

# Redes Neurais Artificiais (RNA)

Prof. André Gustavo Hochuli

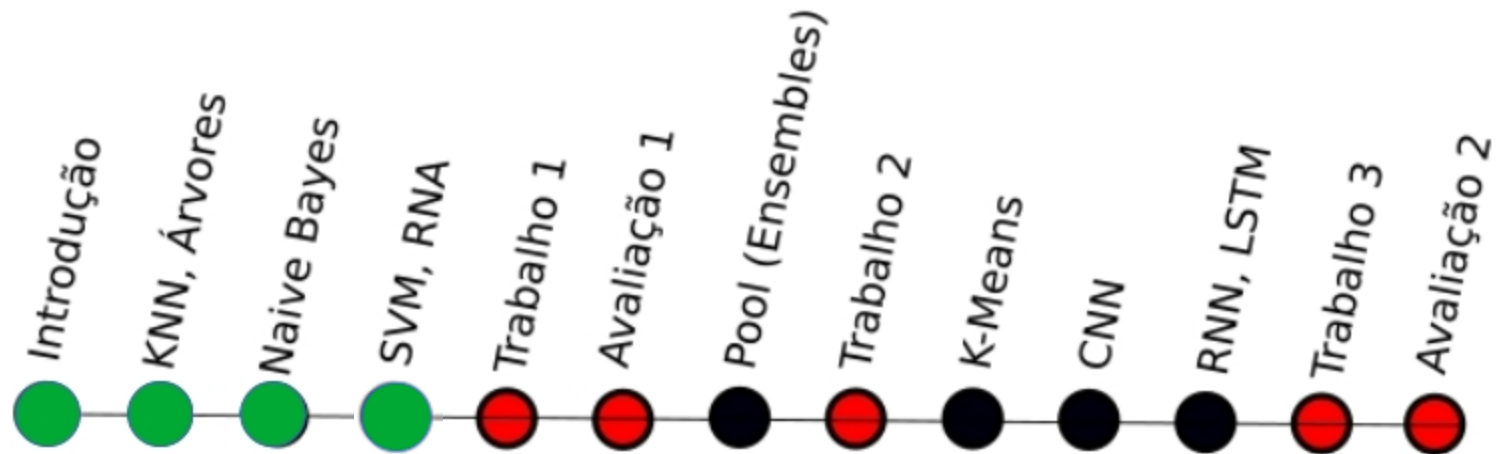
[gustavo.hochuli@pucpr.br](mailto:gustavo.hochuli@pucpr.br)

[aghochuli@ppgia.pucpr.br](mailto:aghochuli@ppgia.pucpr.br)

[github.com/andrehochuli/teaching](https://github.com/andrehochuli/teaching)

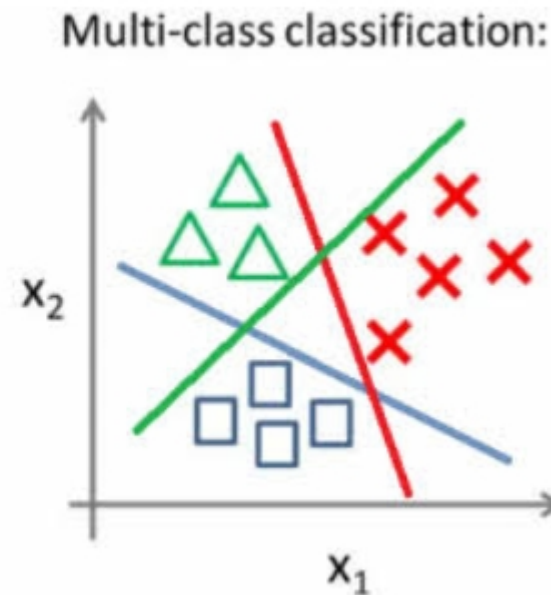
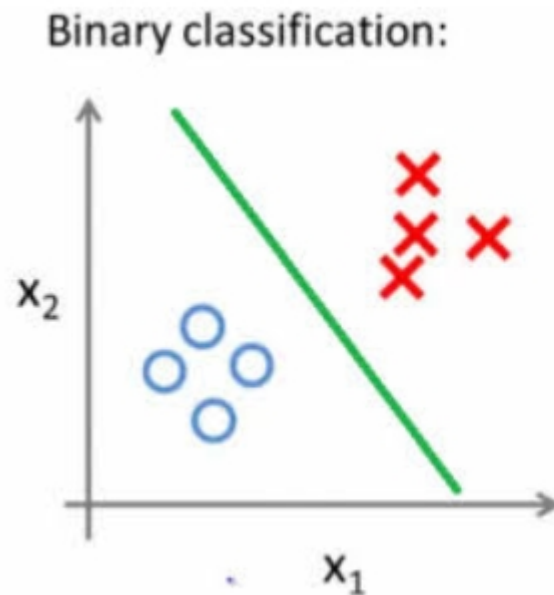
# Plano de Aula

