

## Gradiente Descendente

Alunos: Anabelle Elizabeth Araujo de Souza, Gabriel Luiz dos Santos Silva e Jules Severo Barcos

Prof. Me. Alexandre Garcia de Oliveira

Santos, 12 de Maio de 2023

CONTROLE DE VERSÃO			
Autor	Versão	Data	Descrição
Anabelle Souza	1.0	14/05/2023	Edição do documento
Jules Severo	2.0	14/05/2023	Edição do documento
Gabriel L. S. Silva	3.0	14/05/2023	Edição do documento

## Sumário

<b>1</b>	<b>Definição de Gradiente Descendente</b>	<b>4</b>
<b>2</b>	<b>Parábola com Gradiente Descendente</b>	<b>4</b>
<b>3</b>	<b>Github</b>	<b>11</b>

## 1 Definição de Gradiente Descendente

Gradiente descendente é um algoritmo de otimização usado em aprendizado de máquina e em outros campos relacionados, como a otimização de funções matemáticas. O objetivo do gradiente descendente é encontrar o mínimo global de uma função de custo, ajustando os valores dos parâmetros de um modelo de acordo com a direção do gradiente descendente, ou seja, a direção em que a função está decrescendo mais rapidamente. O algoritmo funciona iterativamente, atualizando os valores dos parâmetros em pequenos passos, de acordo com a inclinação da curva de custo em relação a cada parâmetro. Esses passos são determinados pelo tamanho do passo (também conhecido como taxa de aprendizado) e pela magnitude do gradiente em cada ponto. O processo continua até que a função de custo atinja um mínimo global ou até que um critério de parada seja atingido. O gradiente descendente é um dos algoritmos mais usados em aprendizado de máquina devido à sua simplicidade e eficácia em muitos tipos de problemas de otimização.

## 2 Parábola com Gradiente Descendente

Esse algoritmo utiliza o conjunto de treinamento abaixo para identificar a melhor parábola para minimização de erro:

$$[(1, 4), (-1, 5), (2, 7), (-2, 8), (3, 12), (-3, 13), (4, 19), (-4, 20)]$$

O método do gradiente descendente é um algoritmo de otimização que usa o gradiente da função (derivada parcial em cada dimensão) para encontrar o mínimo local. O algoritmo atualiza iterativamente as variáveis de entrada usando a fórmula:

Para continuar o seu texto em LaTeX, aqui está a fórmula do método do gradiente descendente:

$$x_{t+1} = x_t - \gamma \nabla f(x_t)$$

Nesta fórmula,  $x_t$  representa o vetor de variáveis de entrada na iteração  $t$ ,  $\gamma$  é a taxa de aprendizado (learning rate),  $\nabla f(x_t)$  é o gradiente da função objetivo  $f$  em relação a  $x_t$ , e  $x_{t+1}$  é o vetor de variáveis atualizado na próxima iteração.

O gradiente  $\nabla f(x_t)$  é um vetor composto pelas derivadas parciais da função objetivo em relação a cada dimensão de  $x_t$ . Essas derivadas parciais fornecem a direção e a magnitude do maior crescimento da função em cada dimensão. Assim, ao subtrair  $\gamma \nabla f(x_t)$  de  $x_t$ , o algoritmo realiza um passo em direção ao mínimo local, diminuindo gradualmente o valor da função.

O processo é repetido até chegar a uma condição de desejada, como um número máximo de iterações ou quando o erro for suficientemente próx. de zero..

**Resumindo:** Esse algoritmo é amplamente utilizado em diversas áreas, como aprendizado de máquina e otimização numérica, devido à sua eficácia na busca de mínimos locais em problemas de otimização.

Onde  $x$  representa um ponto na função,  $\nabla f(x)$  é o gradiente da função em  $x$ ,  $\gamma$  é uma constante que determina o tamanho do passo, e  $x'$  é o novo ponto na função.

O algoritmo continua atualizando as variáveis de entrada até que a diferença entre o valor da função atual e o valor anterior esteja abaixo de um valor de tolerância pré-

determinado.

Na imagem abaixo temos a derivação da função feita a mão, com intuito de aplicá-la no conjunto de treinamento determinado acima.

