Instituto Tecnológico de Aeronáutica

CT-213

Laboratório 3 - Otimização com Métodos de Busca Local

Aluno: Pedro Elardenberg Sousa e Souza
Professor: Marcos Ricardo Omena de Albuquerque Maximo

Conteúdo

1 Resumo

Neste Laboratório, foram implementados algoritmos de otimização baseados em busca local, a saber, Descida do Gradiente, *Hill Climbing* e *Simulated Annealing*. Os métodos foram testados em um problema em que se usa regressão linear para obter parâmetros físicos relativos ao movimento de uma bola. No caso, tratava-se apenas de um problema brinquedo, dado que esse problema especificamente tem solução analítica, já que o Método dos Mínimos Quadrados (MMQ) o resolve mais facilmente.

Os resultados foram então mostrados e discutidos neste relatório.

2 Introdução

O problema que se pretende resolver é a predição de movimento de uma bola. Um modelo comum para este problema é dado a partir da velocidade v em função do tempo t submetida a uma força de atrito ou de fricção f, conforme a equação 1:

$$v(t) = v_0 - ft \tag{1}$$

A otimização da busca, portanto, visa aprender o valor de f.

2.1 Descida de Gradiente

Quando se tem um modelo cujo coeficiente pode ser calculado analiticamente, a descida do gradiente mostra-se uma boa solução [1]. A ideia é que o gradiente dá a direção de máximo crescimento da função. Logo, se o algoritmo seguir na direção contrária à do gradiente (máximo decrescimento), ele convergirá para um mínimo local. Tendo um vetor de parâmetros θ e uma função de custo $J(\theta)$, a descida do gradiente pode ser expressa como:

$$\theta_{k+1} = \theta_k - \alpha \frac{\partial J(\theta)}{\partial \theta} \tag{2}$$

Em que α é um hiperparâmetro, ou seja, um parâmetro do meu algoritmo que também precisa ser otimizado para que ele funcione. Neste problema, α é a taxa de aprendizado, ou seja, o quanto meu algoritmo "caminha"na direção do gradiente calculado naquela iteração.

2.2 *Hill Climbing*

A ideia do *Hill Climbing* é: dado um estado inicial, o algoritmo avalia os vizinhos dessa posição e vai para o vizinho melhor avaliado até atingir o critério de parada. Possui a vantagem de de não requerer cálculo de derivada e também funciona quando o espaço de busca é discreto.

2.3 *Simulated Annealing*

Método semelhante ao *Hill Climbing*, mas que permite transição para estados piores. É definido um hiperparâmetro chamado temperatura que, para cada iteração, assume um valor cada vez menor e que, para cada vizinho do ponto analisado, há uma chance de o algoritmo assumir um vizinho cuja função de custo é pior do que a anterior baseado num cálculo dessa temperatura.

Exemplos de *schedule* de temperatura:

$$\begin{aligned} T_i &= T_0\beta^i \\ T_i &= \frac{T_0}{1+\beta i} \end{aligned} \tag{3}$$

Em que β é um hiperparâmetro.

3 Análise dos Resultados

No caso de teste, o problema específico a ser resolvido é a determinação do coeficiente de desaceleração de uma bola em movimento num campo de futebol de robôs. A bola em movimento perde energia devido a um fenômeno conhecido como *rolling friction*. Pode-se determinar o coeficiente de desaceleração seguindo os seguintes passos:

- Usar câmera e visão computacional para obter posições da bola em cada instante.
- Calcular velocidades em e usando diferenças finitas centradas (exceto no primeiro e último elementos, em que deve-se usar derivadas forward e backward, respectivamente):

$$\begin{aligned} v_x[k] &= (x[k+1] - x[k-1])/(t[k+1] - t[k-1]) \\ v_y[k] &= (y[k+1] - y[k-1])/(t[k+1] - t[k-1]) \end{aligned} \tag{4}$$

- Calcular $v[k]$:

$$v[k] = \sqrt{v_x^2 + v_y^2} \tag{5}$$

- Obter v_0 e f através de uma otimização com função de custo:

$$J[v_0, f] = \sum_{k=1}^{n} (v_0 + ft[k] - v[k])^2 \tag{6}$$

Dessa forma, tem-se que os parâmetros a serem otimizados são $\theta_0 = v_0$ e $\theta_1 = f \rightarrow \theta = [v_0 \ f]^T$.