- Discussões Iniciais
- Perceptron
  - Conceção
  - Pesos
  - Bias
- RNA
- Exercícios



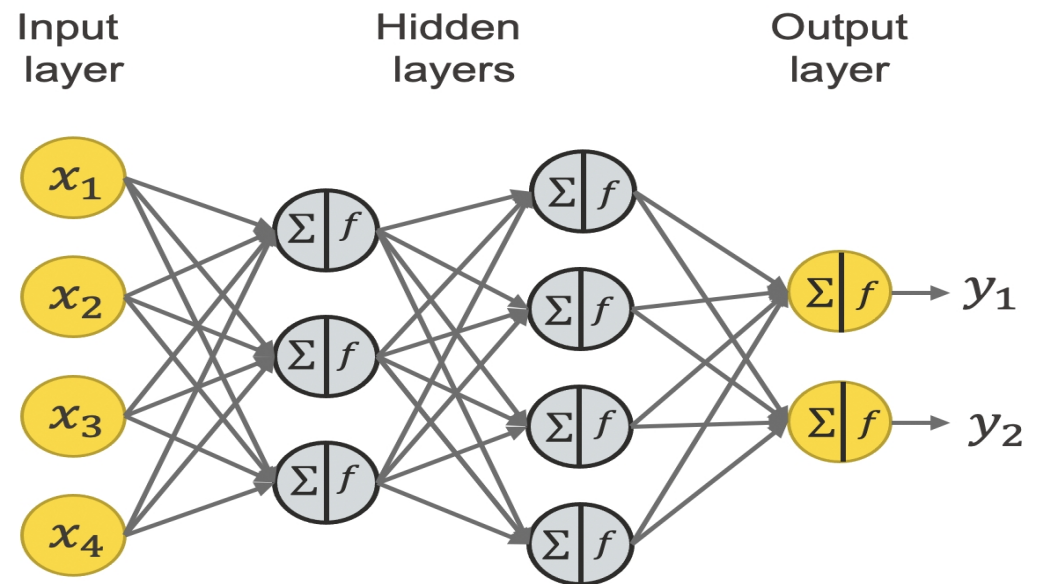
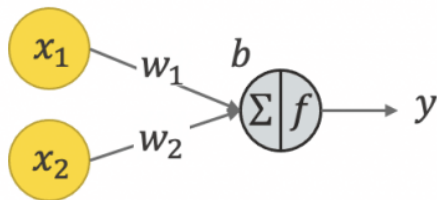
# Discussões Iniciais

