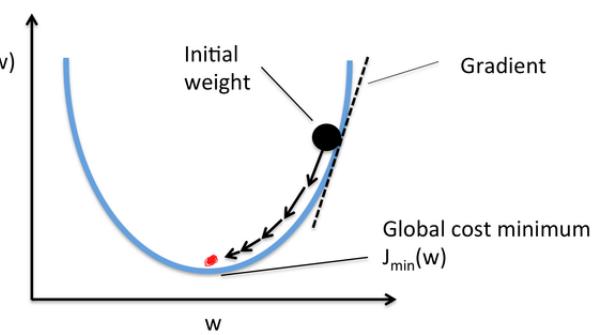
4- Aprendizado: atualizar os pesos!

Para isso, precisamos calcular o quanto a rede está errando:

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

Usamos o Gradiente Descendente para encontrar o minimo global da função

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$



$$w_i = w_i - (\alpha \times \frac{\partial C}{\partial w_i})$$

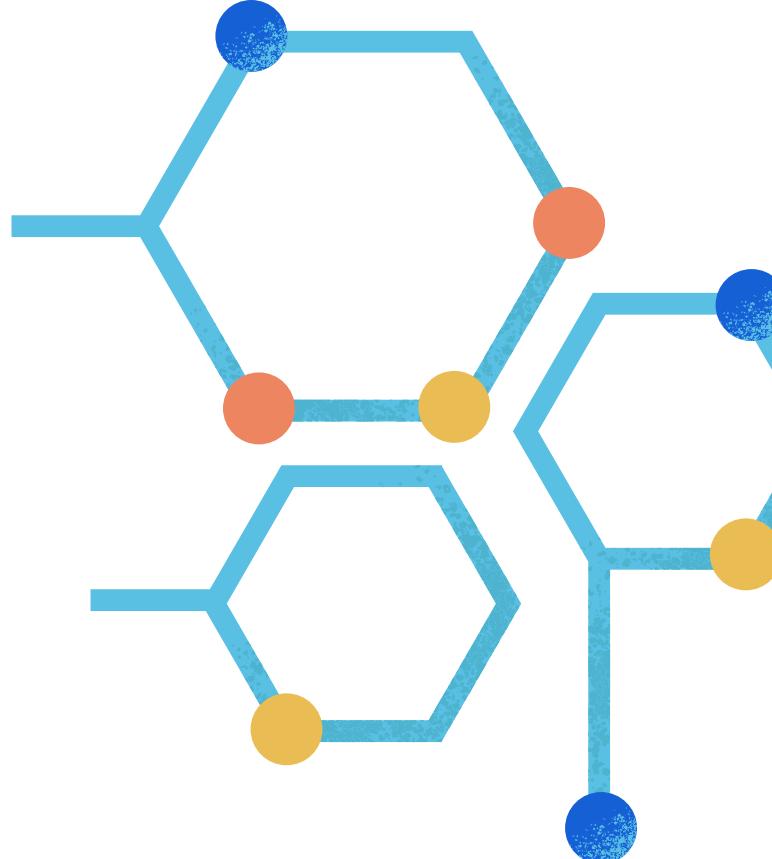
$$b = b - (\alpha \times \frac{\partial C}{\partial b})$$

$$\frac{\partial C}{\partial w_i} = \frac{2}{n} \times \text{sum}(y - \hat{y}) \times \sigma(z) \times (1 - \sigma(z)) \times x_i$$

$$\frac{\partial C}{\partial b} = \frac{2}{n} \times \text{sum}(y - \hat{y}) \times \sigma(z) \times (1 - \sigma(z))$$

Vamos implementar em Python?

```
1 import pandas as pd  
2 import numpy as np  
3 import matplotlib.pyplot as plt  
4 import random
```

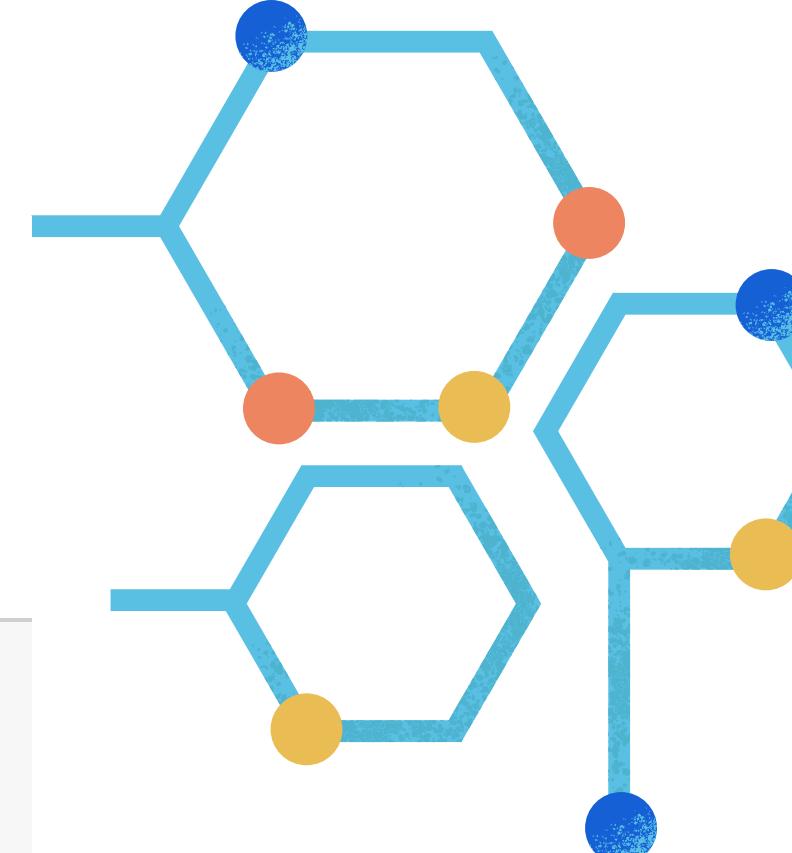


1 Caso

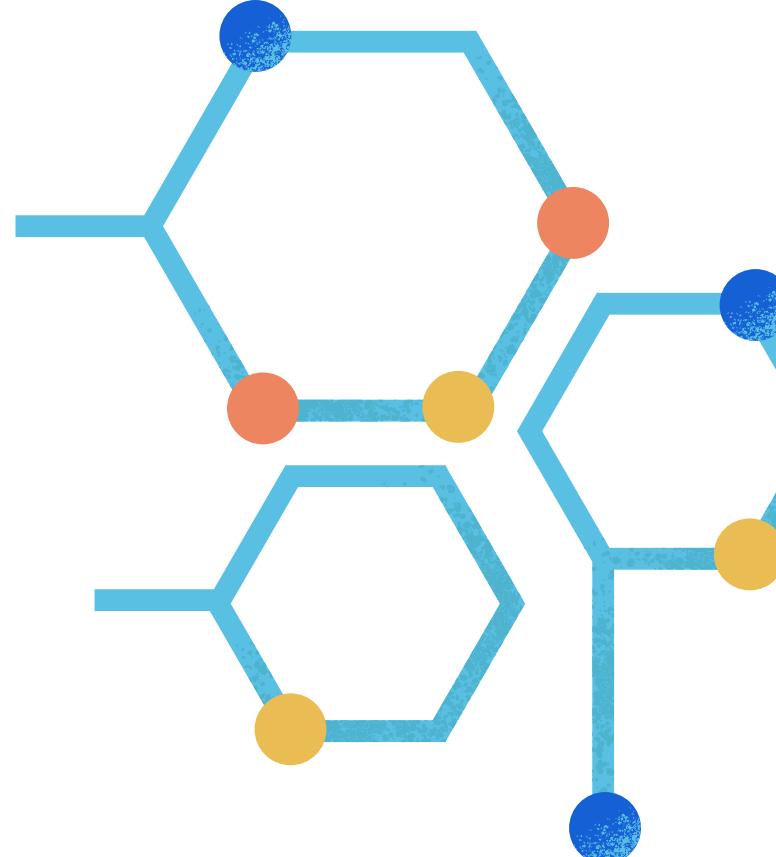
Vamos implementar o neurônio sem aprendizado, passando os pesos previamente conhecidos

```
1 # criando um dataset ficticio
2 dataset = np.array([[1,1,1], [-1,-1,0]])
3 df = pd.DataFrame(dataset, columns=['var_1', 'var_2', 'classe'])
4 df
```