C.T

1	4
-1	5
2	3
-2	8
3	12
-3	13
4	19
-4	80

$$\hat{y} = w_1 x^2 + w_2 x - b$$

$$E(w_1, w_2, b) = \sum_{i=1}^8 (y_i - \hat{y})^2$$

$$E(w_1, w_2, b) = \sum_{i=1}^8 (y_i - w_1 x_i^2 - w_2 x_i - b)^2$$

$$\frac{\partial E}{\partial w_1} = 2(y_1 - w_1 x_1^2 - w_2 x_1 - b) \cdot (-2w_1 x_1)$$

$$= 2 \sum_{i=1}^8 (y_i - w_1 x_i^2 - w_2 x_i - b) \cdot (-2w_1 x_i)$$

$$\frac{\partial E}{\partial w_2} = 2(y_1 - w_1 x_1^2 - w_2 x_1 - b) \cdot (-x_1)$$

$$= 2 \sum_{i=1}^8 (y_i - w_1 x_i^2 - w_2 x_i - b) \cdot (-x_i)$$

$$\frac{\partial E}{\partial b} = 2(y_1 - w_1 x_1^2 - w_2 x_1 - b) \cdot (-1)$$

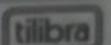
$$= 2 \sum_{i=1}^8 (y_i - w_1 x_i^2 - w_2 x_i - b) \cdot (-1)$$


Figura 1: Derivada a mão 1

```

1 module Main where
2
3 grad :: Double -> Double -> Double -> (Double, Double, Double)
4 grad w1 w2 b =
5   let dedw1 = (2 * (4 - w1 * 1 ** 2 - w2 * 1 - b) * (-2 * w1 * 1) + 2*(5 - w1 * (-1) ** 2 - w2 * (-1) - b) * (-2 * w1 * (-1)) + 2*(7 - w1 * 2 ** 2 - w2 * 2 - b) * (-2 * w1 * 2) + 2*(8 - w1 * (-2) ** 2 - w2 * (-2) - b) * (-2 * w1 * (-2)) + 2*(12 - w1 * 3 ** 2 - w2 * 3 - b) * (-2 * w1 * 3) + 2*(13 - w1 * (-3) ** 2 - w2 * (-3) - b) * (-2 * w1 * (-3)) + 2*(19 - w1 * 4 ** 2 - w2 * 4 - b) * (-2 * w1 * 4) + 2*(20 - w1 * (-4) ** 2 - w2 * (-4) - b) * (-2 * w1 * (-4)))
6     dedw2 = (2 * (4 - w1 * 1 ** 2 - w2 * 1 - b) * (-1) + 2*(5 - w1 * (-1) ** 2 - w2 * (-1) - b) * (-1) + 2*(7 - w1 * 2 ** 2 - w2 * 2 - b) * (-2) + 2*(8 - w1 * (-2) ** 2 - w2 * (-2) - b) * (-(-2)) + 2*(12 - w1 * 3 ** 2 - w2 * 3 - b) * (-3) + 2*(13 - w1 * (-3) ** 2 - w2 * (-3) - b) * (-(-3)) + 2*(19 - w1 * 4 ** 2 - w2 * 4 - b) * (-4) + 2*(20 - w1 * (-4) ** 2 - w2 * (-4) - b) * (-(-4)))
7     dedb = (2 * (4 - w1 * 1 ** 2 - w2 * 1 - b) * (-1) + 2*(5 - w1 * (-1) ** 2 - w2 * (-1) - b) * (-1) + 2*(7 - w1 * 2 ** 2 - w2 * 2 - b) * (-1) + 2*(8 - w1 * (-2) ** 2 - w2 * (-2) - b) * (-1) + 2*(12 - w1 * 3 ** 2 - w2 * 3 - b) * (-1) + 2*(13 - w1 * (-3) ** 2 - w2 * (-3) - b) * (-1) + 2*(19 - w1 * 4 ** 2 - w2 * 4 - b) * (-1) + 2*(20 - w1 * (-4) ** 2 - w2 * (-4) - b) * (-1))
8     in (dedw1, dedw2, dedb)
9
10 tol :: Double
11 tol = 10 ** (-4)
12
13 descent :: Double -> Double -> Double -> Double -> Int -> (Double, Double, Double, Int)
14 descent lr xt yt zt err i
15 | err < tol = (xt, yt, zt, i)
16 | otherwise =
17   let (dedw1, dedw2, dedb) = grad xt yt zt
18   xnovo = xt - lr * dedw1
19   ynovo = yt - lr * dedw2
20   znovo = zt - lr * dedb
21   errnovo = sqrt ((xnovo - xt) ** 2 + (ynovo - yt) ** 2 + (znovo - zt) ** 2)
22   in descent lr xnovo ynovo znovo errnovo (i + 1)
23
24 main = do
25   putStrLn "Digite os valores para executar a função descent:"
26   input <- getLine
27   let [lr, xt, yt, zt, err, i] = map read $ words input
28   (x, y, z, iter) = descent lr xt yt zt err (round i)
29   putStrLn $ "Ponto mínimo encontrado: (" ++ show x ++ ", " ++ show y ++ ", " ++ show z ++ ")"
30   putStrLn $ "Número de iterações: " ++ show iter

