

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

Gradiente Descendente

[HTML]9B9B9B		[HTML]9B9B9B CONTROLE DE VERSÃO		
[HTML]C0C0C0	[HTML]C0C0C0 Autor	[HTML]C0C0C0 Versão	[HTML]C0C0C0 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

1	Definição de Gradiente Descendente	4
2	Descrição do Gradiente Descendente para Ajuste de Parábola	4
2.1	Função do Gradiente Descendente	4
2.2	Execução e Impressão dos Resultados	4
3	Equações	5
4	Github	11

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 Descrição do Gradiente Descendente para Ajuste de Parábola

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. Dado o conjunto de treinamento, fornecido como uma lista de pontos no plano, representando as coordenadas (x, y) . Os pontos de treinamento são os seguintes:

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

2.1 Função do Gradiente Descendente

A função implementa o Gradiente Descendente para ajuste da parábola. Ela recebe como entrada os coeficientes iniciais w_1 , w_2 , b e a taxa de aprendizagem lr (γ).

Gradiente Descendente

Dentro da função, há um loop que itera até que o erro seja menor do que um determinado limiar predefinido. Em cada iteração, o algoritmo calcula o erro quadrático médio entre as previsões da parábola e os valores reais do conjunto de treinamento. A parábola é definida pela função $y = w_1x^2 + w_2x + b$, onde w_1 , w_2 e b são os coeficientes a serem otimizados.

Em seguida, são calculados os gradientes das variáveis w_1 , w_2 e b , usando as derivadas parciais da função de custo em relação a cada coeficiente. Esses gradientes indicam a direção de maior declive da função de custo, permitindo ajustar os coeficientes na direção de minimização do erro.

Os coeficientes são atualizados multiplicando a taxa de aprendizagem pelos gradientes calculados. Esse processo é repetido até que o erro seja menor do que o limiar definido.

Durante a execução do Gradiente Descendente, até a finalização é mostrando o número de iterações, o valor do erro, os valores atualizados dos coeficientes w_1 , w_2 e b .

2.2 Execução e Impressão dos Resultados

O algoritmo é executado chamando a função com os coeficientes iniciais e a taxa de aprendizagem desejada. Os coeficientes otimizados são então impressos na tela, mostrando

os valores finais de w_1 , w_2 e b .

3 Equações

A função da parábola a ser ajustada é definida por:

$$y = w_1x^2 + w_2x + b$$

O erro quadrático médio entre as previsões da parábola e os valores reais do conjunto de treinamento é calculado pela seguinte equação:

$$\text{erro} = \frac{1}{N} \sum_{i=1}^N (y_i - (w_1x_i^2 + w_2x_i + b))^2$$

Onde N é o número de pontos de treinamento, x_i é a coordenada x do ponto i e y_i é a coordenada y correspondente.

Os gradientes em relação aos coeficientes w_1 , w_2 e b são calculados pelas seguintes equações:

$$\begin{aligned}\frac{\partial \text{erro}}{\partial w_1} &= -\frac{2}{N} \sum_{i=1}^N x_i^2 (y_i - (w_1x_i^2 + w_2x_i + b)) \\ \frac{\partial \text{erro}}{\partial w_2} &= -\frac{2}{N} \sum_{i=1}^N x_i (y_i - (w_1x_i^2 + w_2x_i + b)) \\ \frac{\partial \text{erro}}{\partial b} &= -\frac{2}{N} \sum_{i=1}^N (y_i - (w_1x_i^2 + w_2x_i + b))\end{aligned}$$

Os coeficientes são atualizados de acordo com a seguinte regra:

$$\begin{aligned}w_1 &= w_1 - \text{lr} \frac{\partial \text{erro}}{\partial w_1} \\ w_2 &= w_2 - \text{lr} \frac{\partial \text{erro}}{\partial w_2} \\ b &= b - \text{lr} \frac{\partial \text{erro}}{\partial b}\end{aligned}$$

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.

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.