	var_1	var_2	classe
0	1	1	1
1	-1	-1	0



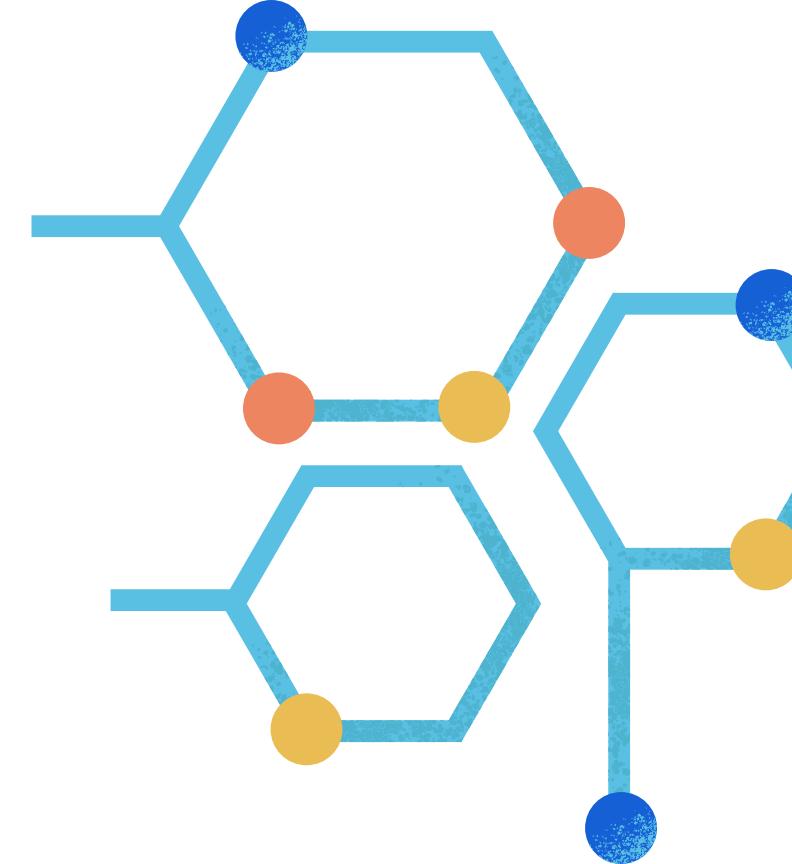
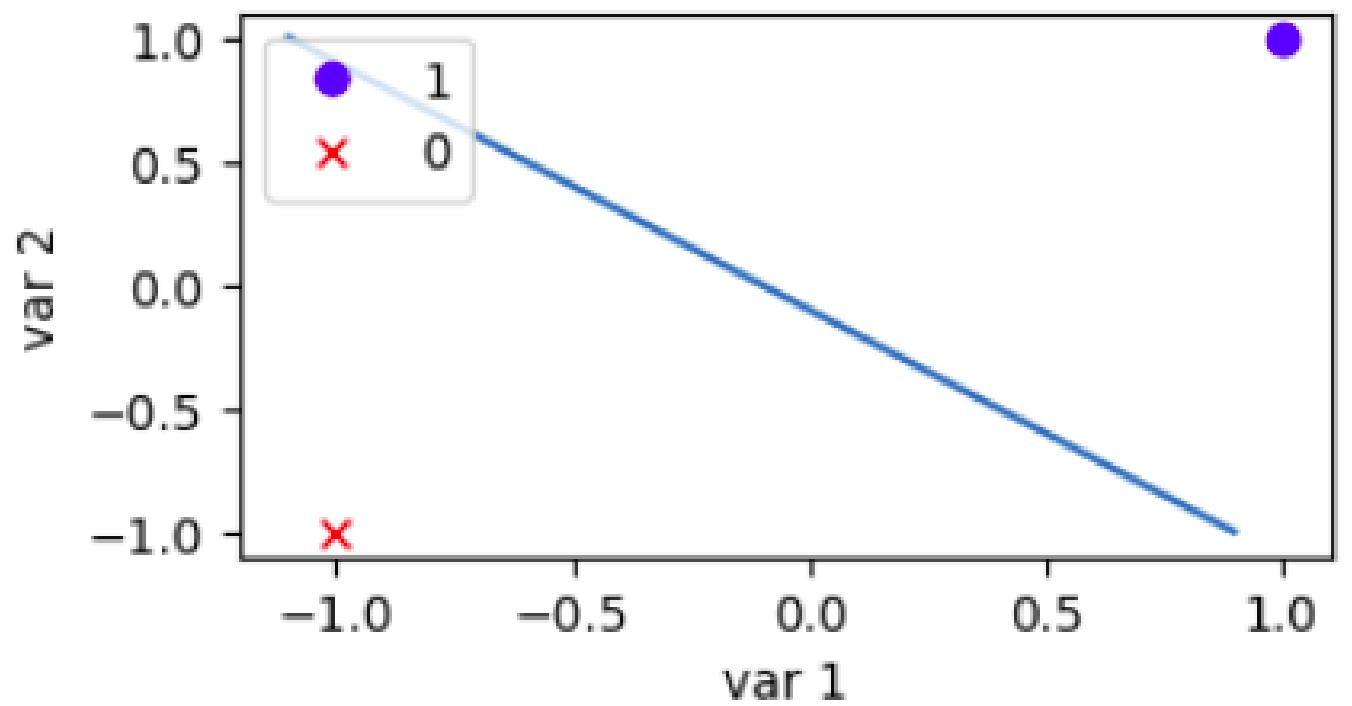
```
1 # separando as classes para o plot
2 classe_1 = df[0:1]
3 classe_0 = df[1:]
```



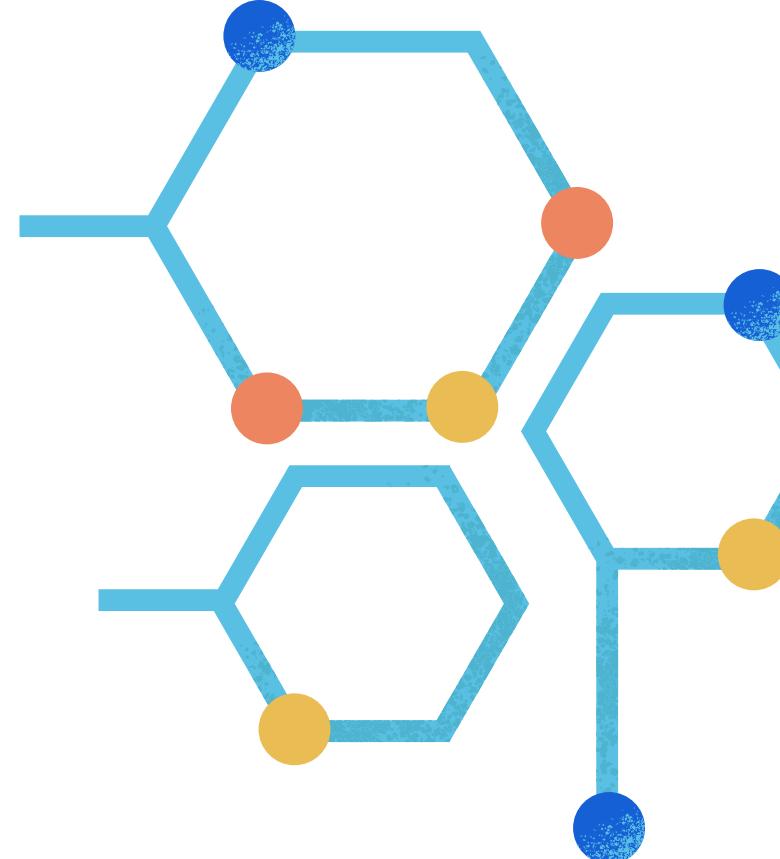
```

1 # criando uma reta qualquer para separar os pontos
2 y = np.array(list(range(10,-11, -1)))/10
3 x = np.array(list(range(-11,10)))/10
4
5 #plot points where blue o's denote classe 1 & red x's denote classe 0
6 plt.figure(figsize=(4,2))
7 plt.plot(classe_1['var_1'], classe_1['var_2'], "bo", label="1")
8 plt.plot(classe_0['var_1'], classe_0['var_2'], "rx", label="0")
9 plt.plot(x, y, '-')
10 plt.xlabel("var 1")
11 plt.ylabel("var 2")
12 plt.legend(loc='upper left')
13 plt.show();

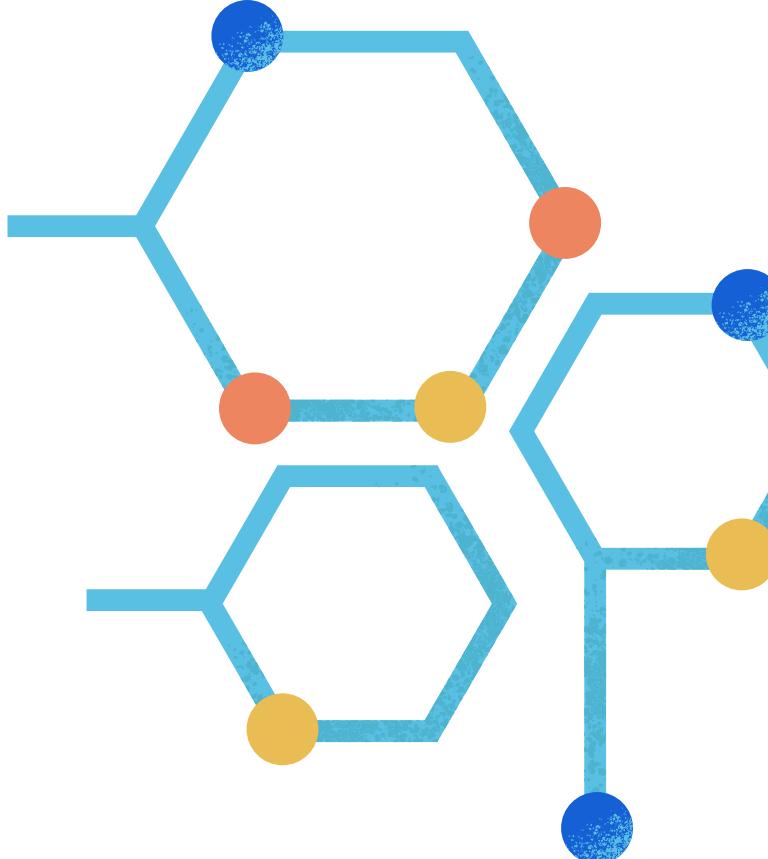
```