- Classificação Binária vs Multi-Classes
  - One vs All
  - One vs One



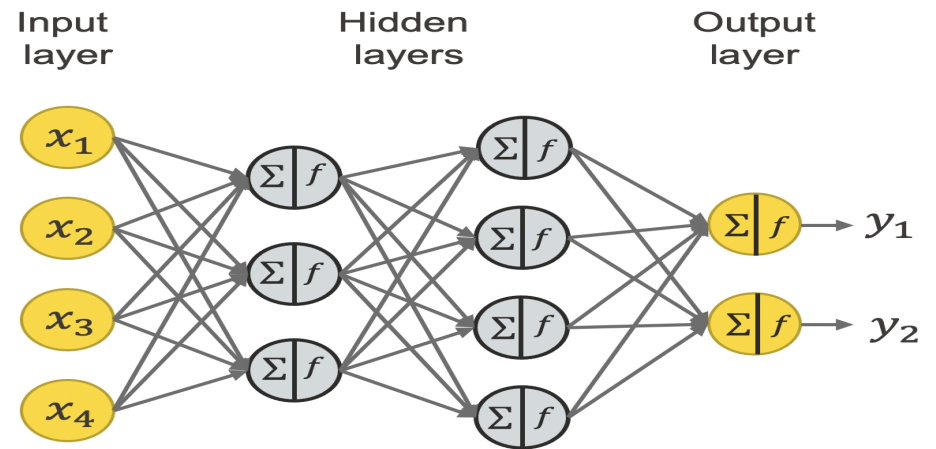
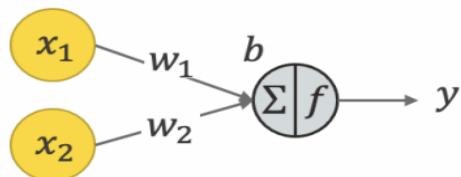
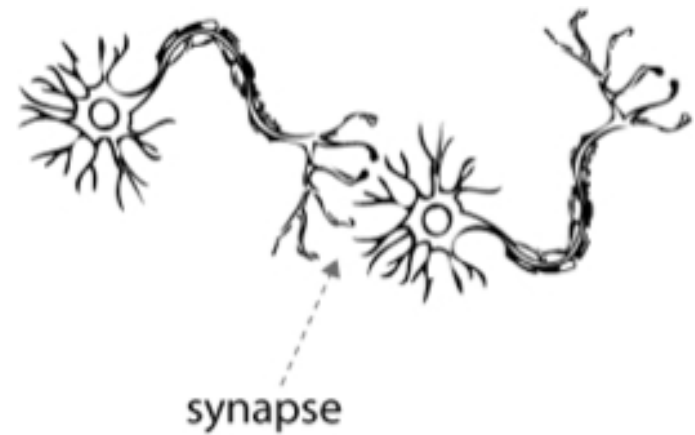
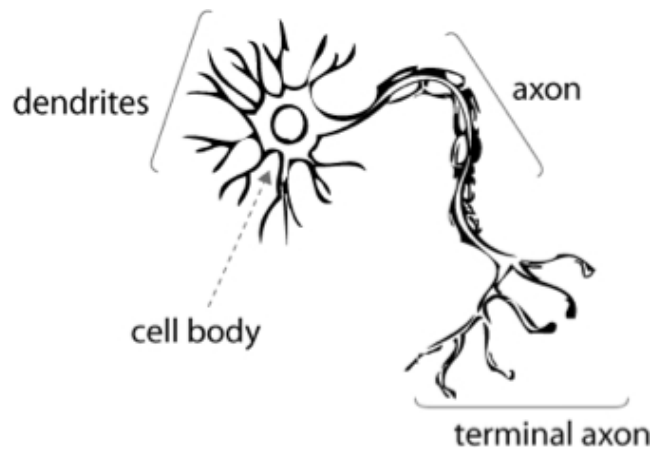
# Redes Neurais Artificiais

- Resolve problemas complexos
- Arquitetura é composta de neurônios artificiais conectados
- Número de Camadas é ilimitado



