Instituto Tecnológico de Aeronáutica

PO-249
INTRODUÇÃO ÀS REDES NEURAIS E AOS GRADENS MODELOS DE LINGUAGEM
ATIVIDADE 01

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

São José dos Campos, SP - Brasil
2025

1 Introdução

Parte fundamental do desenvolvimento e implementação de modelos de redes neurais está relacionada à escolha do algoritmo de otimização a ser utilizado, uma vez que ele será o responsável por determinar a velocidade do treinamento e também o desempenho que o modelo terá após o treinamento. Choi [1] aponta que não existe uma teoria que explique de forma adequada como realizar essa escolha. Em vez disso, a comunidade científica recorre principalmente a estudos empíricos [4] e a esforços de benchmarking sistemático [3].

Por isso é importante entender como os principais algoritmos de otimização funcionam, de modo a podermos usá-los de maneira mais eficiente de acordo com o problema que desejamos resolver. Nesta atividade iremos explicar e avaliar o Método do Gradiente Descendente, usando duas configurações distintas de "tamanho de passo".

2 Método do Gradiente Descendente

O Gradiente Descendente [2] é um dos algoritmos mais populares para realizar otimização e, também, a maneira mais comum de otimizar redes neurais.

Ele é uma forma de minimizar uma função objetivo $f(x_k)$ parametrizada pelos parâmetros de um modelo $k \in R^d$, atualizando os parâmetros na direção oposta do gradiente da função objetivo $\nabla f(x_k)$. A taxa de aprendizagem η determina o tamanho dos passos que damos para atingir um mínimo (local). O que é resumido na Equação 1.

$$x_{k+1} = x_k - \eta \nabla f(x_k) \tag{1}$$

O gradiente é dado pela Equação 2.

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \right] \tag{2}$$

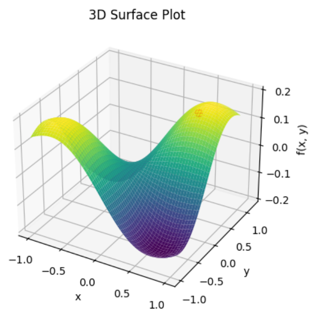
3 Formulação do Problema

3.1 Função Objetivo

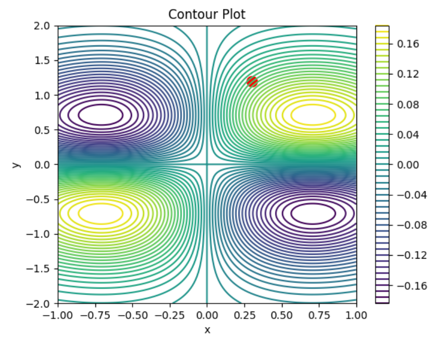
A função proposta para ser otimizada é dada por:

$$f(x, y) = xye^{-x^2-y^2} \tag{3}$$

A Figura 1 apresenta a representação gráfica de uma função de duas variáveis. O ponto em vermelho assinala a condição inicial adotada para o processo de otimização, definida em $(0.3, 1.2)$.



(a) Representação tridimensional da função



(b) Curvas de nível da função

Figura 1: Representação gráfica da função e do ponto inicial.

3.2 Gradiente da Função

A partir da Equação 3, observamos que a derivada parcial do termo $e^{-x^2-y^2}$ em relação a x é:

$$\frac{\partial}{\partial x} e^{-x^2-y^2} = -2x\, e^{-x^2-y^2}. \tag{4}$$

De forma análoga, em relação a y tem-se:

$$\frac{\partial}{\partial y} e^{-x^2-y^2} = -2y\, e^{-x^2-y^2}. \tag{5}$$

Assim, a derivada parcial de $f(x, y)$ em relação a x resulta em:

$$\begin{aligned} \frac{\partial f}{\partial x} &= y\, e^{-x^2-y^2} + xy\big(-2x\big)e^{-x^2-y^2} \\ &= y\, e^{-x^2-y^2}\, \big(1 - 2x^2\big). \end{aligned} \tag{6}$$