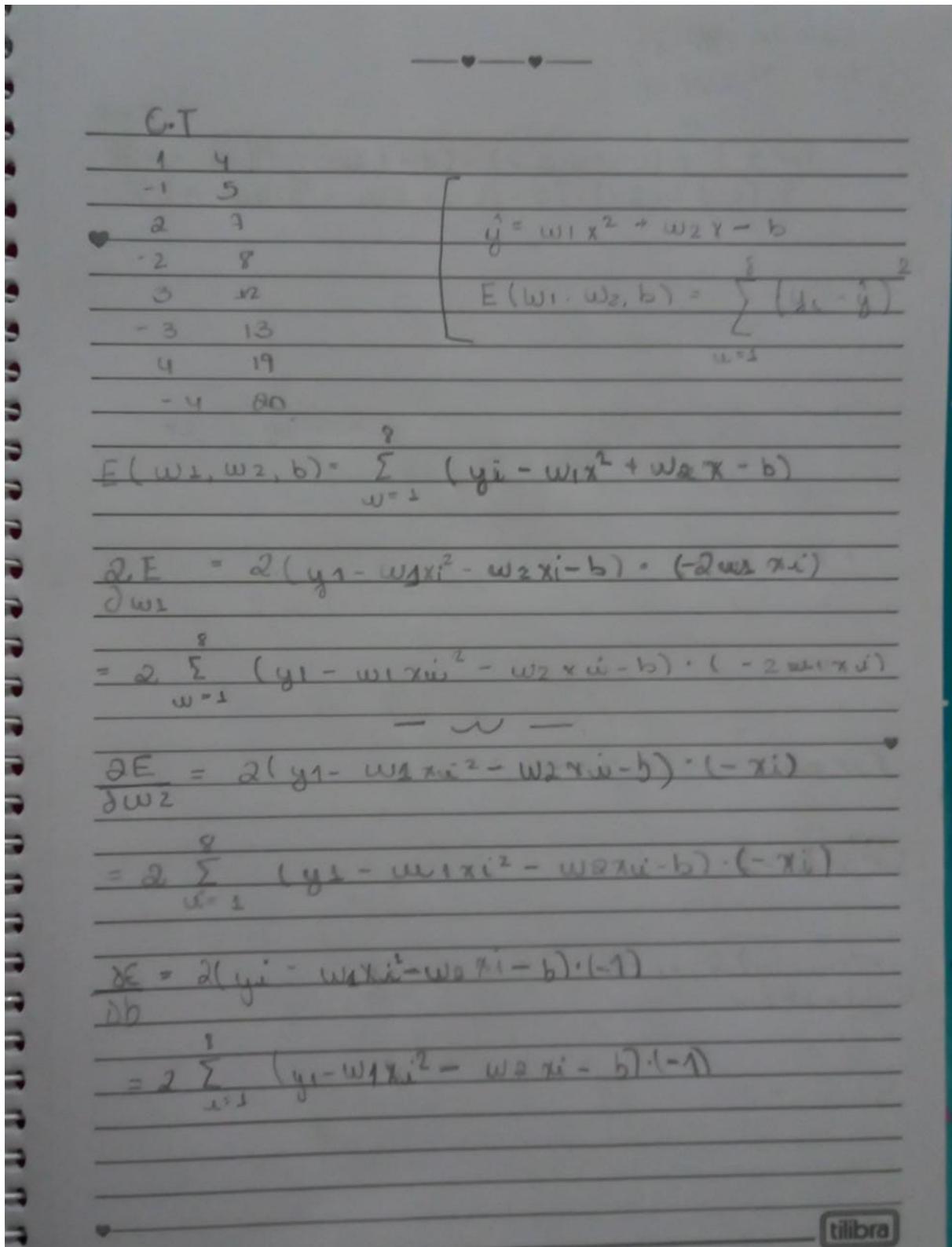


Figura 1: Derivada a mão 1

```

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 * 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)) * (-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)) * (-4) - b) * (-(-4)))
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 0.01 1.6 0 40 999 0
(1.0229299492608945,-0.1666666666666663,3.3285374790437743,64)

```

Figura 3: Teste1.