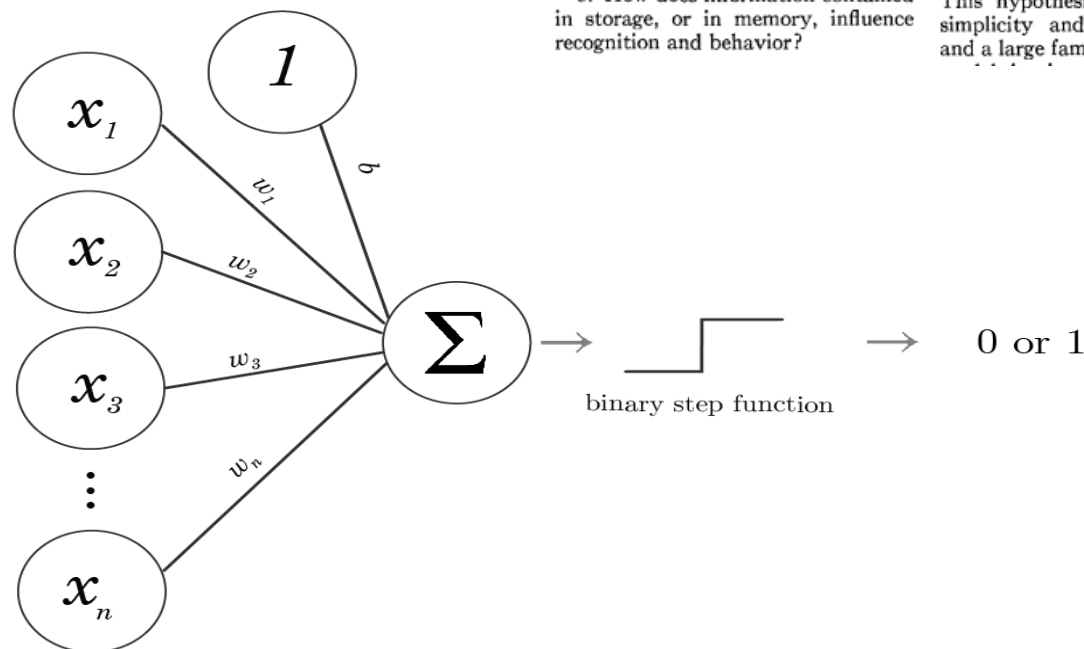
# Redes Neurais Artificiais

- Modelo Biológico vs Modelo Computacional



# Perceptron

- Neurônio Artificial (Rosenblatt) (1958)
- Classificador Binário (0 | 1)
  - Pesos
  - Bias
  - Função de Ativação



*Psychological Review*  
Vol. 65, No. 6, 1958

## THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN<sup>1</sup>

F. ROSENBLATT

*Cornell Aeronautical Laboratory*

If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

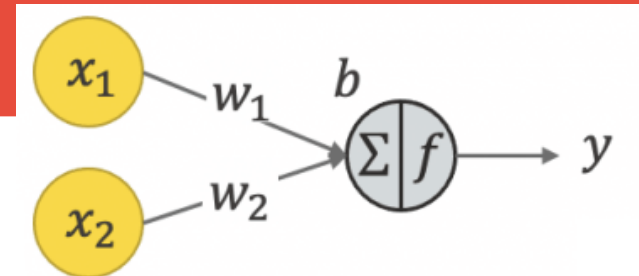
1. How is information about the physical world sensed, or detected, by the biological system?