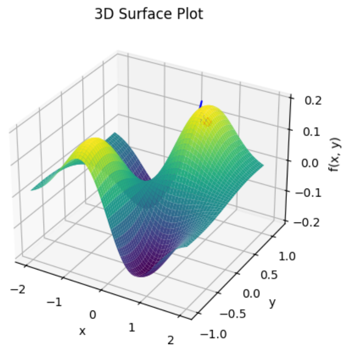
De modo análogo, a derivada parcial em relação a y é:

$$\begin{aligned} \frac{\partial f}{\partial y} &= x\, e^{-x^2-y^2} + xy\big(-2y\big)e^{-x^2-y^2} \\ &= x\, e^{-x^2-y^2}\, \big(1 - 2y^2\big). \end{aligned} \tag{7}$$

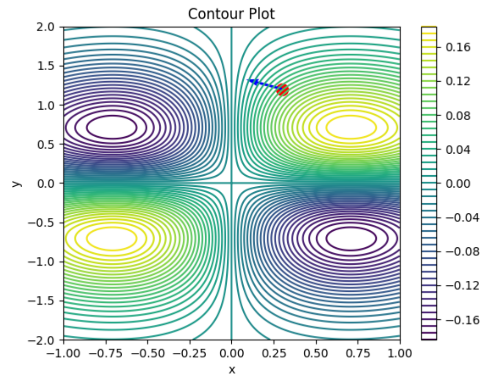
Portanto, o gradiente da função analisada é dado por:

$$\nabla f(x, y) = \Big(y\, e^{-x^2-y^2}\, (1 - 2x^2),\ x\, e^{-x^2-y^2}\, (1 - 2y^2)\Big). \tag{8}$$

A Figura 2 apresenta a representação gráfica do vetor gradiente, evidenciando sua direção em relação ao ponto inicial considerado.



(a) Representação tridimensional da função com vetor gradiente.



(b) Curvas de nível da função com vetor gradiente.

Figura 2: Representação gráfica da função e do vetor gradiente no ponto inicial.

4 Experimentos

4.1 Descrição dos Experimentos

Com o objetivo de compreender de forma mais aprofundada o comportamento do otimizador, foram conduzidos dois experimentos distintos. No primeiro, o tamanho do passo foi mantido fixo em 0.1. No segundo, o tamanho do passo foi ajustado dinamicamente a cada iteração, sendo determinado a partir da avaliação do vetor gradiente no ponto corrente. Em ambos os experimentos, adotou-se como critério de parada a verificação de que a norma do gradiente fosse inferior à tolerância estabelecida de 10^{-5}.

4.2 Implementação dos Otimizadores

Na sequência, são apresentados os algoritmos utilizados nos experimentos numéricos, implementados em linguagem Python.

O Algoritmo 1 corresponde ao método do Gradiente Descendente com passo fixo, no qual a atualização das variáveis é realizada de forma direta a partir de uma taxa de aprendizado previamente definida.

Algoritmo 1: Gradiente Descendente

Input: Valores iniciais x_0, taxa de aprendizado α, tolerância ε, número máximo de iterações N

Output: Solução x^*, número de iterações k, tempo decorrido t

$x \leftarrow x_0$;

$start \leftarrow$ tempo atual ;

for $k \leftarrow 1$ **to** N **do**

    $g \leftarrow \nabla f(x)$;

    $f \leftarrow f(x)$;

    $x \leftarrow x - \alpha g$;

    if $\|g\| < \varepsilon$ **then**

        $t \leftarrow$ tempo atual $- start$;

        return (x, k, t) ;

$t \leftarrow$ tempo atual $- start$;

return (x, N, t) ;

Já o Algoritmo 2 refere-se ao método do Gradiente Descendente com passo adaptativo, que emprega uma busca linear em cada iteração para determinar o tamanho de passo mais adequado na direção do gradiente. Esse procedimento garante uma convergência mais rápida e estável, reduzindo a necessidade de ajustar manualmente o parâmetro da taxa de aprendizado.