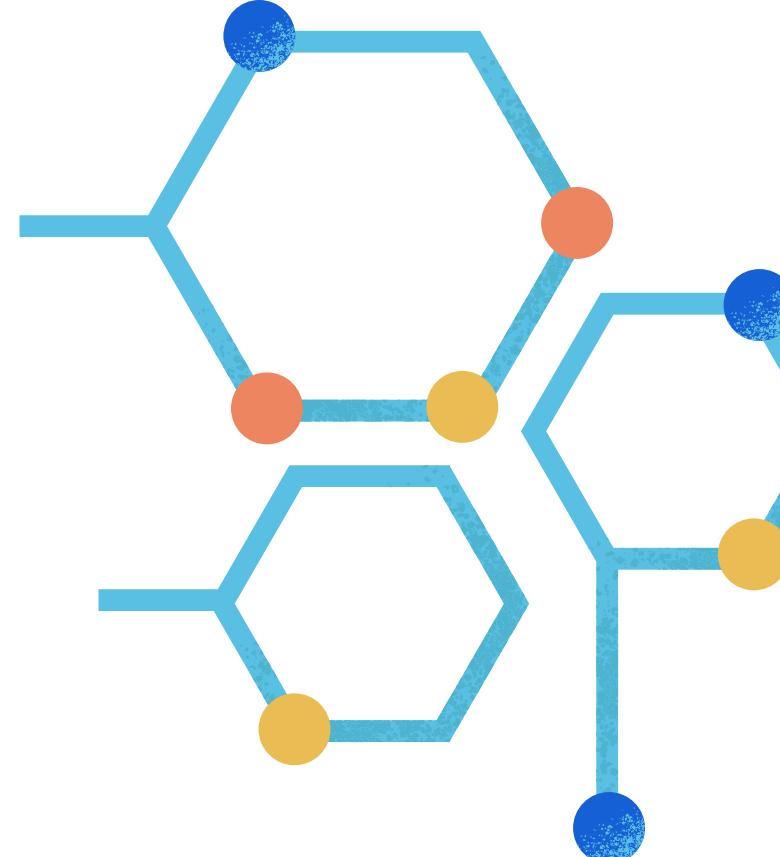
```
1 # separando os dados em variáveis de entrada e target  
2 X = df[['var_1', 'var_2']]  
3 y = df[['classe']]
```



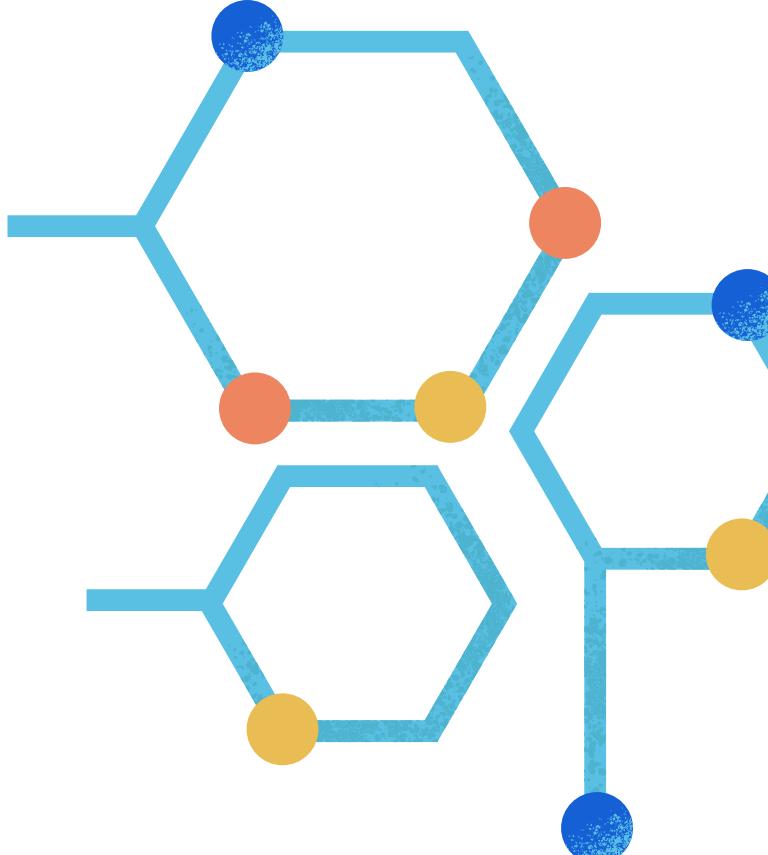
```
1 # ja sabemos os pesos  
2 pesos = [-0.5, 1, 0.5]
```



```
1 predict = []
2 for i in range(0,len(X)):
3     predict.append(perceptron(X.iloc[i], pesos))
```

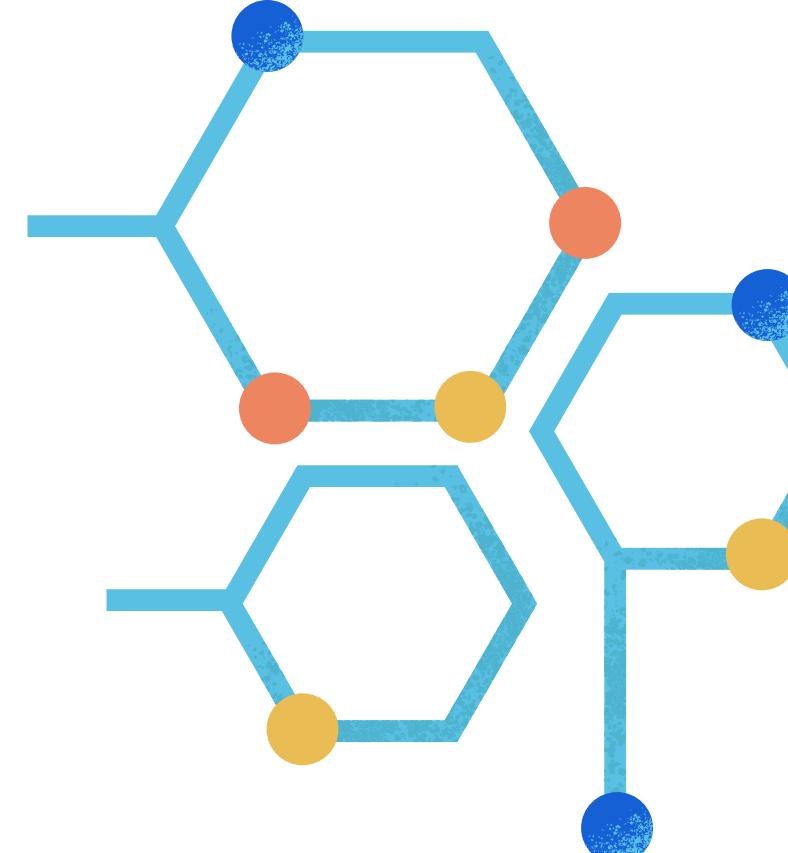


```
1 def perceptron( row, pesos ):  
2     ativacao = pesos[0]  
3     for j in range(0, len(row) ):  
4         ativacao += pesos[j+1] * row[j]  
5     return 1 if ativacao >=1 else 0
```



```
1 print('Predido: ', predict, 'Esperado: ', y.values.tolist())
```