2. In what form is information stored, or remembered?

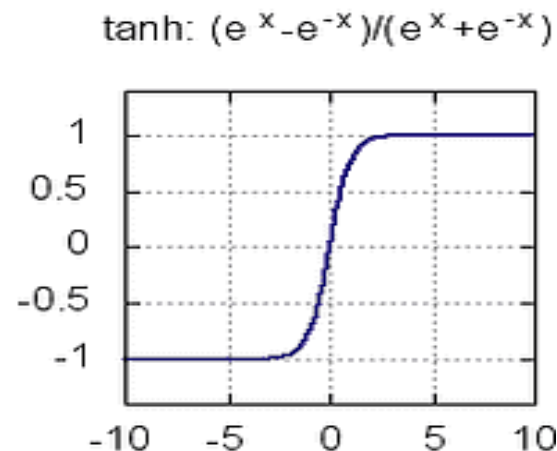
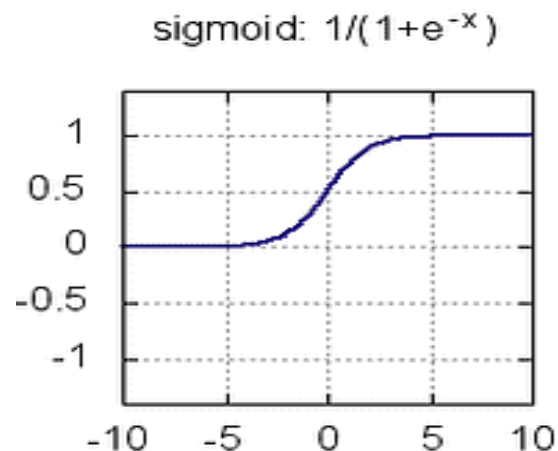
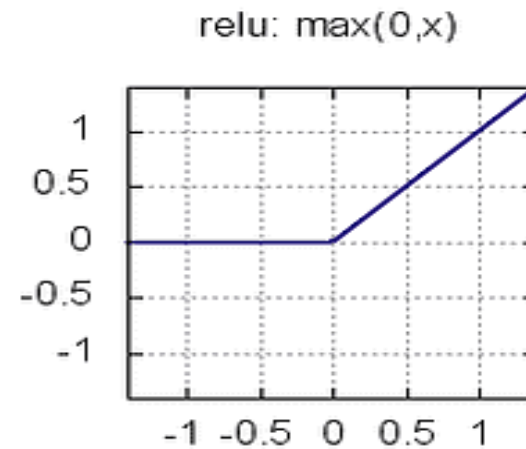
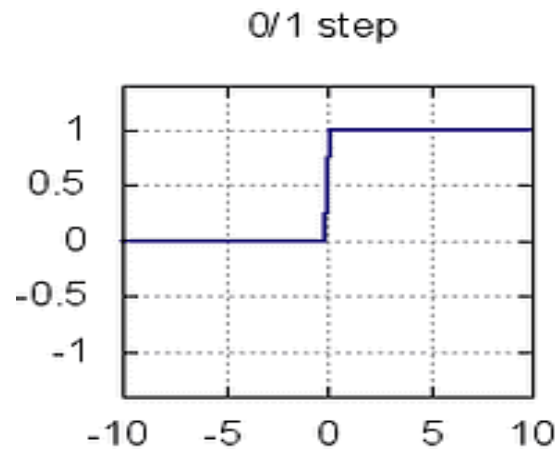
3. How does information contained in storage, or in memory, influence recognition and behavior?

and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the "memory traces" which they have left, much as we might develop a photographic negative, or translate the pattern of electrical charges in the "memory" of a digital computer. This hypothesis is appealing in its simplicity and ready intelligibility, and a large family of theoretical brain

# Perceptron



- Função de Ativação



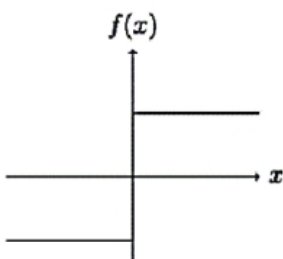
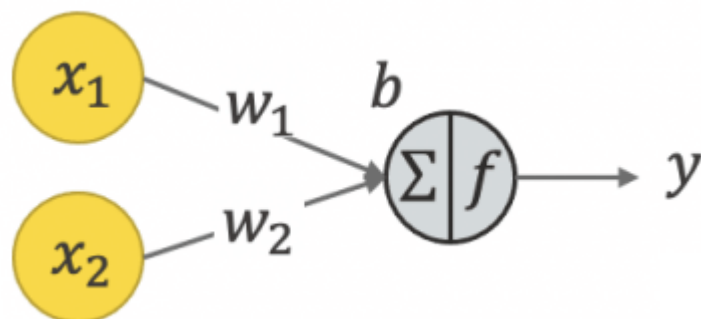
# Perceptron

- Vamos considerar uma entrada com 2 atributos (features)

- $x_1 * w_1 + x_2 * w_2 + b$

- Considere:

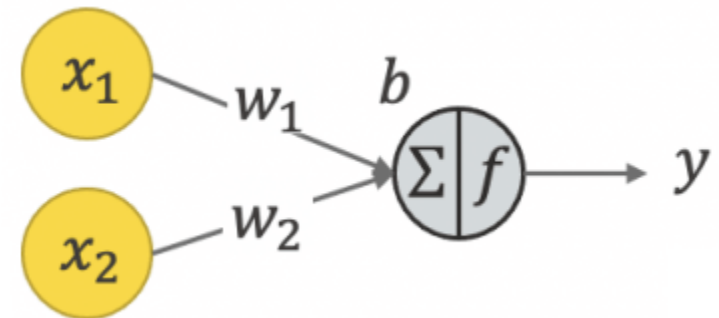
- $w_1 = 1$
  - $w_2 = 0$
  - $w_1 = 0.2$
  - $w_2 = 0.8$
  - $b = 0$
  - $Y = 1$
  - Ativação: Step Function





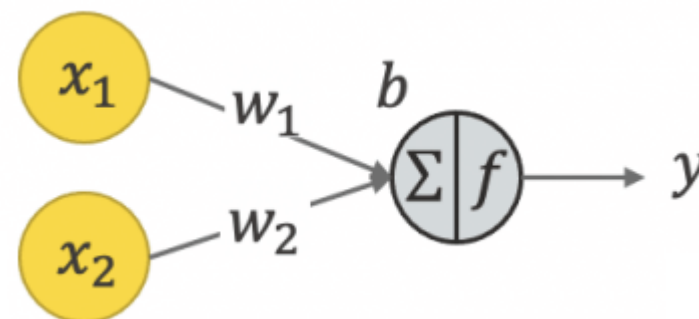
# Perceptron

- Treinamento
  - Atualizar  $w_1$ ,  $w_2$  e  $b$
  - Até que seja atingido o 'y' desejado
- Como?
  - Loss (Função de Perda)
  - Learning Rate (Atualização dos pesos)



# Perceptron

- Rotina de Treino
  - Inicializar pesos e bias aleatórios
  - Definir a função de ativação (exemplo: STEP Function)
  - Para cada exemplo no conjunto de treinamento:
    - a. Calcular a saída do Perceptron
    - b. Atualizar os pesos e bias com base no erro
- Rotina de Teste
  - Computar com os pesos do treino



# Perceptron - Implementação Python

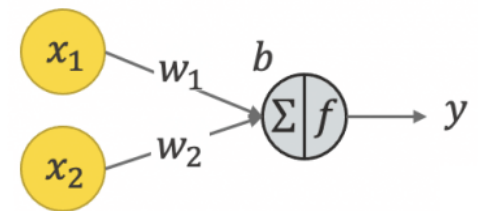
- Definição das funções básicas

```
import numpy as np

def activation(x):
    return

def predict(X, weights, bias):
    return

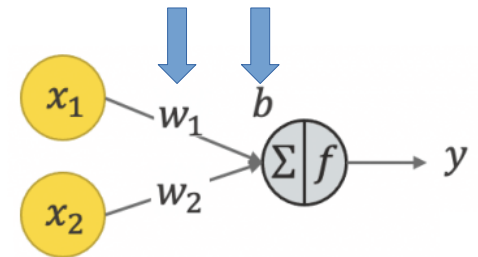
def fit(X, y, learning_rate=0.001, epochs=100):
    return
```



# Perceptron - Implementação Python

- Fit()
- Inicialização dos pesos

```
def fit(X, y, learning_rate=0.001, epochs=100):  
    n_features = X.shape[1]  
  
    # Inicialização  
    weights = weights = np.random.rand(n_features)  
    bias = 0
```

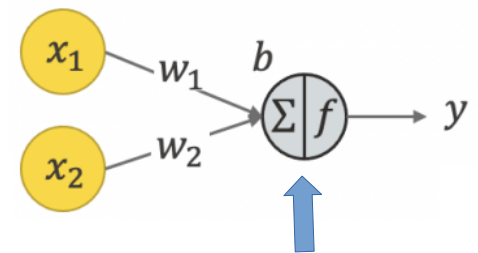


# Perceptron - Implementação Python

- Fit()
  - Iteração sobre a base
    - Computa o produto escalar dos pesos e bias
    - Função de Ativação
    - Perda

```
def activation(x):  
    return np.where(x >= 0, 1, 0)
```