```

Ln 7, Col 334 History ↵

Figura 2: Haskell - Código.

```

> descent 0.001 2 0 8 999 0
(1.4278918556644966,-0.1666666666666666,0.29688609340716765,438)

```

Figura 3: Teste1.

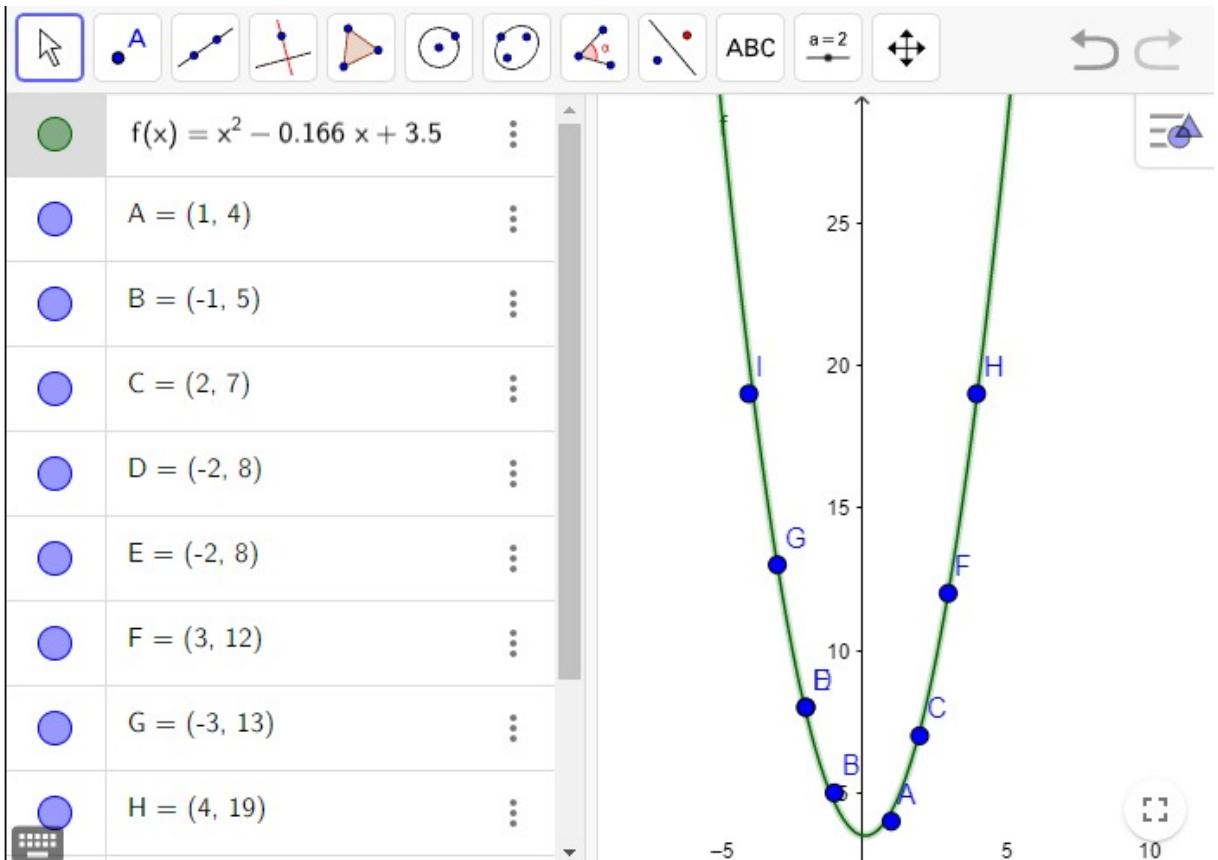


Figura 4: Geogebra.



Figura 5: Geogebra -Último ponto.

Dado o conj. de treinamento  $(1, 4), (1, 2), (2, 7), (2, 8), (3, 11), (3, 13), (4, 19), (4, 20)$   
Qual é a melhor parábola que apoia o c.t de modo a minimizar o erro  
 $\hat{y} = w_1x^2 + w_2x + b$        $E(w_1, w_2, b) = \sum_{i=1}^n (y_i - \hat{y})^2$

$$\frac{\partial E}{\partial w_1}(w_1, w_2, b) = \sum_{i=1}^n (y_i - (w_1x^2 + w_2x + b))^2 \quad \left| \frac{\partial E}{\partial w_1}(w_1, w_2, b) = \sum_{i=1}^n 2(y_i - \hat{y}) \cdot [y_i - (w_1x^2 + w_2x + b)] \right|_{w_1}$$

$$\frac{\partial E}{\partial w_2}(w_1, w_2, b) = \sum_{i=1}^n 2(y_i - \hat{y}) \cdot (-x^2) \rightarrow \frac{\partial E}{\partial w_2}(w_1, w_2, b) = -2 \sum_{i=1}^n (y_i - \hat{y})x^2 //$$

$$\frac{\partial E}{\partial w_2}(w_1, w_2, b) = \sum_{i=1}^n (y_i - (w_1x^2 + w_2x + b))^2 \quad \left| \frac{\partial E}{\partial w_2}(w_1, w_2, b) = \sum_{i=1}^n 2(y_i - \hat{y}) \cdot [y_i - (w_1x^2 + w_2x + b)] \right|_{w_2}$$