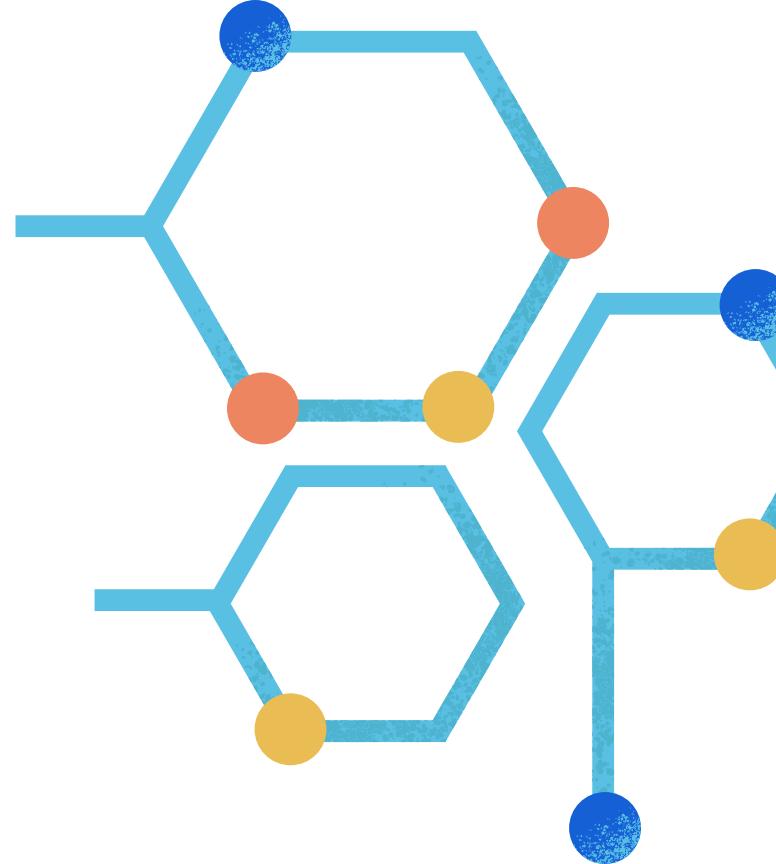
```
Predido: [1, 0] Esperado: [[1], [0]]
```



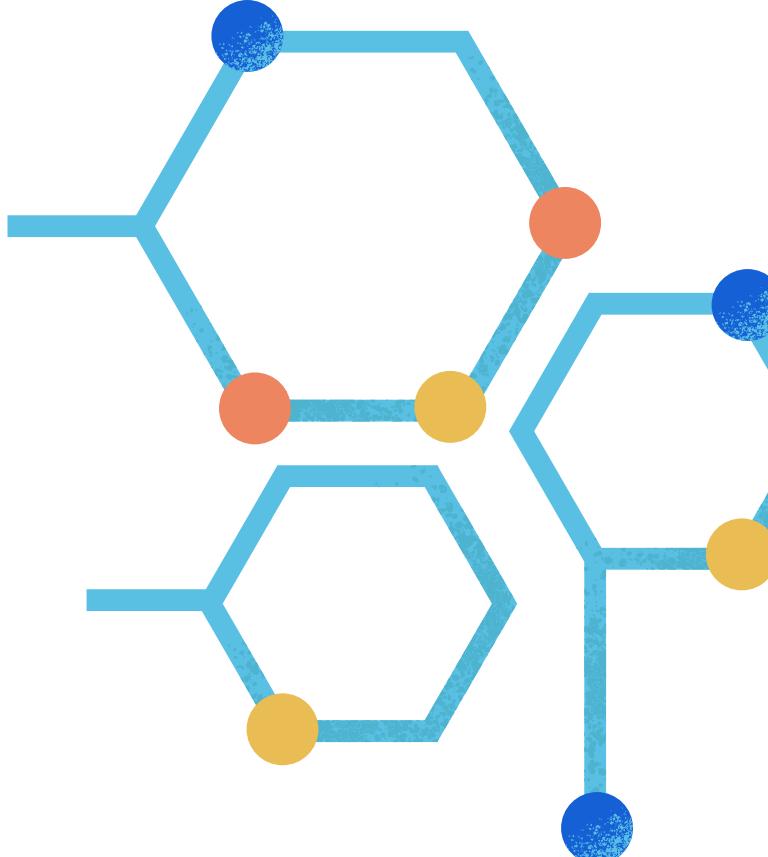
2 Caso

Vamos calcular os pesos de acordo com as entradas

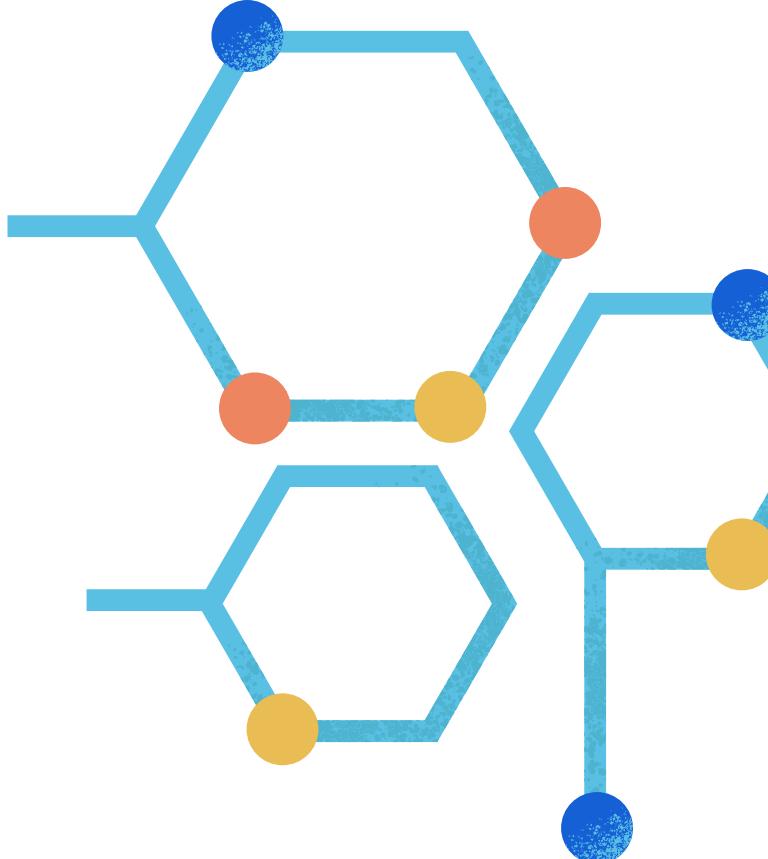
```
: 1 df = pd.DataFrame(np.array([[0,0,0],[0,1,1], [1,0,1], [1,1,1]]), columns=['var1','var2', 'classe'])
 2 df
:
  var1  var2  classe
0      0      0       0
1      0      1       1
2      1      0       1
3      1      1       1
```



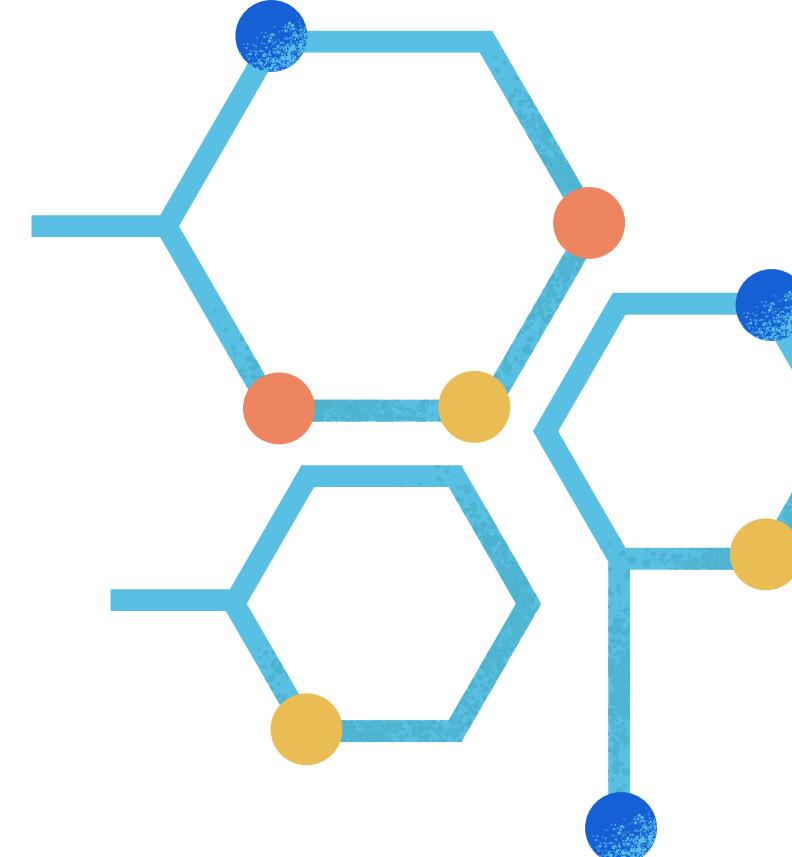
```
1 # separando os dados em variáveis de entrada e target  
2 X = df[['var1', 'var2']]  
3 y = df['classe']
```



