

Instituto Tecnológico de Aeronáutica

**PO-249**

**INTRODUÇÃO ÀS REDES NEURAIS E AOS GRADENS MODELOS DE  
LINGUAGEM  
ATIVIDADE 02**

Aluno: Luiz Eduardo T. C. Cartolano  
Prof: Profº Dr. Mauri de Oliveira

São José dos Campos, SP - Brasil  
2025

# 1 Introdução

O Perceptron é um dos modelos mais simples de redes neurais artificiais, apresentado por Frank Rosenblatt em 1958 [1]. Ele é utilizado principalmente para problemas de classificação binária linearmente separáveis, isto é, problemas em que os dados podem ser divididos por uma linha (em 2D), plano (em 3D) ou hiperplano (em dimensões superiores).

O funcionamento do Perceptron consiste em combinar as entradas com pesos, aplicar uma função de ativação e, com base nisso, tomar uma decisão sobre a classe prevista.

## 2 Método de Aprendizado do Perceptron

Para realizar o treinamento de um Perceptron são necessárias algumas informações que modelem o problema. Primeiro, precisamos de um vetor de entrada, que pode ser descrito como:

$$\mathbf{x} = (x_1, x_2, \dots, x_n) \quad (1)$$

Cada entrada está associada a um peso  $w_i$ , que define a importância daquela entrada, logo, teremos também um conjunto de pesos dado por:

$$\mathbf{w} = (w_1, w_2, \dots, w_n) \quad (2)$$

Parte essencial do processo é o cálculo do potencial, que associa pesos e entradas, e pode ser descrito como:

$$v = \sum_{i=1}^n w_i x_i \quad (3)$$

Também é preciso uma função de ativação, que transforma o potencial em um valor discreto, por exemplo para a regra do **OU** que iremos analisar nesse relatório, temos:

$$y = f(v) = \begin{cases} 1, & \text{se } v \geq 0 \\ 0, & \text{se } v < 0 \end{cases} \quad (4)$$

Por fim, os pesos são ajustados conforme o erro entre a saída esperada  $d$  e a saída obtida  $y$ :

$$w_{k+1} = w_k + \eta(d - y)\mathbf{x} \quad (5)$$

onde  $\eta$  é a taxa de aprendizado.

## 3 Aprendizado do Perceptron para o operador OU

Considere o problema lógico do **OR** (OU) com duas entradas. Para incluir o termo de bias, adicionamos  $x_0 = 1$  em todas as amostras:

$x_0$	$x_1$	$x_2$	Saída Esperada ( $d$ )
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0

Inicializamos os pesos como:

$$\mathbf{w} = [w_0, w_1, w_2] = [0, 0, 0] \quad (6)$$

e utilizamos taxa de aprendizado:

$$\eta = 0.5 \quad (7)$$

## Primeira Época

Para a primeira amostra  $(x_0, x_1, x_2) = (1, 1, 1)$  com saída esperada  $d = 1$ :

$$\begin{aligned} v &= w \cdot x = (0, 0, 0) \cdot (1, 1, 1) = 0 \\ y &= f(v) = 0 \\ e &= d - y = 1 - 0 = 1 \\ w &= w + \eta \cdot e \cdot x \\ &= (0, 0, 0) + 0.5 \cdot 1 \cdot (1, 1, 1) \\ &= (0.5, 0.5, 0.5) \end{aligned}$$

Para a segunda amostra  $(x_0, x_1, x_2) = (1, 1, 0)$  com saída esperada  $d = 1$ :

$$\begin{aligned} v &= w \cdot x = (0.5, 0.5, 0.5) \cdot (1, 1, 0) = 1 \\ y &= f(v) = 1 \\ e &= d - y = 1 - 1 = 0 \end{aligned}$$

Como o erro é zero, não há atualização de pesos.

Para a terceira amostra  $(x_0, x_1, x_2) = (1, 0, 1)$  com saída esperada  $d = 1$ :

$$\begin{aligned} v &= w \cdot x = (0.5, 0.5, 0.5) \cdot (1, 0, 1) = 1 \\ y &= f(v) = 1 \\ e &= d - y = 1 - 1 = 0 \end{aligned}$$

Como o erro é zero, não há atualização de pesos.