Algoritmo 2: Gradiente Descendente com Passo Adaptativo

```
Input: Valores iniciais x_0, tolerância ε, número máximo de iterações N
Output: Solução x*, número de iterações k, tempo decorrido t
x ← x_0 ;
start ← tempo atual ;
for k ← 1 to N do
    g ← ∇f(x) ;
    f ← f(x) ;
    if ||g|| < ε then
        t ← tempo atual − start ;
        return (x, k, t) ;
    α ← 1.0 ;
    c ← 10^−4 ;
    ρ ← 0.5 ;
    repeat
    until x_new ← x − αg ;
    f_new ← f(x_new) ;
    if f_new ≤ f − c α||g||^2 then
        break
    α ← ρ · α ;
    ;
    x ← x_new ;
t ← tempo atual − start ;
return (x, N, t) ;
```

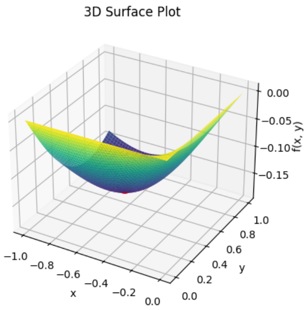
5 Resultados

O ponto final obtido em ambos os experimentos foi bastante semelhante. Para o método do Gradiente Descendente com passo fixo, o mínimo encontrado foi $(-0.70710235, 0.7071177)$, como ilustrado na Figura 3.

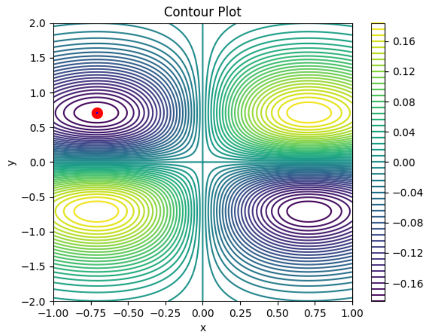
No caso do Gradiente Descendente com passo variável, o mínimo encontrado foi $(-0.70710605, 0.7071116)$, apresentado na Figura 4.

A principal diferença entre os dois experimentos foi o número de iterações necessárias para convergir. Para o passo fixo, foram necessárias 178 iterações, conforme ilustrado na Figura 5.

Já para o passo variável, foram necessárias apenas 15 iterações, como mostrado na

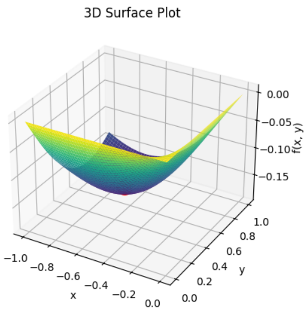


(a) Representação tridimensional da função com ponto de mínimo.

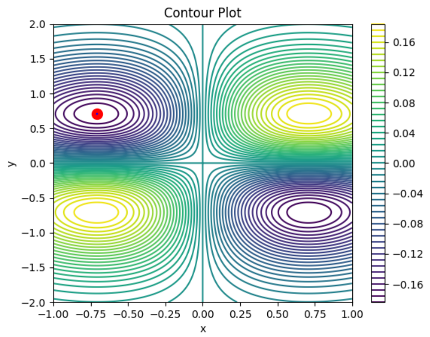


(b) Curvas de nível da função com ponto de mínimo.

Figura 3: Representação gráfica do ponto de mínimo para o método do Gradiente Descendente com passo fixo.



(a) Representação tridimensional da função com ponto de mínimo.



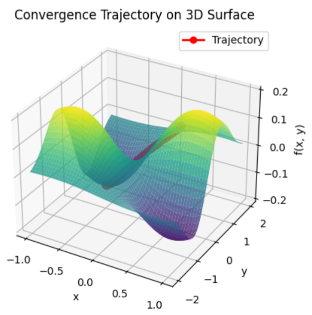
(b) Curvas de nível da função com ponto de mínimo.