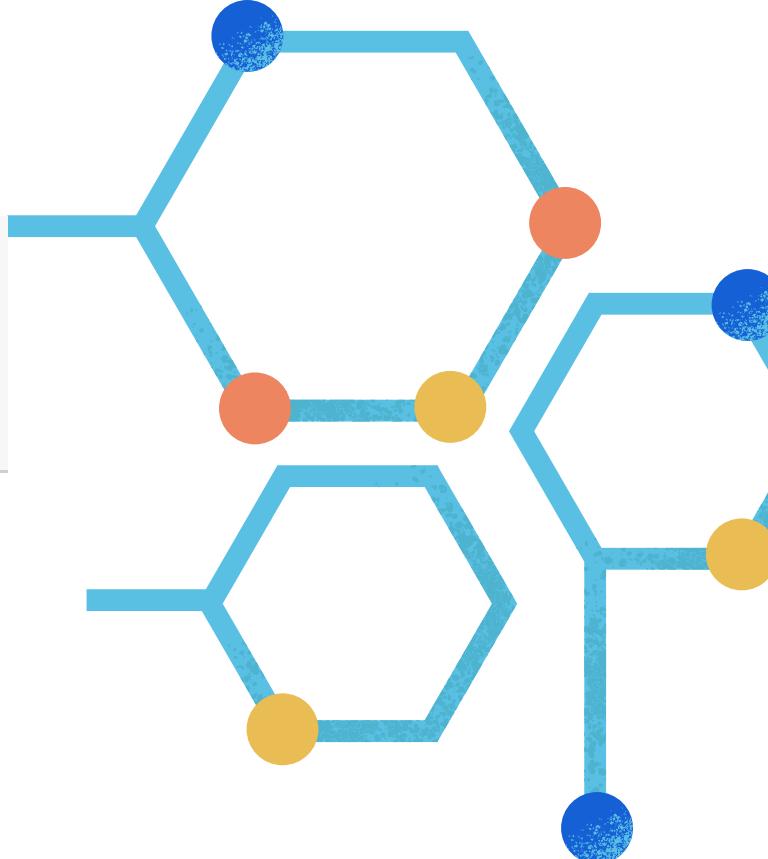
```
1 # formatando as entradas, inicializando os pesos e a taxa de aprendizagem
2 entradas = np.array(X)
3 saidas = np.array(y)
4 pesos = np.array([0.0, 0.0])
5 taxaAprendizagem = 0.1
```



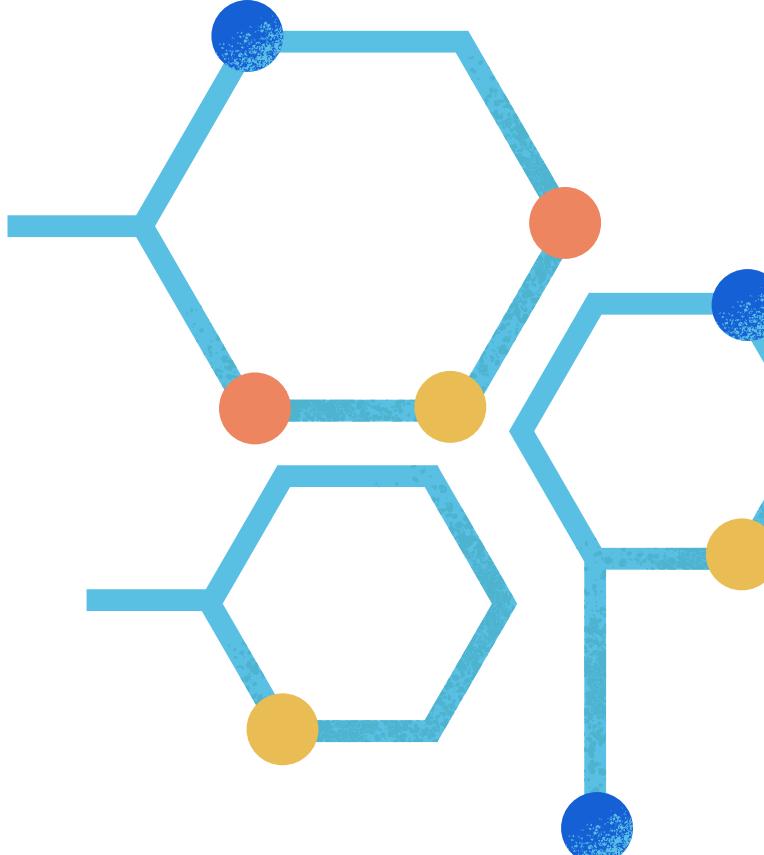
```
1 # função que receberá os valores e fará a predição
2 def perceptron_predict(row, pesos):
3     saída = soma(row, pesos)
4     return step(saída)
```



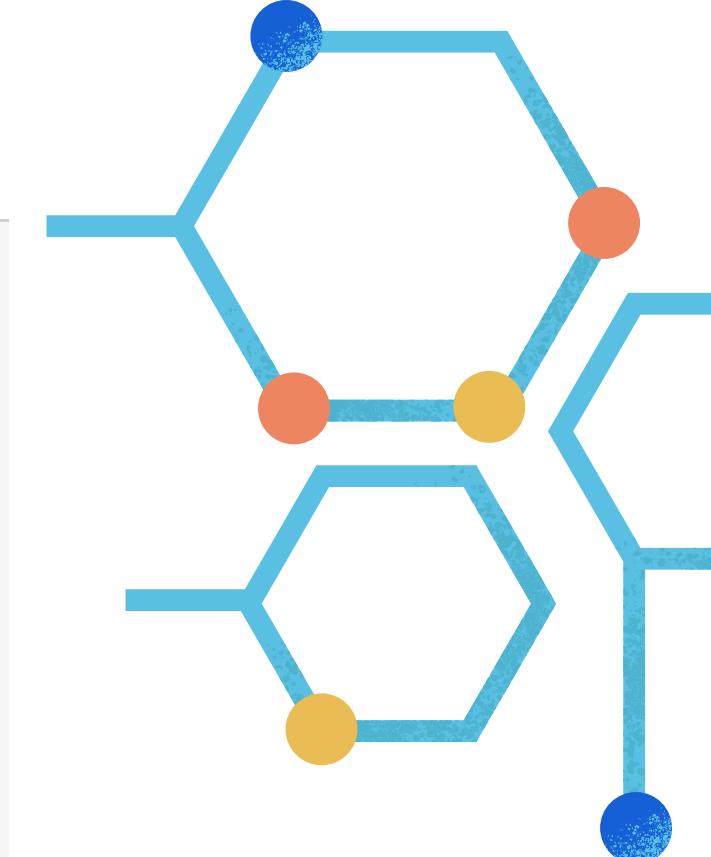
```
v 1 # função que combina linearmente entradas e pesos  
v 2 def soma(row, pesos):  
 3     soma = row.dot(pesos) # é o mesmo que multiplicar 1 por 1, como fizemos acima usando um for  
 4     return soma
```



```
1 # função de ativação step
2 def step(soma):
3     if (soma >= 1):
4         return 1
5     return 0
```

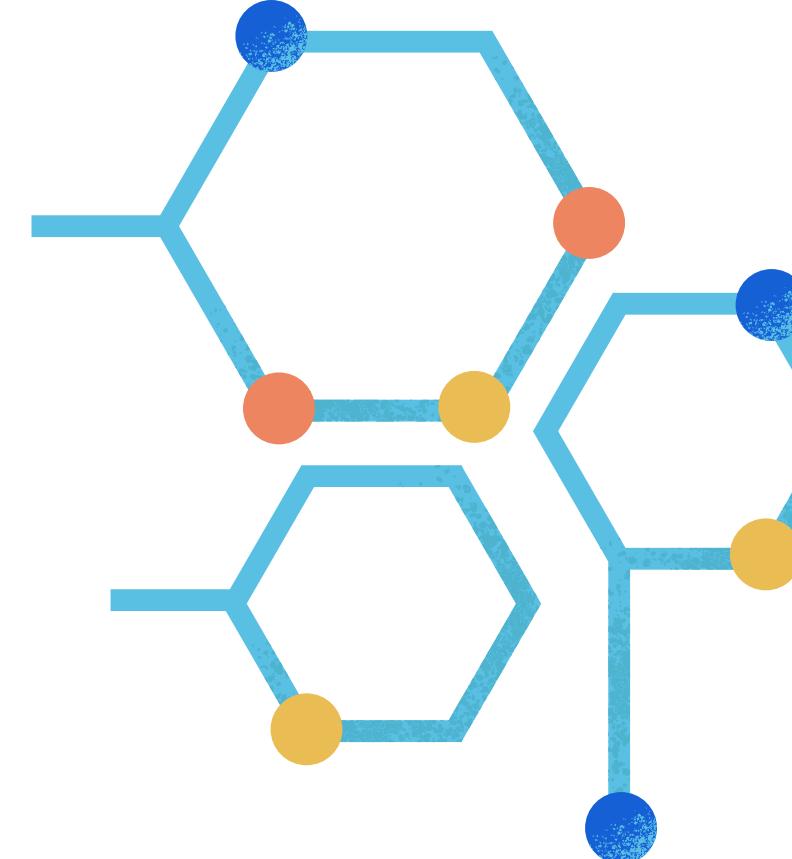


```
1 def perceptron_fit(entradas, saidas, pesos, taxaAprendizagem ):  
2     erroTotal = None  
3     #enquanto o erro não for igual à 0 (zero)  
4     while (erroTotal != 0):  
5         erroTotal = 0  
6         #faça o ajuste dos pesos para cada uma das nossas classes  
7         for i in range(len(saidas)):  
8             saidaCalculada = perceptron_predict(np.asarray(entradas[i]), pesos)  
9             #Calcula o erro da nossa classificação  
10            erro = saidas[i] - saidaCalculada  
11            erroTotal += erro  
12            #para cada um dos pesos: atualize o valor dele com base no nosso erro  
13            for j in range(len(pesos)):  
14                pesos[j] = pesos[j] + (taxaAprendizagem * entradas[i][j] * erro)  
15                print('Peso atualizado: ' + str(pesos[j]))  
16            print('Total de erros: ' + str(erroTotal))  
17        return pesos
```



```
1 print(perceptron_predict(entradas[0], pesos))  
2 print(perceptron_predict(entradas[1], pesos))  
3 print(perceptron_predict(entradas[2], pesos))  
4 print(perceptron_predict(entradas[3], pesos))
```