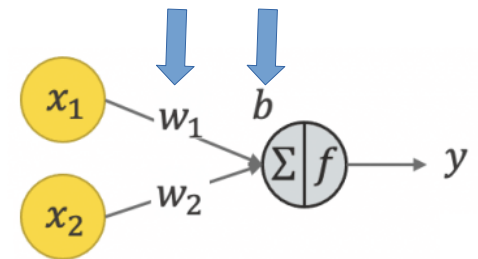
```
def fit(X, y, learning_rate=0.001, epochs=100):  
    n_features = X.shape[1]  
  
    # Inicialização  
    weights = np.random.rand(n_features)  
    bias = 0  
  
    # Iterating until the number of epochs  
    for epoch in range(epochs):  
        # iteração entre as amostras  
        for i in range(len(X)):  
            z = np.dot(X, weights) + bias # Produto escalar e bias  
            y_pred = activation(z) # Função de ativação  
            loss = (y[i] - y_pred[i]) # cálculo da perda
```



# Perceptron - Implementação Python

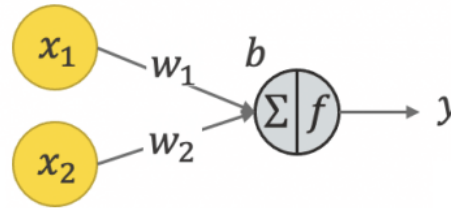
- Fit()
- Atualização dos pesos e bias com base no erro

```
def fit(X, y, learning_rate=0.001, epochs=100):  
    n_features = X.shape[1]  
  
    # Inicialização  
    weights = np.random.rand(n_features)  
    bias = 0  
  
    # Iterating until the number of epochs  
    for epoch in range(epochs):  
        # iteração entre as amostras  
        for i in range(len(X)):  
            z = np.dot(X[i], weights) + bias # Produto escalar e bias  
            y_pred = activation(z) #Função de ativação  
            loss = (y[i] - y_pred[i]) #calculo da perda  
  
            #Atualização dos pesos com base no erro  
            weights = weights + learning_rate * loss * X[i]  
            bias = bias + learning_rate * loss  
  
    return weights, bias
```



# Perceptron - Implementação Python

- Fit()
- Criar o dataset e treinar



```
def fit(X, y, learning_rate=0.001, epochs=100):
    n_features = X.shape[1]

    # Inicialização
    weights = np.random.rand(n_features)
    bias = 0

    # Iterating until the number of epochs
    for epoch in range(epochs):
        # iteração entre as amostras
        for i in range(len(X)):
            z = np.dot(X, weights) + bias # Produto escalar e bias
            y_pred = activation(z) #Função de ativação
            loss = (y[i] - y_pred[i]) #calculo da perda

            #Atualização dos pesos com base no erro
            weights = weights + learning_rate * loss * X[i]
            bias = bias + learning_rate * loss

    return weights, bias
```

```
#Features e Labels
X = np.array([[0, 0, 0],
              [0, 1, 0],
              [1, 1, 0],
              [1, 1, 1]])

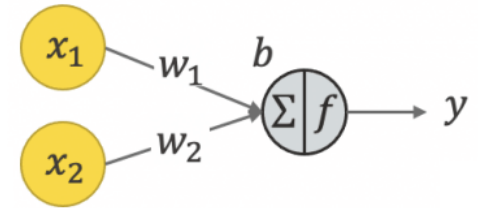
y = np.array([1,
              0,
              0,
              1])

w,b = fit(X, y,
          learning_rate=0.001,
          epochs=100)
```

# Perceptron - Implementação Python

- Predict(): Utilizar os pesos do fit e testar

```
def predict(X, weights, bias):  
    z = np.dot(X, weights) + bias  
    return activation(z)  
  
#Features e Labels  
X = np.array([[0, 0, 0],  
              [0, 1, 0],  
              [1, 1, 0],  
              [1, 1, 1]])  
  
y = np.array([1,  
              0,  
              0,  
              1])  
  
w, b = fit(X, y,  
           learning_rate=0.001,  
           epochs=100)  
  
pred = predict(X, w, b)  
print(pred)
```



[Link para Código Fonte](#)