Todas as implementações dos algoritmos a seguir foram feitas da forma mais genérica possível, de modo que as informações do problema do *fit* da bola apenas surgem na chamada do método que calcula a função de custo.

3.1 Descida de Gradiente

No código *gradient_descent.py*, foi implementada a função *gradient_descent()*. Ela verifica o critério de convergência segundo a função de custo dada na equação 6 e calcula o vetor de parâmetros $\theta = [v_0 \ f]^T$ com a descida de gradiente conforme a equação 2. O resultado é dado na figura 1

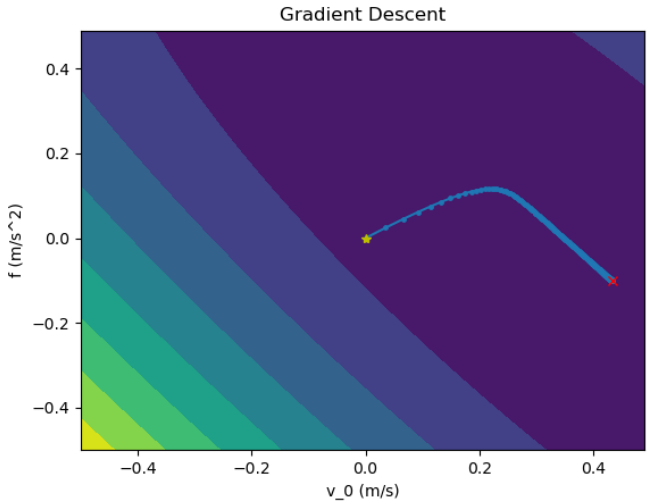

Figura 1: Resultado da busca pelo algoritmo Descida de Gradiente

Como trata-se de um método analítico e cuja função é derivável em todos os pontos, observa-se uma "descida"suave do gradiente até o ponto ótimo da função.

3.2 *Hill Climbing*

A implementação do *Hill Climbing* realizada neste laboratório é mostrada abaixo:

```
while j[theta[0], theta[1]] > epsilon and iteration < max_iterations:
    best = [None, None]
    for neighbor in neighbors(theta):
        j[neighbor[0], neighbor[1]] = cost_function(neighbor)
        if j[neighbor[0], neighbor[1]] < j[best[0], best[1]]:
            best = neighbor

    if j[best[0], best[1]] > j[theta[0], theta[1]]:
        return theta, history
```

```
        theta = best
        history.append(theta)
        iteration += 1
```

Como a função *neighbors()* avalia os vizinhos 8-conectados ao ponto da iteração, ou seja, os pontos nas 4 direções vertical e horizontal e as respectivas diagonais, a "descida"da função apresenta um formato de segmentos de reta justapostos, em oposição ao caminho sempre suave da descida de gradiente. Entretanto, o método também converge para o ótimo global, como se observa na figura 2.

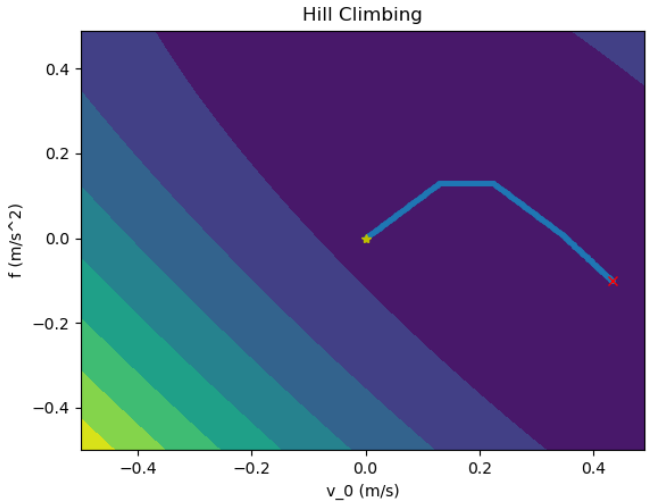

Figura 2: Resultado da busca pelo algoritmo *Hill Climbing*

3.3 *Simulated Annealing*

A implementação do *Simulated Annealing* realizada neste laboratório é mostrada abaixo:

```
    while j[theta[0], theta[1]] > epsilon and iteration < max_iterations:
        T = schedule(iteration)

        if T < 0:
            return theta, history

        neighbor = random_neighbor(theta)
        j[neighbor[0], neighbor[1]] = cost_function(neighbor)
```

```
deltaE = j[theta[0], theta[1]] - j[neighbor[0], neighbor[1]]

if deltaE > 0:
    theta = neighbor
else:
    r = random.uniform(0.0, 1.0)

    if r <= exp(deltaE/T):
        theta = neighbor

history.append(theta)
iteration += 1
```

A função *schedule* é dada pela segunda versão do cálculo de T_i da equação 3. Como esse algoritmo apresenta a ideia do parâmetro temperatura para tentar fugir de ótimos locais, a função tem um comportamento bastante ruidoso, principalmente nas primeiras iterações. O método, entretanto, também converge para o ótimo global, conforme figura 3.