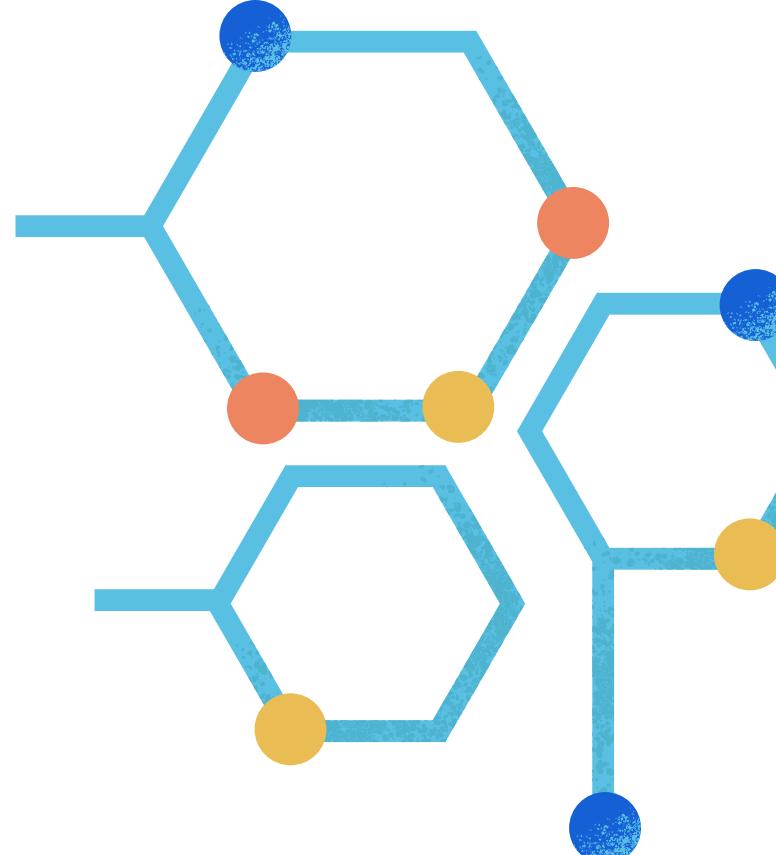
0
1
1
1



3 Caso

Vamos aplicar a um caso mais proximo da realidade?

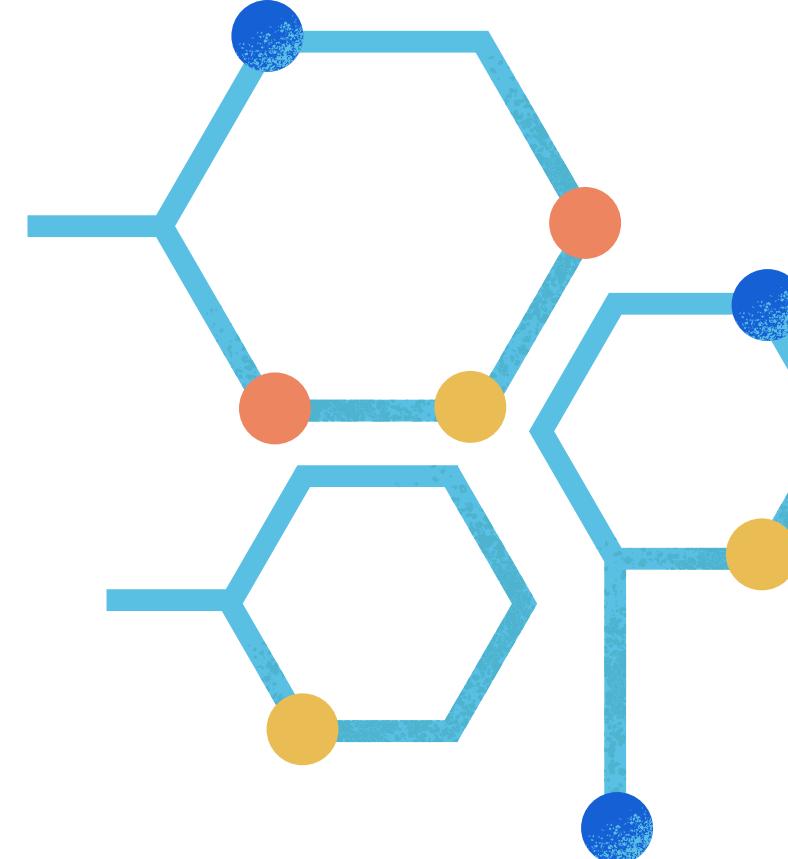
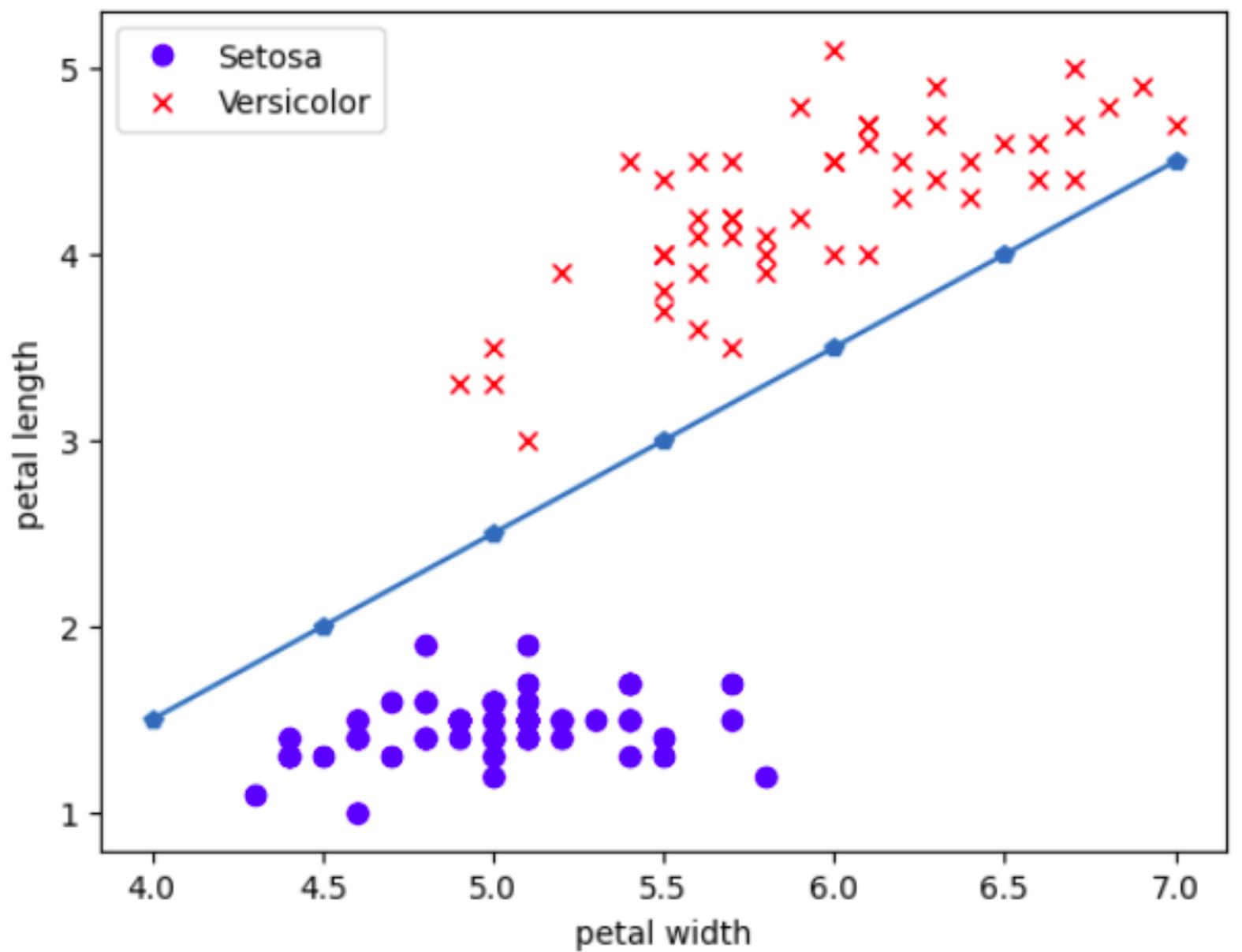
```
1 # leitura do dataset
2 df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', header=None)
3
4 #vamos usar apenas duas classes, setosas e versicolor
5 setosa = np.array(df.iloc[0:50, [0,2]])
6 versicolor = np.array(df.iloc[50:100, [0,2]])
```