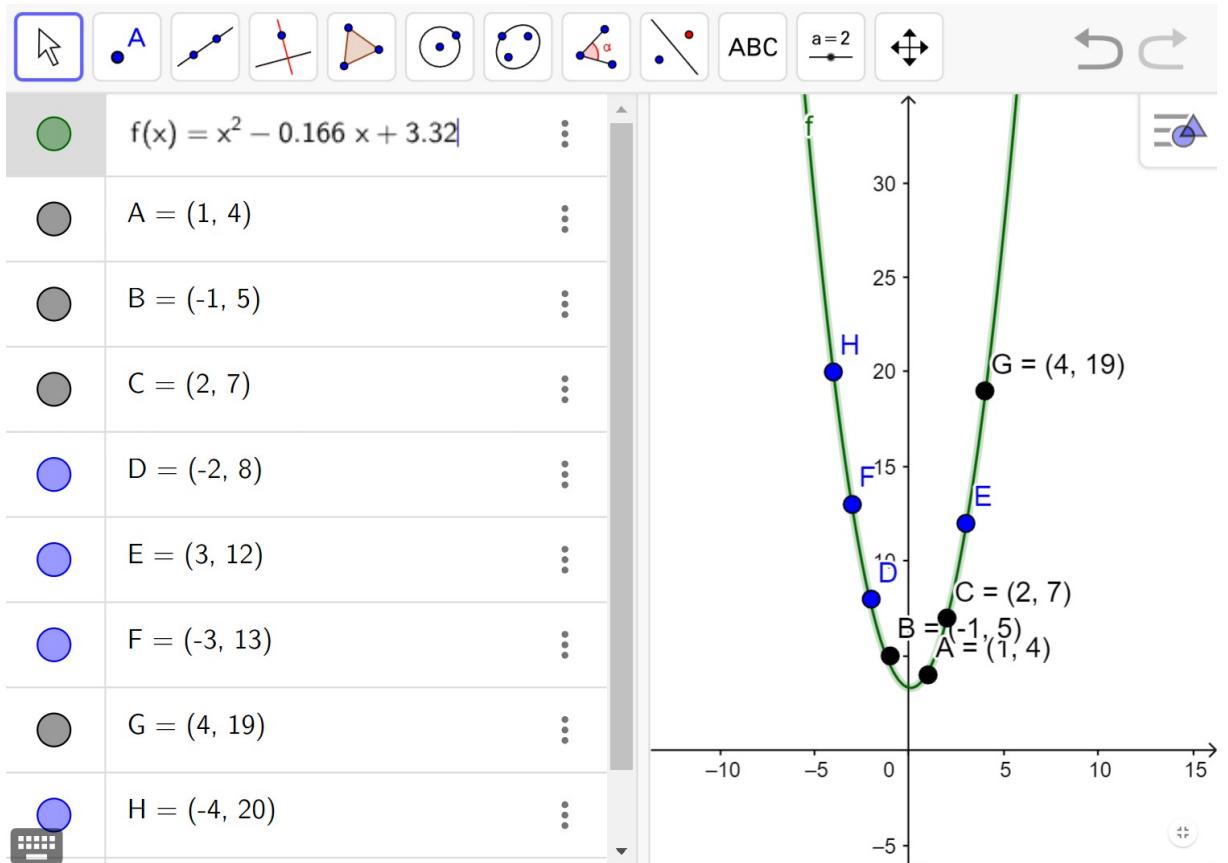


Figura 4: Geogebra.

Dado o conj. de treinamento $(1,4), (-1,5), (2,7), (-2,8), (3,12), (-3,13), (4,19), (-4,20)$

Qual é a melhor parábola que aprova o conj de modo a minimizar o erro

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

$$\frac{\partial E}{\partial w_1}(w_1, w_2, b) = \sum_{i=1}^8 (y_i - (w_1 x^2 + w_2 x + b))^2 \cdot \frac{\partial E}{\partial w_1}(w_1, w_2, b) = \sum_{i=1}^8 2(y_i - \hat{y}) \cdot [y_i - (w_1 x^2 + w_2 x + b)]$$

