MO431 - Tarefa 2

Jônatas Trabuco Belotti
RA: 230260
jonatas.t.belotti@hotmail.com

I. Introdução

O objetivo desse trabalho é aplicar a técnica de otimização Descida do Gradiente para minimizar uma função de 3 dimensões. A Descida do Gradiente permite que o cálculo do gradiente seja feito de diversas formas, nesse trabalho foram exploradas duas: através da diferenciação simbólica e diferenciação automática.

Todos os códigos deste trabalho foram escritos utilizando a linguagem de programação **Python 3.6.9** com as seguintes bibliotecas: **Numpy 1.18.2**, **TensorFlow 2.1** e **Matplotlib 3.1.1**. Sendo estas obrigatórias para execução dos códigos. Todo o código desenvolvido pode ser acessado no repositório do GitHub `https://github.com/jonatastbelotti/mo431_trabalho2`.

II. Descida do Gradiente

O Método da Descida do Gradiente é um método iterativo que busca maximizar ou minimizar uma função. A cada iteração se faz uso do gradiente da função para dar um passo em direção ao mínimo ou máximo da função. O gradiente de uma função $f(x, y, z)$ definida em $\mathbb{R}^3$ é representado por ∇f e definida pela Equação 1.

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{bmatrix} \tag{1}$$

O gradiente $\nabla f(w_0)$ aplicado ao ponto w_0 é a direção em que a função mais cresce a partir do ponto w_0. Em contrapartida $-\nabla f(w_0)$ é a direção em que o ponto mais diminuiu em relação ao ponto w_0.

Portanto, se tratando de minimização de funções, estando no ponto w_i o próximo ponto w_{i+1} é dado pela Equação 2.

$$w_{i+1} = w_i - \in \nabla f(w_i) \tag{2}$$

tal que, w_i é o ponto atual, w_{i+1} é o próximo ponto, $\nabla f(w_i)$ é o gradiente aplicado ao ponto atual e $\in$ é o *Learning Rate* (Taxa de aprendizagem) que deve ser um número pequeno com o proposito de limitar o tamanho dos passos. O uso de um *Learning Rate* alto torna o algoritmo instável e impossibilita sua convergência. Do mesmo modo que um *Learning Rate* muito pequeno torna o processo de convergência extremamente demorado.

III. Estudo de caso

Nesse trabalho foi realizada a minimização da função *Rosenbrock* em 3D que é definida pela Equação 3. Essa é uma função conhecida, comumente utilizada como **benchmark** para algoritmos de otimização e tem seu ponto de mínimo em $(1, 1, 1) = 0$.

$$\begin{aligned} f(x_1, x_2, x_3) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \\ +100(x_3 - x_2^2)^2 + (1 - x_2)^2 \end{aligned} \tag{3}$$

Foram testadas duas variações da Descida do Gradiente, a primeira realiza o cálculo do gradiente de maneira explícita a partir da diferenciação simbólica (Seção III-A) e a segunda faz uso de uma biblioteca externa para realizar a diferenciação automática (Seção III-B).

Para cada variação foram testadas duas *Learning Rates*: $lr = 10^-3$ e $lr = 10^-4$. Além disso, o ponto inicial para a execução do algoritmo foi definido em $(0, 0, 0)$. Como critérios de parada foram definidos um número máximo de 20.000 passos e uma tolerância mínima de $tol = 10^{-4}$. A tolerância é definida pela Equação 4.

$$tol = \frac{|x_i - x_{i-1}|}{|x_{i-1}|} \tag{4}$$

de modo que x_i é o vetor $[x_1, x_2, x_3]$ com as coordenadas do ponto no passo i e $|x|$ é a Norma Euclidiana de um vetor.

A. Gradiente Explícito

Para essa variação do algoritmo o gradiente da função *Rosenbrock* foi calculado por diferenciação simbólica através da ferramenta *WolframAlpha* e pode ser observado na Equação 5.

$$\nabla f = \begin{bmatrix} 2(200x_1^3 - 200x_1x_2 + x_1 - 1) \\ -200x_1^2 + 400x_2^3 + x_2(202 - 400x_3) - 2 \\ 200(x_3 - x_2^2) \end{bmatrix} \tag{5}$$

Com $lr = 10^{-4}$ o algoritmo foi executado durante 5.893 passos e chegou ao ponto $x_1 = 0,7063967009819981$, $x_2 = 0,4983415476838231$ e $x_3 = 0,2455021231287078$ com valor $f(x_1, x_2, x_3) = 0,3387147658078193$. O gráfico da Figura 1 apresenta a evolução do valor da função de minimização durante os 5.893 passos da execução do algoritmo.