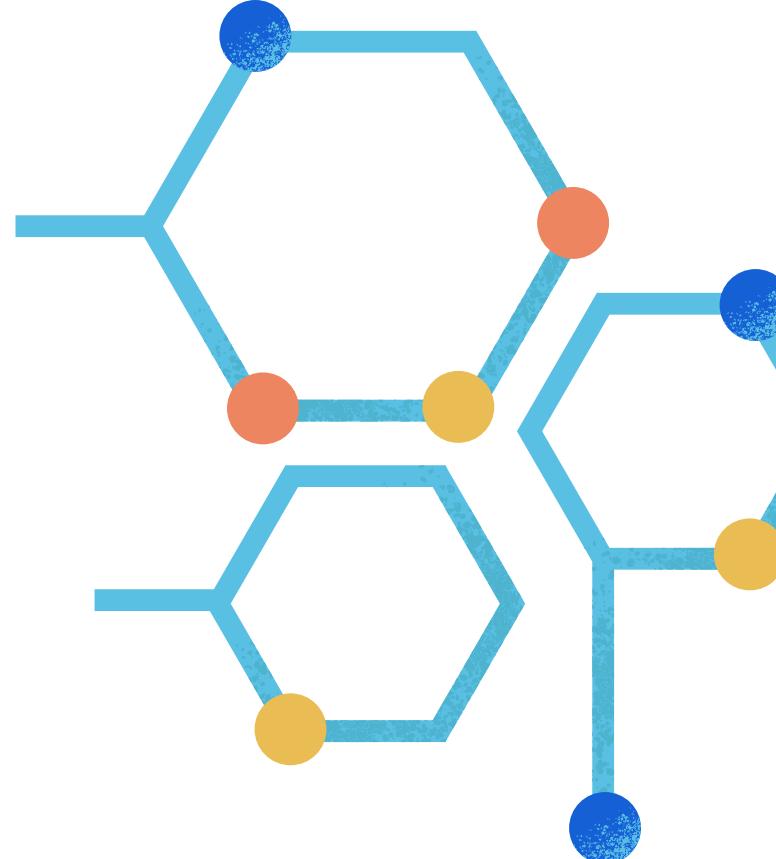
```

1 # criando uma reta qualquer para separar os pontos
2 y = np.arange(1.5, 5, 0.5)
3 x = np.arange(4, 7.5, 0.5)
4
5 #plot onde azul denota setosa e red denota versicolor
6 plt.plot(setosa[:, 0], setosa[:, 1], "bo", label="Setosa")
7 plt.plot(versicolor[:, 0], versicolor[:, 1], "rx", label="Versicolor")
8 plt.plot(x, y, 'p-')
9
10 plt.xlabel("petal width")
11 plt.ylabel("petal length")
12 plt.legend(loc='upper left')
13 plt.show()

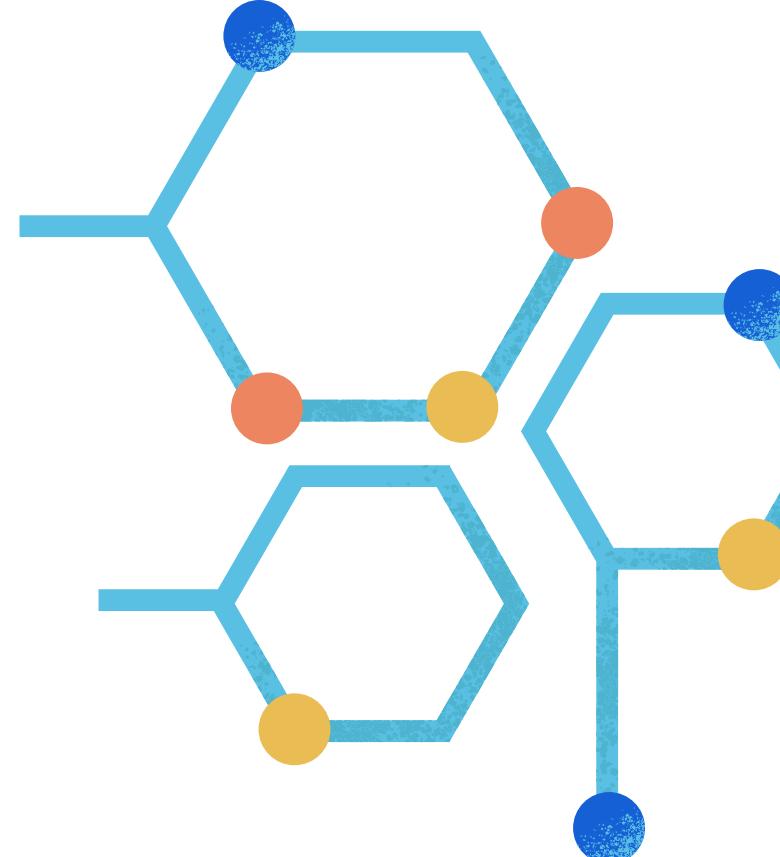
```



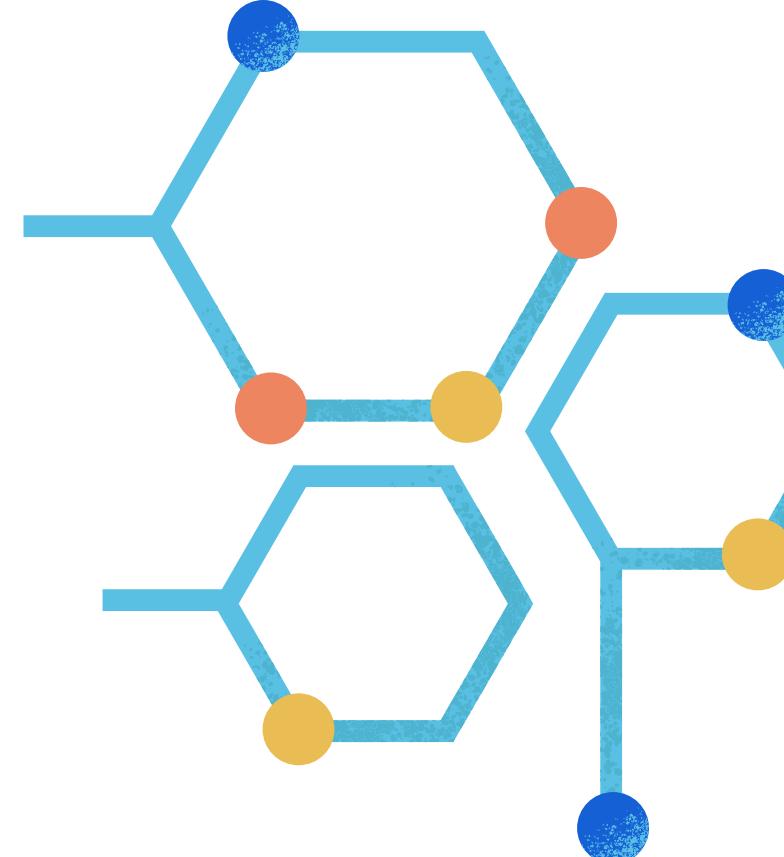
```
1 # separando os dados em variáveis de entrada e target
2 X = df.iloc[0:100, [0,2]].values
3 y = df.iloc[0:100, 4].values
4 y = np.where(y == 'Iris-setosa', -1, 1)
```



```
1 # inicializando os pesos e a taxa de aprendizagem  
2 pesos = np.array([0.0, 0.0])  
3 taxaAprendizagem = 0.1
```