$$\frac{\partial E}{\partial w_2}(w_1, w_2, b) = \sum_{i=1}^n 2(y_i - \hat{y}) \cdot (-x) \rightarrow \frac{\partial E}{\partial w_2}(w_1, w_2, b) = -2 \sum_{i=1}^n (y_i - \hat{y})x;$$

$$\frac{\partial E}{\partial b}(w_1, w_2, b) = \sum_{i=1}^n (y_i - (w_1x^2 + w_2x + b))^2 \quad \left| \frac{\partial E}{\partial b}(w_1, w_2, b) = \sum_{i=1}^n 2(y_i - \hat{y}) \cdot [y_i - (w_1x^2 + w_2x + b)] \right|_b$$

$$\frac{\partial E}{\partial b}(w_1, w_2, b) = \sum_{i=1}^n -2(y_i - \hat{y}) \rightarrow \frac{\partial E}{\partial b}(w_1, w_2, b) = -2 \sum_{i=1}^n (y_i - \hat{y})$$

$$\nabla E(w_1, w_2, b) = 0, \quad \left( -2 \sum_{i=1}^n (y_i - \hat{y})x^2, -2 \sum_{i=1}^n (y_i - \hat{y})x, -2 \sum_{i=1}^n (y_i - \hat{y}) \right) = 0$$

Figura 6: Derivada a mão 2 (Imagen Melhorada com IA)

**▼ Gradiente Descendente**

Parabola de um Conj. de Treinamento

Dado o conjunto de treinamento. Qual é a melhor parábola que aprova o conjunto de treinamento de modo a minimizar o erro?