Figura 4: Representação gráfica do ponto de mínimo para o método do Gradiente Descendente com passo variável.

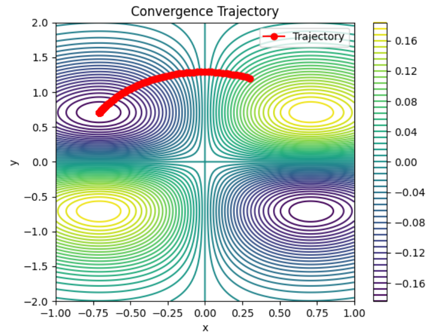
Figura 6.

6 Discussão

Os experimentos realizados demonstram que ambos os métodos de Gradiente Descendente, com passo fixo e com passo variável, convergem para pontos de mínimo muito próximos, indicando que a função analisada possui um único mínimo global na região estudada. Os valores obtidos para os mínimos foram $(-0.70710235, 0.7071177)$ para o passo fixo e $(-0.70710605, 0.7071116)$ para o passo variável, mostrando uma diferença praticamente desprezível.

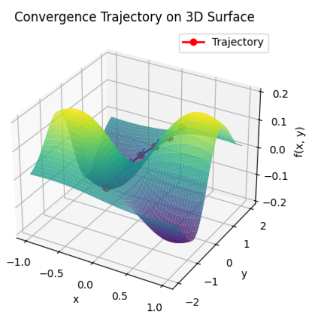


(a) Representação tridimensional da função com trajetória até o ponto de mínimo.

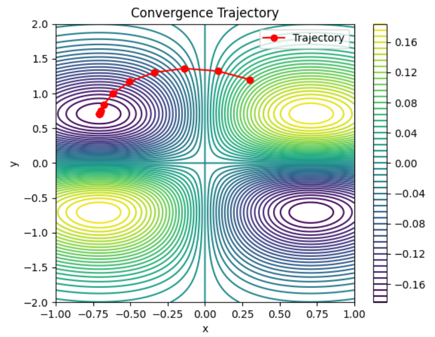


(b) Curvas de nível da função com trajetória até o ponto de mínimo.

Figura 5: Trajetória até o ponto de mínimo para o método do Gradiente Descendente com passo fixo.



(a) Representação tridimensional da função com trajetória até o ponto de mínimo.



(b) Curvas de nível da função com trajetória até o ponto de mínimo.

Figura 6: Trajetória até o ponto de mínimo para o método do Gradiente Descendente com passo variável.

A principal distinção entre os dois métodos reside no número de iterações necessárias para a convergência. Enquanto o Gradiente Descendente com passo fixo necessitou de 178 iterações, o método com passo adaptativo alcançou o mínimo em apenas 15 iterações, evidenciando a maior eficiência deste último. Este comportamento pode ser explicado pelo ajuste dinâmico do tamanho do passo em função do gradiente, permitindo que o algoritmo avance mais rapidamente em regiões suaves e reduza o passo em regiões íngremes, mantendo a estabilidade da convergência.

Em termos práticos, os resultados sugerem que, embora o passo fixo seja suficiente para encontrar o mínimo, a utilização de passo adaptativo apresenta uma vantagem significativa em termos de eficiência computacional, especialmente em funções de múltiplas variáveis ou com superfícies mais complexas.

Referências

[1] D Choi. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019.

[2] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[3] Frank Schneider, Lukas Balles, and Philipp Hennig. Deepobs: A deep learning optimizer benchmark suite. *arXiv preprint arXiv:1903.05499*, 2019.

[4] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. *Advances in neural information processing systems*, 30, 2017.