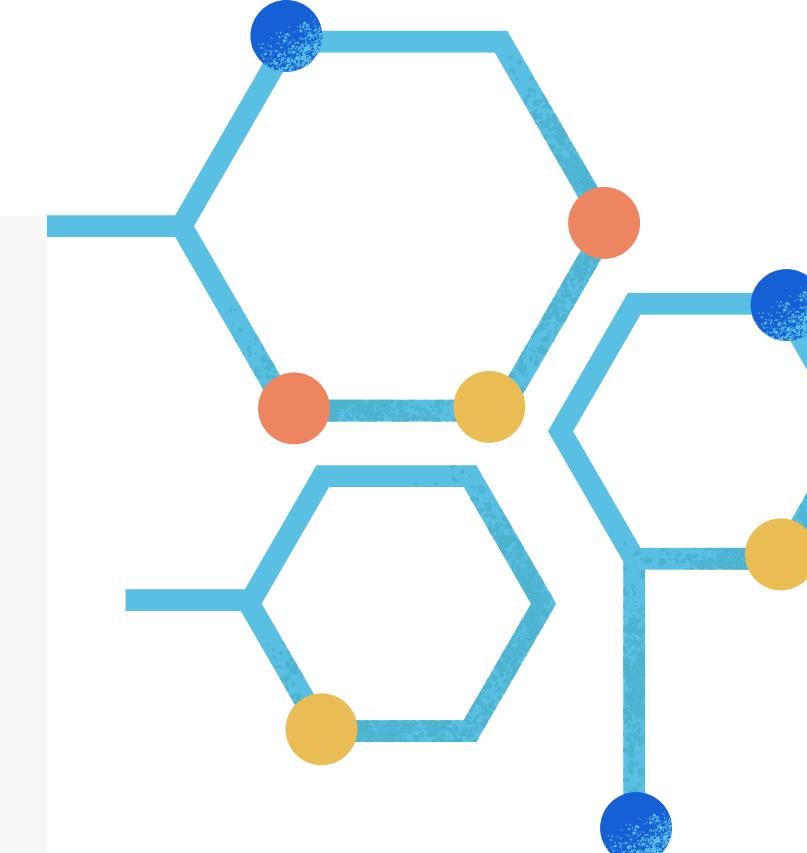
```
1 # modificando a função de ativação
2 def step(soma):
3     if (soma >= 0):
4         return 1
5     return -1
```



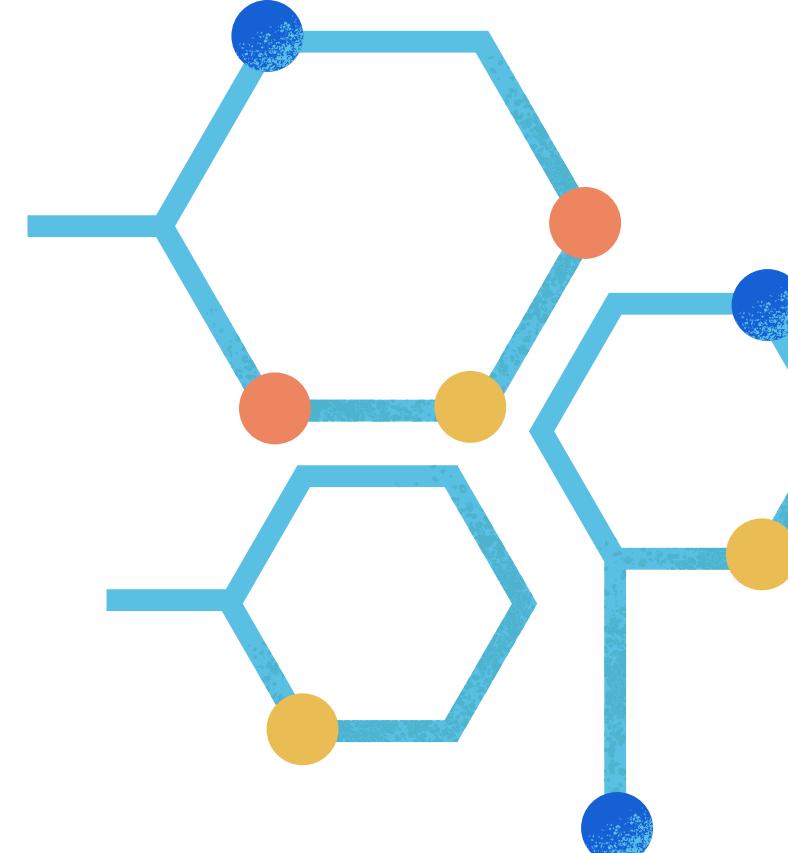
```

1 def perceptron_fit(entradas, saidas, pesos, taxaAprendizagem, niter=10 ):
2     erroTotal = None
3     max_iter = 0
4     #enquanto o erro não for igual à 0 (zero) ou o numero de iterações não for o max
5     while ((erroTotal != 0) or (max_iter < niter)):
6         erroTotal = 0
7         #faça o ajuste dos pesos para cada uma das nossas classes
8         for i in range(len(saidas)):
9             saidaCalculada = perceptron_predict(np.asarray(entradas[i]), pesos)
10            #Calcula o erro da nossa classificação
11            erro = saidas[i] - saidaCalculada
12            erroTotal += erro
13            #para cada um dos pesos: atualize o valor dele com base no nosso erro
14            for j in range(len(pesos)):
15                pesos[j] = pesos[j] + (taxaAprendizagem * entradas[i][j] * erro)
16                print('Peso atualizado: ' + str(pesos[j]))
17                print('Niter', max_iter)
18            print('Total de erros: ' + str(erroTotal))
19            max_iter = max_iter +1
20    return pesos

```

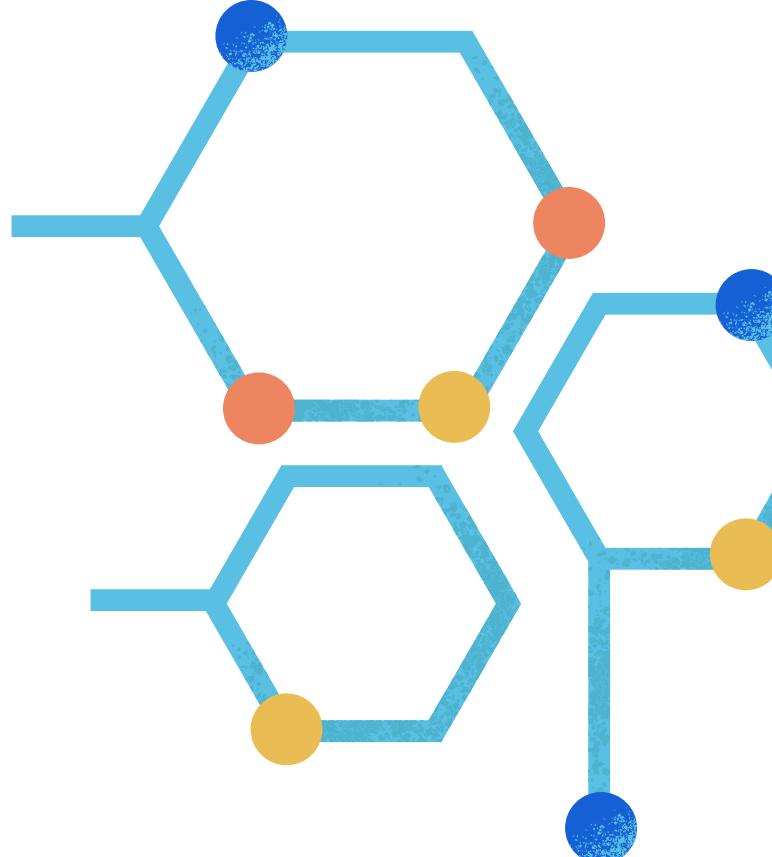


```
1 pesos = perceptron_fit(X, y, pesos, taxaAprendizagem, niter=8)
2 print('Rede neural treinada')
...
Niter 7
Peso atualizado: -0.8599999999999997
Niter 7
Peso atualizado: 1.7400000000000004
Niter 7
Peso atualizado: -0.8599999999999997
Niter 7
Peso atualizado: 1.7400000000000004
Niter 7
Peso atualizado: -0.8599999999999997
Niter 7
Peso atualizado: 1.7400000000000004
Niter 7
Peso atualizado: -0.8599999999999997
Niter 7
Peso atualizado: 1.7400000000000004
Niter 7
Total de erros: 0
Rede neural treinada
```



```
1 print(perceptron_predict(X[0], pesos))  
2 print(y[0])
```

-1
-1



Usando o sklearn

4 Caso

Usando o sklearn

```
1 from sklearn.datasets import load_digits
2 from sklearn.linear_model import Perceptron
3 X, y = load_digits(return_X_y=True)
4 clf = Perceptron(tol=1e-3, random_state=0)
5 clf.fit(X, y)
6
7 clf.score(X, y)
```

```
1 print('Predito:', clf.predict(X[[0]]))
2 print('Real:', y[0])
```

Predito: [0]

Real: 0

Referências

- <https://towardsdatascience.com/rosenblatts-perceptron-the-very-first-neural-network-37a3ec09038a>
- <https://www.deeplearningbook.com.br>
- <https://towardsdatascience.com/how-to-choose-the-right-activation-function-for-neural-networks-3941ff0e6f9c>
- <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>
- <https://towardsdatascience.com/introduction-to-math-behind-neural-networks-e8b60dbbdeba>
- <http://neuralnetworksanddeeplearning.com/>
- <https://mml-book.github.io/book/mml-book.pdf>
- <https://medium.com/ensina-ai/redes-neurais-roots-1-introdu%C3%A7%C3%A3o-ffdd6f8b9f01>
- <https://www.youtube.com/watch?v=9uS0qiMeZu0>
- **Livro: Inteligência Artificial, Peter Norvig e Stuart Russel**



Walkiria Resende

walkiriaresende@gmail.com

<https://www.linkedin.com/in/walkiria-resende-vieira/>