$[(1, 4), (-1, 5), (2, 7), (-2, 8), (3, 12), (-3, 13), (4, 19), (-4, 20)]$

```
[71] 1 # Importações e Pré-Definições
     2 import matplotlib.pyplot as plt
     3 from tqdm import tqdm # Barra de Carregamento
     4 plt.style.use('ggplot')
     5
     6 # Dados de treinamento normalizados
     7 X = [1, -1, 2, -2, 3, -3, 4, -4]
     8 Y = [4, 5, 7, 8, 12, 13, 19, 20]
```

Figura 7: Dados e Bibliotecas

```

[72] 1 # Função direta do Gradiente Descendente
2 def GradDesc(w1, w2, b, lr):
3     erro, i = 1, 0
4     e = 6 # Elevado a 6, tipo um 10***-6
5     with tqdm() as pbar:
6         while round(erro, e) > round(1/3, e): # Quantas casas decimais de apro:
7
8             # Calculo do Erro
9             erro = ((w1 * (-1)**2 + w2 * (-1) + b) - 4)**2 + (
10                 (w1 * (-1)**2 + w2 * (-1) + b) - 5)**2 + (
11                 (w1 * (-2)**2 + w2 * (-2) + b) - 7)**2 + (
12                 (w1 * (-2)**2 + w2 * (-2) + b) - 8)**2 + (
13                 (w1 * (-3)**2 + w2 * (-3) + b) - 12)**2 + (
14                 (w1 * (-3)**2 + w2 * (-3) + b) - 13)**2 + (
15                 (w1 * (-4)**2 + w2 * (-4) + b) - 19)**2 + (
16                 (w1 * (-4)**2 + w2 * (-4) + b) - 20)**2
17
18             dw1 = -2 * (
19                 (-1)**2 * (4 - (w1 * (-1)**2 + w2 * (-1) + b))) + (
20                 (-1)**2 * (5 - (w1 * (-1)**2 + w2 * (-1) + b))) + (
21                 (-2)**2 * (7 - (w1 * (-2)**2 + w2 * (-2) + b))) + (
22                 (-2)**2 * (8 - (w1 * (-2)**2 + w2 * (-2) + b))) + (
23                 (-3)**2 * (12 - (w1 * (-3)**2 + w2 * (-3) + b))) + (
24                 (-3)**2 * (13 - (w1 * (-3)**2 + w2 * (-3) + b))) + (
25                 (-4)**2 * (19 - (w1 * (-4)**2 + w2 * (-4) + b))) + (
26                 (-4)**2 * (20 - (w1 * (-4)**2 + w2 * (-4) + b))) / 100
27
28             dw2 = -2 * (
29                 (-1) * (4 - (w1 * (-1)**2 + w2 * (-1) + b))) + (
30                 (-1) * (5 - (w1 * (-1)**2 + w2 * (-1) + b))) + (
31                 (-2) * (7 - (w1 * (-2)**2 + w2 * (-2) + b))) + (
32                 (-2) * (8 - (w1 * (-2)**2 + w2 * (-2) + b))) + (
33                 (-3) * (12 - (w1 * (-3)**2 + w2 * (-3) + b))) + (
34                 (-3) * (13 - (w1 * (-3)**2 + w2 * (-3) + b))) + (
35                 (-4) * (19 - (w1 * (-4)**2 + w2 * (-4) + b))) + (
36                 (-4) * (20 - (w1 * (-4)**2 + w2 * (-4) + b))) / 100
37
38             db = -2 * (
39                 (4 - (w1 * (-1)**2 + w2 * (-1) + b))) + (
40                 (5 - (w1 * (-1)**2 + w2 * (-1) + b))) + (
41                 (7 - (w1 * (-2)**2 + w2 * (-2) + b))) + (
42                 (8 - (w1 * (-2)**2 + w2 * (-2) + b))) + (
43                 (12 - (w1 * (-3)**2 + w2 * (-3) + b))) + (
44                 (13 - (w1 * (-3)**2 + w2 * (-3) + b))) + (
45                 (19 - (w1 * (-4)**2 + w2 * (-4) + b))) + (
46                 (20 - (w1 * (-4)**2 + w2 * (-4) + b))) / 100
47
48             # Atualização dos Coeficientes
49             w1 = w1 - lr * dw1
50             w2 = w2 - lr * dw2
51             b = b - lr * db
52             i += 1
53
54             # Barra de Progresso
55             pbar.set_description(f'{i:5} | Error: {erro:.30f} | w1: {w1:.{e}f} | '
56             pbar.update(1)
57
58     return w1, w2, b