Observando o gráfico da Figura 1 nota-se que o decaimento da curva diminui conforme se passam as iterações.

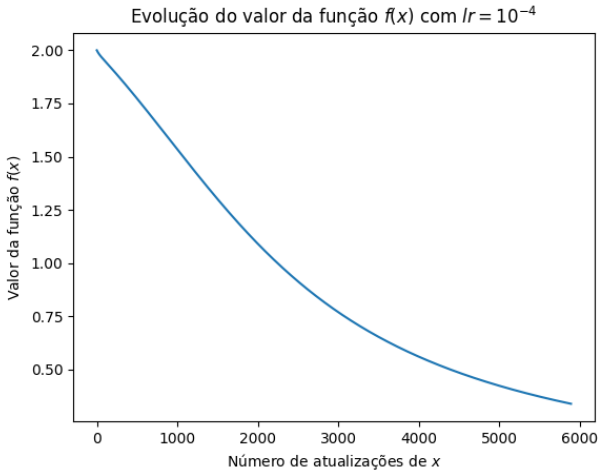

Fig. 1: Gráfico da evolução do valor da função com Gradiente Explícito e $lr = 10^{-4}$.

Em virtude desse fato o algoritmo da Descida do Gradiente demora tempo infinito para alcançar o mínimo da função. Justificando assim o uso dos critérios de parada.

Por sua vez, com $lr = 10^{-3}$ o Gradiente Descendente foi executado por 2.491 iterações, chegando ao ponto $x_1 = 0,9368876946913619$, $x_2 = 0,8775232152973693$ e $x_3 = 0,7694184586551830$ com valor $f(x_1, x_2, x_3) = 0,0190287698214329$, obtendo um resultado melhor que o uso de $lr = 10^{-4}$. O gráfico da Figura 2 apresenta a evolução da curva da função durante a execução do algoritmo.

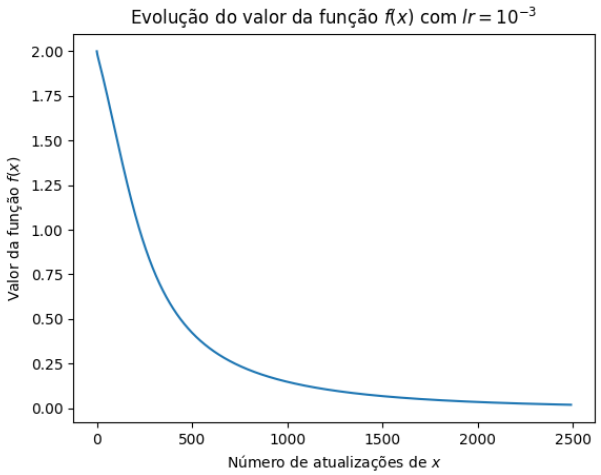

Fig. 2: Gráfico da evolução do valor da função com Gradiente Explícito e $lr = 10^{-3}$.

Analisando o comportamento da curva do gráfico da Figura 2 fica ainda mais evidente que a cada iteração o passo dado pelo algoritmo em direção ao mínimo é menor.

B. TensorFlow Para o Cálculo do Gradiente

A segunda variação da Descida do Gradiente testada fez uso da classe **GradientTape** do **TensorFlow** para calcular automaticamente o gradiente da função *Rosenbrock*.

Utilizando $lr = 10^{-4}$ a Descida do Gradiente com cálculo automático do gradiente pelo **TensorFlow** foi executado durante 5.893 iterações, chegando ao ponto $x_1 = 0,7063967003732921$, $x_2 = 0,4983415474623440$ e $x_3 = 0,2455021235601819$ com valor $f(x_1, x_2, x_3) = 0,3387147659331141$. O gráfico da Figura 3 apresenta a curva com a evolução do valor da função $f(x_1, x_2, x_3)$ durante a execução do algoritmo.