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

$$\frac{\partial E}{\partial w_1}(w_1, w_2, b) = \sum_{i=1}^8 (y_i - (w_1 x^2 + w_2 x + b))^2 \cdot \frac{\partial E}{\partial w_2}(w_1, w_2, b) = \sum_{i=1}^8 2(y_i - \hat{y}) \cdot [y_i - (w_1 x^2 + w_2 x + b)]$$

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

$$\frac{\partial E}{\partial b}(w_1, w_2, b) = \sum_{i=1}^8 (y_i - (w_1 x^2 + w_2 x + b))^2 \cdot \frac{\partial E}{\partial b}(w_1, w_2, b) = \sum_{i=1}^8 2(y_i - \hat{y}) \cdot [y_i - (w_1 x^2 + w_2 x + b)]$$

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

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

Figura 5: Derivada a mão 2 (Imagen Melhorada com IA)

Gradiante Descendente

Parábola 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 6: 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 = -0.1666666666666665
      b = 3.4997630914066944
```

Figura 8: 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.

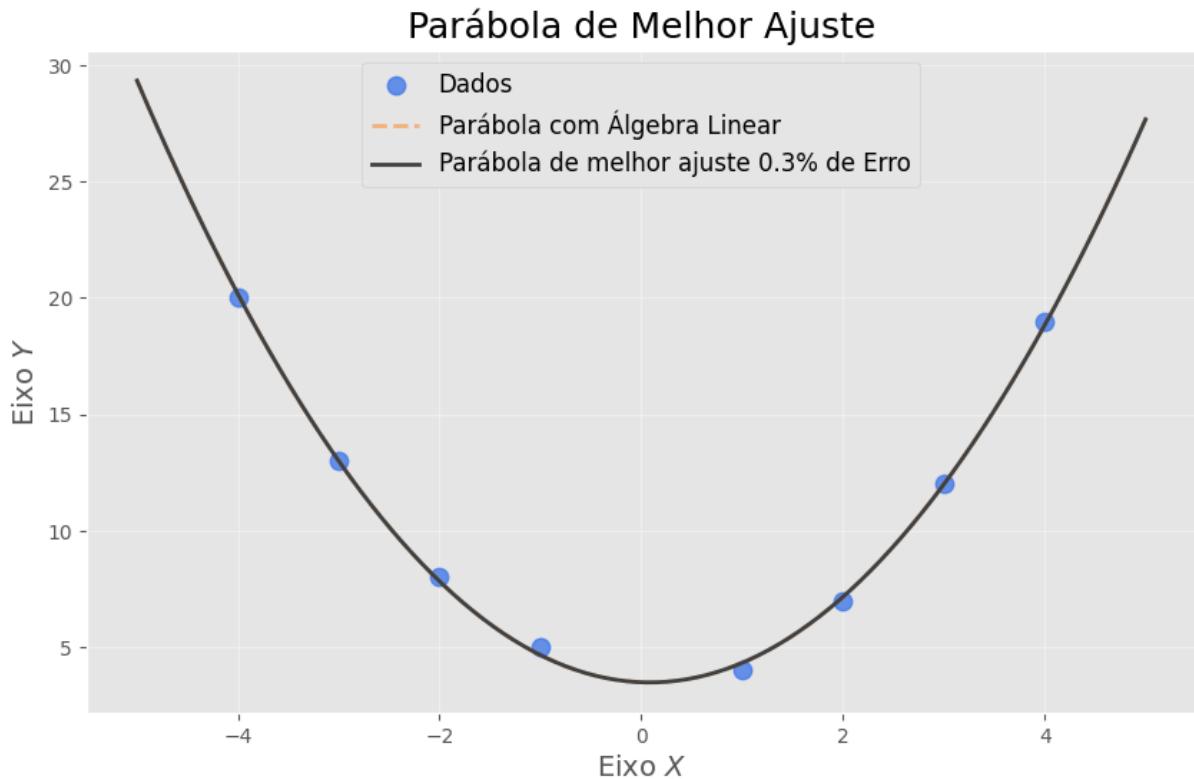


Figura 9: Melhor Parabola

4 Github

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