Para a quarta amostra  $(x_0, x_1, x_2) = (1, 0, 0)$  com saída esperada  $d = 0$ :

$$\begin{aligned} v &= w \cdot x = (0.5, 0.5, 0.5) \cdot (1, 0, 0) = 0.5 \\ y &= f(v) = 1 \\ e &= d - y = 0 - 1 = -1 \\ w &= w + \eta \cdot e \cdot x \\ &= (0.5, 0.5, 0.5) + 0.5 \cdot -1 \cdot (1, 0, 0) \\ &= (0, 0.5, 0.5) \end{aligned}$$

## Segunda Época

Para a primeira amostra  $(x_0, x_1, x_2) = (1, 1, 1)$  com saída esperada  $d = 1$ :

$$\begin{aligned} v &= w \cdot x = (0, 0.5, 0.5) \cdot (1, 1, 1) = 1 \\ y &= f(v) = 1 \\ e &= d - y = 1 - 1 = 0 \end{aligned}$$

Como o erro é zero, não há atualização de pesos.

Para a segunda amostra  $(x_0, x_1, x_2) = (1, 1, 0)$  com saída esperada  $d = 1$ :

$$\begin{aligned} v &= w \cdot x = (0, 0.5, 0.5) \cdot (1, 1, 0) = 1 \\ y &= f(v) = 1 \\ e &= d - y = 1 - 1 = 0 \end{aligned}$$

Como o erro é zero, não há atualização de pesos.

Para a terceira amostra  $(x_0, x_1, x_2) = (1, 0, 1)$  com saída esperada  $d = 1$ :

$$\begin{aligned} v &= w \cdot x = (0, 0.5, 0.5) \cdot (1, 0, 1) = 1 \\ y &= f(v) = 1 \\ e &= d - y = 1 - 1 = 0 \end{aligned}$$

Como o erro é zero, não há atualização de pesos.

Para a quarta amostra  $(x_0, x_1, x_2) = (1, 0, 0)$  com saída esperada  $d = 0$ :

$$\begin{aligned}v &= w \cdot x = (0, 0.5, 0.5) \cdot (1, 0, 0) = 0 \\y &= f(v) = 0 \\e &= d - y = 0 - 0 = 0\end{aligned}$$

Como o erro é zero, não há atualização de pesos.

## Resultado

Como não houve atualização dos pesos durante toda a segunda época do treinamento, podemos concluir que o treinamento foi concluído, e o vetor de pesos final é dado por:

$$\mathbf{w} = [0, 0.5, 0.5] \quad (8)$$

## 4 Solução em Python para o Aprendizado do Perceptron

A seguir, uma implementação em Python do treinamento de um Perceptron utilizando a regra de aprendizado:

```
1 import time
2
3 import numpy as np
4
5 from models.training_result import TrainingResult
6
7
8 class PerceptronRule:
9     """
10
11         Implements the Perceptron learning rule.
12         This rule updates the weights based on the error between the
13             predicted and actual output.
14         It iterates through the input data, calculates the predicted
15             output, computes the error,
16         and updates the weights accordingly.
17
18     Attributes:
19
```

```

16     lr (float): Learning rate for weight updates.
17     max_iters (int): Maximum number of iterations to perform.
18     verbose (bool): If True, prints detailed logs of the
19         training process.
20
21     Methods:
22
23         fit(input_data: np.ndarray, output_data: np.ndarray,
24             weights: np.ndarray, y_func: callable) -> np.ndarray:
25             Trains the perceptron using the provided input and
26             output data.
27             Returns the updated weights after training.
28
29             """
30
31     def __init__(self, lr: float, max_iters: int, verbose: bool =
32         False):
33         self.lr = lr
34         self.max_iters = max_iters
35         self.verbose = verbose
36
37     def __log(self, message: str = "") -> None:
38
39             """
40             Logs a message if verbose mode is enabled.
41             :param message: a string message to log
42             :return: a None
43
44             """
45
46         if self.verbose:
47             print(message)
48
49
50     def fit(self, input_data: np.ndarray, output_data: np.ndarray
51         , weights: np.ndarray, y_func: callable) -> TrainingResult
52         :
53
54             """
55             Trains the perceptron using the provided input and output
56             data.
57             :param input_data: a 2D NumPy array where each row is an
58                 input vector.
59             :param output_data: a 1D NumPy array containing the
60                 expected outputs corresponding to the input vectors.
61             :param weights: a 1D NumPy array representing the initial
62                 weights of the perceptron.