```

The screenshot shows a Jupyter Notebook cell. The code is:

```
✓ 4s 1 w1, w2, b = GradDesc(2, 2, 2, 0.1)
2 print('\n\n• Coeficientes Otimizados')
3 print(f'w1 = {w1:.16f}')
4 print(f'w2 = {w2:.16f}')
5 print(f'b = {b:.16f}')
```

The output is:

```
1516 | Error: 0.333333498869728350744168210440 | w1
      • Coeficientes Otimizados
      w1 = 1.0000201594133327
      w2 = -0.16666666666666665
      b = 3.4997630914066944
```

Figura 9: Gradiente Descendente em Python

$$\hat{y} = x^2 + (-0.166\dots)x + 3.5$$

E realizando a seguinte visualização da parábola e dos pontos do conjunto de treinamento, chegamos a conclusão que encontramos a melhor função.

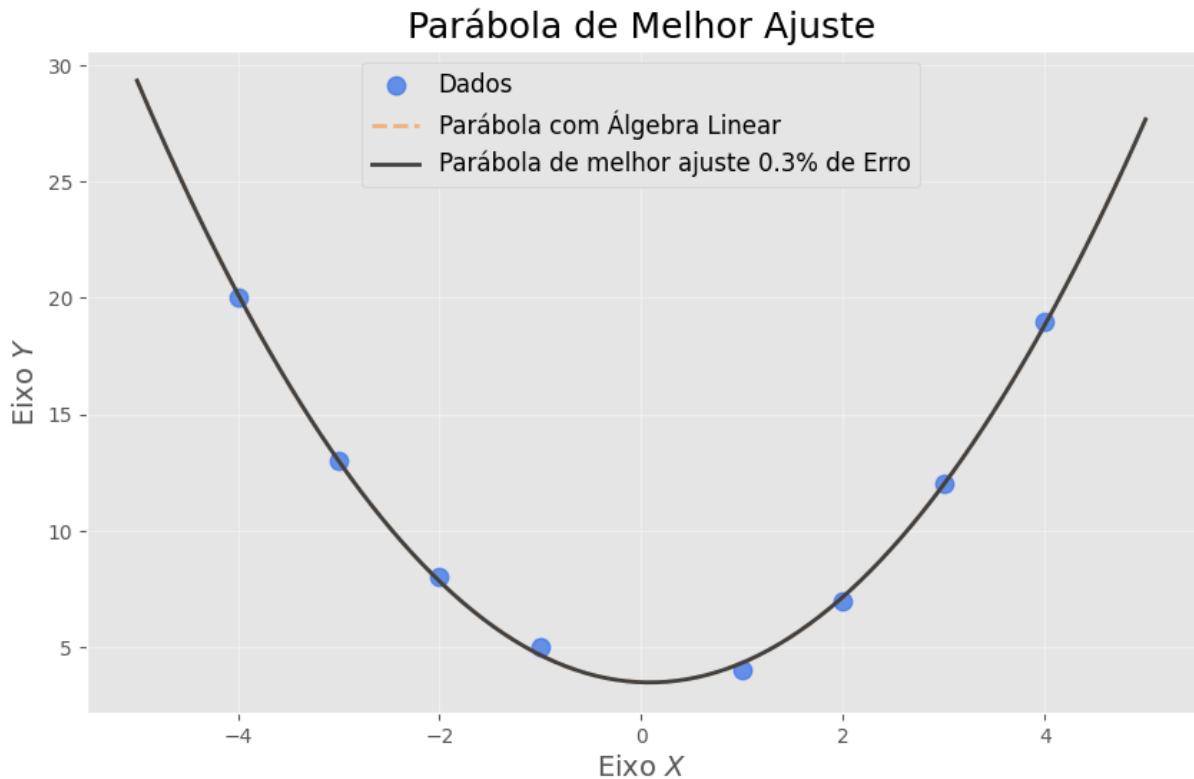


Figura 10: Melhor Parabola

### 3 Github

**Link do repositório no Github:** AnabelleSouza/Haskell  
**Repositório:** gabrielluizone/Gradient-Descent