Anexos

Com o intuito de validar o funcionamento dos métodos de otimização discutidos, os algoritmos foram implementados em Python. A listagem de código a seguir apresenta a implementação dos otimizadores empregados nos experimentos numéricos, servindo como base para a análise dos resultados apresentados na Seção 5.

```python
import time
import numpy as np

from models.OptimizationResult import OptimizationResult
from optimizers.OptimizerBase import OptimizerBase


class GradientDescent(OptimizerBase):
    def optimize(self, initial_values: np.ndarray) ->
        OptimizationResult:
        """
        Perform gradient descent optimization.

        :param initial_values: Initial guess for the variables (
            NumPy array).
        :return: The optimized values of the variables and the
            function value at the optimized point.
        """
        x = np.array(initial_values, dtype=np.float64)
        start_time = time.time()

        for i in range(self.max_iters):
            self.log(f'Iteration {i}')
            # Evaluate the function and the gradient at the
                current point
            func_value = self.function.evaluate_function_at(x)
            self.log(f'\tf(x) = {func_value}')
            grad = self.function.evaluate_gradient_at(x)
            self.log(f'\tgrad(x) = {grad}')

            # Save history for plotting
            self.history['values'].append(x.copy())
            self.history['func_values'].append(func_value)
```

```
            self.history['grad_magnitudes'].append(np.linalg.norm(grad))

            # Update the variables using gradient descent rule
            self.log(f'\tx = x - lr * grad = {x} - {self.learning_rate} * {grad} = {x - self.learning_rate * grad}')
            x = x - self.learning_rate * grad

            # Stop if the gradient is small enough (converged)
            if np.linalg.norm(grad) < self.tolerance:
                elapsed_time = time.time() - start_time
                self.log(f"Converged after {i + 1} iterations.")
                return OptimizationResult(x, i+1, elapsed_time)

            self.log("")

        elapsed_time = time.time() - start_time
        self.log(f"Reached maximum iterations ({self.max_iters}) without convergence.")

        return OptimizationResult(x, self.max_iters, elapsed_time)
```

Código 1: Gradiente Descendente com passo fixo

```
import time
import numpy as np
from models.OptimizationResult import OptimizationResult
from optimizers.OptimizerBase import OptimizerBase


class SteepestDescent(OptimizerBase):
    def optimize(self, initial_values: np.ndarray) -> OptimizationResult:
        """
        Perform the steepest descent optimization using line search for step size.

        :param initial_values: Initial guess for the variables (
```

```
            NumPy array).
        :return: The optimized values of the variables.
        """
        x = np.array(initial_values, dtype=np.float64)
        start_time = time.time()

        for i in range(self.max_iters):
            self.log(f'Iteration {i}')

            # Evaluate the function and the gradient at the
                current point
            func_value = self.function.evaluate_function_at(x)
            self.log(f'\tf(x) = {func_value}')
            grad = self.function.evaluate_gradient_at(x)
            self.log(f'\tgrad(x) = {grad}')

            # Save history for plotting
            self.history['values'].append(x.copy())
            self.history['func_values'].append(func_value)
            self.history['grad_magnitudes'].append(np.linalg.norm
                (grad))

            # Check for convergence
            if np.linalg.norm(grad) < self.tolerance:
                elapsed_time = time.time() - start_time
                self.log(f"Converged after {i + 1} iterations.")
                return OptimizationResult(x, i+1, elapsed_time)

            # --- Line search along the negative gradient
                direction ---
            alpha = 1.0
            c = 1e-4
            rho = 0.5
            while True:
                x_new = x - alpha * grad
                f_new = self.function.evaluate_function_at(x_new)
                if f_new <= func_value - c * alpha * np.linalg.
                    norm(grad)**2:
                    break
```

```python
                alpha *= rho
            self.log(f'\tLine search found alpha = {alpha}')

            # Update the variables
            self.log(f'\tx = x - alpha * grad = {x} - {alpha} * {grad} = {x_new}')
            x = x_new

            self.log("")

        elapsed_time = time.time() - start_time
        self.log(f"Reached maximum iterations ({self.max_iters}) without convergence.")
        return OptimizationResult(x, self.max_iters, elapsed_time)
```

Código 2: Gradiente Descendente com passo adaptativo