```



```

76             self._log("\t\tNo weight update (prediction
77                         correct)")
78             iter_j += 1
79             iter_i += 1
80
81             self._log("==" * 90)
82
83     return TrainingResult(
84         weights=weights,
85         epochs=iter_i,
86         elapsed_time=time.time() - start_time
87     )

```

Código 1: Aprendizado do Perceptron

E também o script com a declaração da entrada, pesos iniciais e função de ativação:

```

1 import numpy as np
2
3 from trainingRules.perceptron_rule import PerceptronRule
4
5
6 def f_net(f_in):
7     """
8         Activation function for the perceptron.
9         :param f_in: a float representing the net input to the
10            perceptron.
11
12         :return: a binary output (1 or 0) based on the net input.
13
14
15 if __name__ == "__main__":
16     # Input and Desired Output
17     x = np.array([[1, 1, 1], [1, 1, 0], [1, 0, 1], [1, 0, 0]])
18     d = np.array([1, 1, 1, 0])
19
20     # Initial Weights
21     w = np.array([0, 0, 0])
22
23     # Learning Rate
24     eta = 0.5
25
26     # Maximum Number of Iterations

```

```

24     num_iterations = 100
25     # Print Output
26     print("=" * 90)
27     print("Perceptron Rule:")
28     perceptron_rule = PerceptronRule(eta, num_iterations, verbose
29         =True)
30     perceptron_results = perceptron_rule.fit(x, d, w.copy(),
31         f_net)
32     print(f"Final weights with Perceptron Rule: {
33         perceptron_results.weights}")
34     print("Number of epochs:", perceptron_results.epochs)
35     print("Time taken for training: {:.6f} seconds".format(
36         perceptron_results.elapsed_time))
37     print("=" * 90)

```

Código 2: Script para aprendizado do Perceptron

E a saída do código é mostrada na Figura 1.

## Referências

- [1] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

## Anexos

```
=====
Perceptron Rule:
=====
Iteration 1
=====
[Sample 1] Input: [1 1 1], Expected: 1
    Net input (v): 0.0000 = [1 1 1] * [0 0 0]
    Activation f(v): 0.0000
    Error: 1 - 0.0000 = 1.0000
    Weight update: [0 0 0] + 0.5 * 1.0000 * [1 1 1] = [0.5 0.5 0.5]
[Sample 2] Input: [1 1 0], Expected: 1
    Net input (v): 1.0000 = [1 1 0] * [0.5 0.5 0.5]
    Activation f(v): 1.0000
    Error: 1 - 1.0000 = 0.0000
    No weight update (prediction correct)
[Sample 3] Input: [1 0 1], Expected: 1
    Net input (v): 1.0000 = [1 0 1] * [0.5 0.5 0.5]
    Activation f(v): 1.0000
    Error: 1 - 1.0000 = 0.0000
    No weight update (prediction correct)
[Sample 4] Input: [1 0 0], Expected: 0
    Net input (v): 0.5000 = [1 0 0] * [0.5 0.5 0.5]
    Activation f(v): 1.0000
    Error: 0 - 1.0000 = -1.0000
    Weight update: [0.5 0.5 0.5] + 0.5 * -1.0000 * [1 0 0] = [0.  0.5 0.5]
=====
Iteration 2
=====
[Sample 1] Input: [1 1 1], Expected: 1
    Net input (v): 1.0000 = [1 1 1] * [0.  0.5 0.5]
    Activation f(v): 1.0000
    Error: 1 - 1.0000 = 0.0000
    No weight update (prediction correct)
[Sample 2] Input: [1 1 0], Expected: 1
    Net input (v): 0.5000 = [1 1 0] * [0.  0.5 0.5]
    Activation f(v): 1.0000
    Error: 1 - 1.0000 = 0.0000
    No weight update (prediction correct)
[Sample 3] Input: [1 0 1], Expected: 1
    Net input (v): 0.5000 = [1 0 1] * [0.  0.5 0.5]
    Activation f(v): 1.0000
    Error: 1 - 1.0000 = 0.0000
    No weight update (prediction correct)
[Sample 4] Input: [1 0 0], Expected: 0
    Net input (v): 0.0000 = [1 0 0] * [0.  0.5 0.5]
    Activation f(v): 0.0000
    Error: 0 - 0.0000 = 0.0000
    No weight update (prediction correct)
=====
Final weights with Perceptron Rule: [0.  0.5 0.5]
Number of epochs: 2
Time taken for training: 0.001709 seconds
=====
```

Figura 1: Output do código em Python para treinamento do Perceptron.