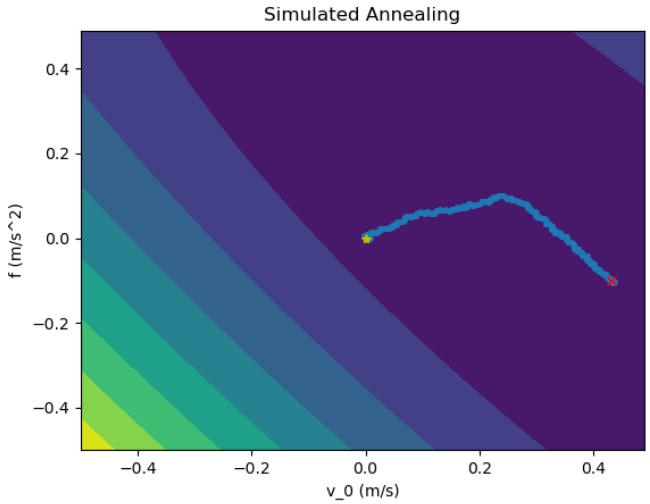

Figura 3: Resultado da busca pelo algoritmo *Simulated Annealing*

3.4 Comparação com Método dos Mínimos Quadrados

Os algoritmos implementados foram comparados com a solução dos mínimos quadrados, e o gráfico das soluções está mostrado na figura 4.

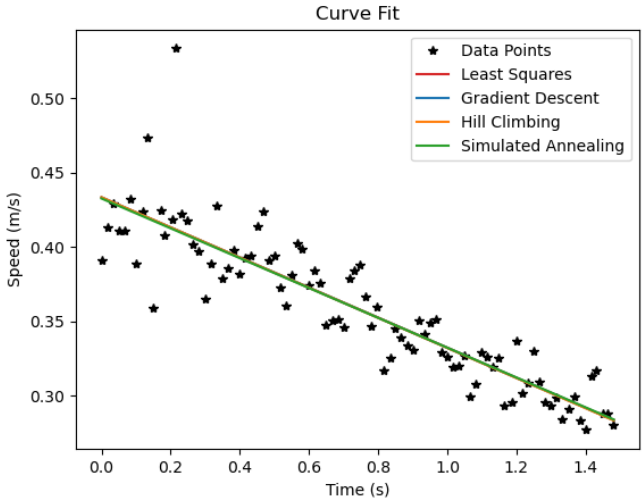

Figura 4: Comparação dos métodos analisados

Na figura, pode-se ver uma única reta, mas porque todas estão antepostas, demonstrando que todas chegaram ao mesmo resultado. Uma visualização numérica dos valores encontrados é mostrada na figura 5

```
(base) elardenberg@elardenberg-Inspiron-7572:/media/elardenberg/DATA/Documentos/COMP/CT-213/ai_for_robotics/Laboratório
3 - Otimização com Métodos de Busca Local$ python3 ball_fit.py
Least Squares solution:  [ 0.43337277 -0.10102096]
Gradient Descent solution:  [ 0.43337067 -0.10101846]
Hill Climbing solution:  [ 0.43341125 -0.10119596]
Simulated Annealing solution:  [ 0.43261732 -0.1002254 ]
```

Figura 5: Comparação dos métodos analisados

4 Conclusão

O presente laboratório mostrou que algoritmos de otimização com métodos de busca local são eficazes para otimização de parâmetros de aprendizado de máquina. Embora esse seja um problema considerado simples, dado que a solução ótima pode ser encontrada com o método dos mínimos quadrados, a utilização desses algoritmos mostra que eles são capazes também de resolver problemas mais genéricos.

Referências

[1] Marcos Ricardo Omena de Albuquerque Maximo. Ct-213 - aula 3 - métodos de otimização de busca local. Apostila, 2020. Curso CT-213 - Inteligência Artificial para Robótica Móvel, Instituto Tecnológico de Aeronáutica.