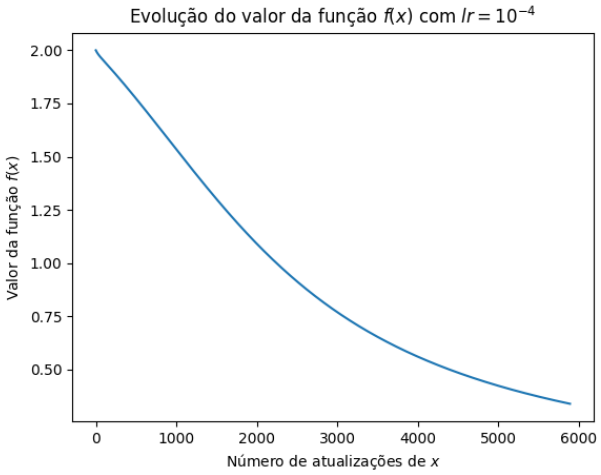

Fig. 3: Gráfico da evolução do valor da função com Gradiente Automático e $lr = 10^{-4}$.

No segundo teste, com $lr = 10^{-3}$, foram executadas 2.491 iterações do algoritmo. Tendo obtido o valor $f(x_1, x_2, x_3) = 0,0190287694413366$ no ponto $x_1 = 0,9368877094462518$, $x_2 = 0,8775232106745252$ e $x_3 = 0,7694184598415733$. Sendo este um resultado melhor que com o uso de $lr = 10^{-4}$. O gráfico da Figura 4 apresenta a evolução do valor da função *Rosenbrock* no decorrer da execução do algoritmo.

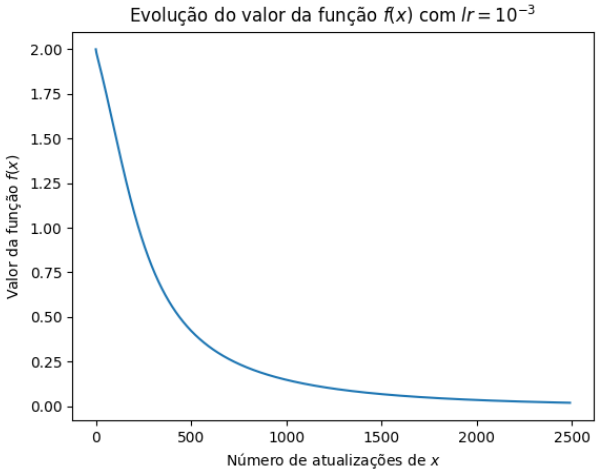

Fig. 4: Gráfico da evolução do valor da função com Gradiente Automático e $lr = 10^{-3}$.

Comparando os resultados desta seção com os da

Seção III-A é possível verificar que os valores obtidos pela diferenciação simbólica e a automática foram extremamente similares, se diferenciando apenas após a sétima casa decimal. Vale mencionar ainda que para os dois valores de *Leaning Rate* ambas as variações executaram o mesmo número de iterações.

C. Teste Learning Rate

Com o objetivo de exemplificar os efeitos que uma *Learning Rate* alta podem causar na execução do algoritmo foi realizado mais um teste com o uso do cálculo do gradiente explícito, dessa vez utilizando $lr = 10^{-2}$.

Diferentemente dos outros testes, onde foram executadas mais de 2.000 iterações do algoritmo, aqui foram executadas apenas 25, uma vez que na iterações 26 ocorreu um erro de cálculo e o valor da função foi para infinito. O gráfico da Figura 5 apresenta a evolução do valor do função durante a execução do algoritmo.

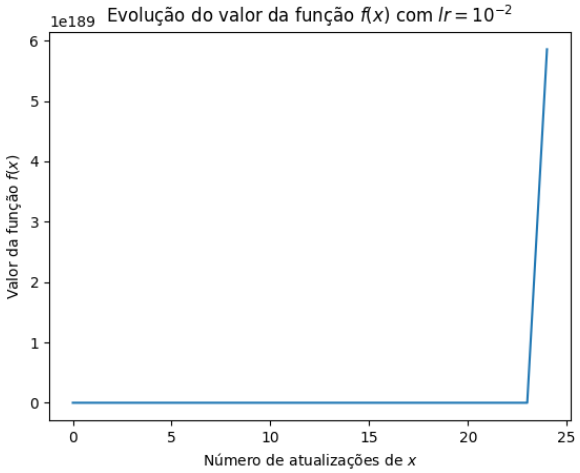

Fig. 5: Gráfico da evolução do valor da função com Gradiente Explícito e $lr = 10^{-2}$.

Observando o gráfico da Figura 5 a primeira coisa que se nota é que diferentemente dos gráficos das figuras 1, 2, 3 e 4 aqui o valor da função não diminuiu ao decorrer da execução, ele aumenta. Além disso, os valores obtidos foram altíssimos, uma vez que a escala do gráfico é de 1 para 1×10^{189}. Comprovando assim a necessidade da *Learning Rate* ser um valor pequeno.