Nome: Daniz Eduardo Tonga Chaves Contelone

$$\text{entradas} \Rightarrow \mathbf{x} = [x_0, x_1, x_2]$$

$$w_0(k) = w_0(k-1) + \eta(d(k-1) - y(k-1))x_0(k-1)$$

$$\text{pesos} \Rightarrow \mathbf{w} = [w_0, w_1, w_2]$$

$$w_1(k) = w_1(k-1) + \eta(d(k-1) - y(k-1))x_1(k-1)$$

$$\text{potencial} \Rightarrow \mathcal{J} = \sum_{i=1}^m w_i x_i + w_0$$

$$w_2(k) = w_2(k-1) + \eta(d(k-1) - y(k-1))x_2(k-1)$$

$$= w_0 x_0 + w_1 x_1 + w_2 x_2 + w_0$$

$$1^{\text{a}} \text{ entrada: } [1, 1, 1] \quad d = 1$$

$$= w_0 x_0 + w_1 x_1 + w_2 x_2 + w_0$$

$$2^{\text{a}} \text{ entrada: } [1, 1, 0] \quad d = 1$$

$$= w_0 x_0 + w_1 x_1 + w_2 x_2 + w_0$$

$$3^{\text{a}} \text{ entrada: } [1, 0, 1] \quad d = 1$$

$$= w_0 x_0 + w_1 x_1 + w_2 x_2 + w_0$$

$$4^{\text{a}} \text{ entrada: } [1, 0, 0] \quad d = 0$$

### PRIMEIRA ÉPOCA

$$w_0 = 0 \quad \triangleleft \text{entrada}$$

$$\Rightarrow \mathcal{J} = x_0 w_0 + x_1 w_1 + x_2 w_2$$

$$w_1 = 0 \quad \triangleleft [1, 1, 1]$$

$$= 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 0$$

$$w_2 = 0$$

$$= 0$$

$$\eta = 0,5$$

$$\varphi(0) = 0$$

$$e = d - \varphi(0)$$

$$1 - 0$$

$$= 1$$

$$\Rightarrow w_0 = 0 + 0,5 \cdot 1 \cdot 1 \\ = 0,5$$

$$w_1 = 0 + 0,5 \cdot 1 \cdot 1 \\ = 0,5$$

$$w_2 = 0 + 0,5 \cdot 1 \cdot 1 \\ = 0,5$$

$$\Rightarrow \text{Pesos após a } 1^{\text{a}} \text{ entrada: } [0,5, 0,5, 0,5]$$

LISTA - II

Figura 2: Aprendizado do Perceptron para a regra do OU.

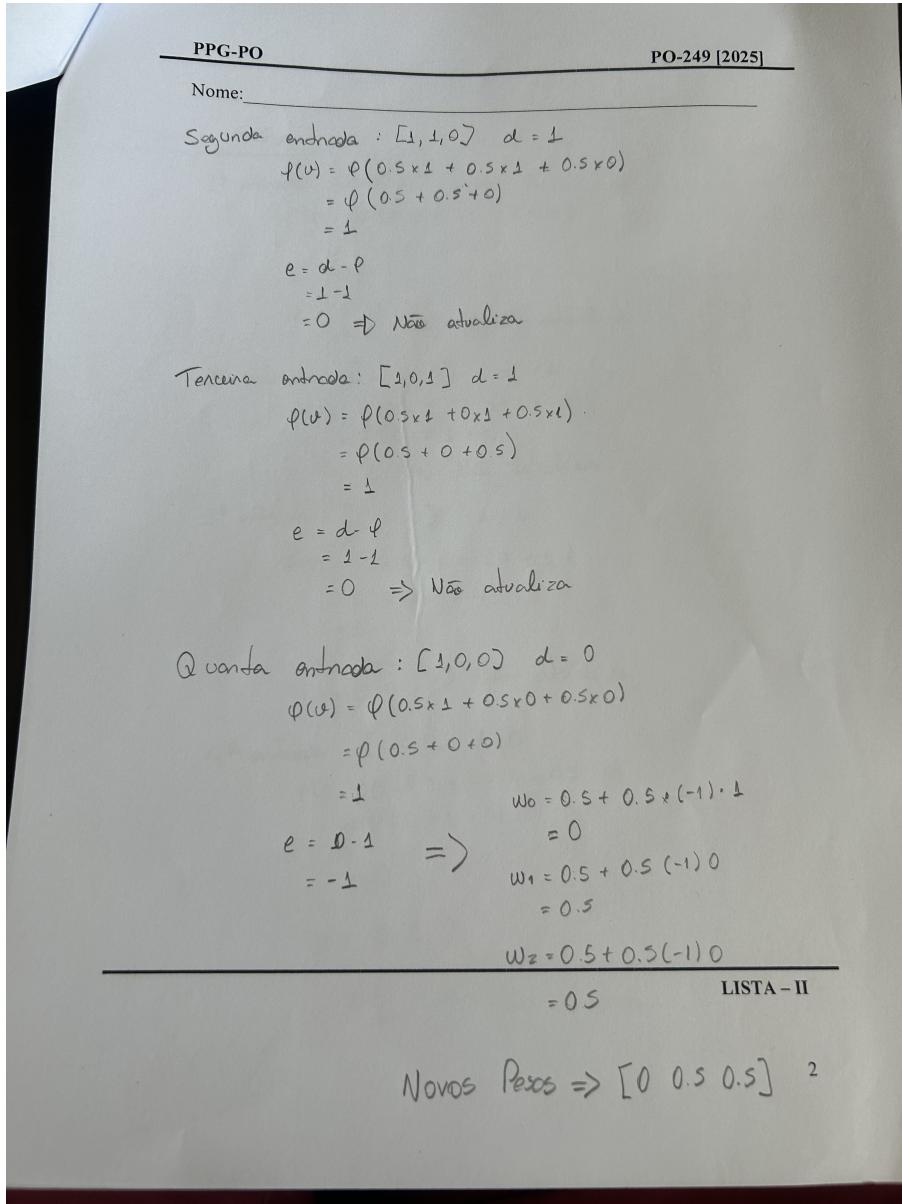


Figura 3: Aprendizado do Perceptron para a regra do OU.

Nome: \_\_\_\_\_

SEGUNDA ÉPOCA1<sup>a</sup> entrada:  $[1 \ 1 \ 1]$   $d = 1$ 

$$\begin{aligned}\varphi(w) &= \varphi(0 \cdot 1 + 0.5 \cdot 1 + 0.5 \cdot 1) \\ &= \varphi(0 + 0.5 + 0.5) \\ &= 1\end{aligned}$$

$$e = \frac{1}{1} - 1 = 0 \Rightarrow \text{Não atualiza}$$

2<sup>a</sup> entrada:  $[1 \ 1 \ 0]$   $d = 1$ 

$$\begin{aligned}\varphi(w) &= \varphi(0 \cdot 1 + 0.5 \cdot 1 + 0 \cdot 0) \\ &= \varphi(0 + 0.5 + 0) \\ &= 1 \Rightarrow \text{Não atualiza}\end{aligned}$$

3<sup>a</sup> entrada:  $[1 \ 0 \ 1]$   $d = 1$ 

$$\begin{aligned}\varphi(w) &= \varphi(0 \cdot 1 + 0.5 \cdot 0 + 0.5 \cdot 1) \\ &= \varphi(0 + 0 + 0.5) \\ &= 1\end{aligned}$$

$$e = \frac{1}{1} - 1 = 0 \Rightarrow \text{Não atualiza}$$

4<sup>a</sup> entrada:  $[1, 0, 0]$   $d = 0$ 

$$\begin{aligned}\varphi(w) &= \varphi(0 \cdot 1 + 0.5 \cdot 0 + 0.5 \cdot 0) \\ &= \varphi(0 + 0 + 0) \\ &= 0\end{aligned}$$

$$e = d \cdot 0$$

$$= 0 \Rightarrow \text{Não atualiza}$$

LISTA - II

Figura 4: Aprendizado do Perceptron para a regra do OU.

Nome: \_\_\_\_\_

Conclusão

=> Como os pesos não  
foram atualizados durante  
a época (p/ qualquer entrada)  
podemos concluir que os pesos  
finais são:  
 $[0 \ 0.5 \ 0.5]$

---

LISTA - II

2

Figura 5: Aprendizado do Perceptron para a regra do OU.