Lecture Note: Introduction to Artificial Intelligence
Bài Giảng: Nhập Môn Trí Tuệ Nhân Tạo

Nguyễn Quản Bá Hồng*

Ngày 14 tháng 5 năm 2025

Tóm tắt nội dung

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:
URL: https://nqbh.github.io/advanced_STEM/.
Latest version:

- *Lecture Note: Introduction to Artificial Intelligence – Bài Giảng: Nhập Môn Trí Tuệ Nhân Tạo.*
  PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/AI/lecture/NQBH_introduction_AI_lecture.pdf.
  TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/AI/lecture/NQBH_introduction_AI_lecture.tex.
- *Codes*:
  - C++: https://github.com/NQBH/advanced_STEM_beyond/tree/main/AI/C++.
  - Python: https://github.com/NQBH/advanced_STEM_beyond/tree/main/AI/Python.

Mục lục

1 Basic

1.1 Gradient – Độ dốc

Resources – Tài nguyên.

1. [Tiệ25]. Vũ Hữu Tiệp. *Machine Learning Cơ Bản*. Chap. 12: Gradient Descent.

Ví dụ 1 ([Tiệ25], p. 160)**.** *Xét hàm số $f(x) = x^2 + 5\sin x$, $f \in C(\mathbb{R}$ có đạo hàm $f'(x) = 2x + 5\cos x$. Giả sử xuất phát từ 1 điểm x_0, quy tắc cập nhật tại vòng lặp thứ t là*

$$x_{t+1} = x_t - \eta f'(x_t) = x_t - \eta(2x_t + 5\cos x_t).$$

Codes:

- *Python:*

*A scientist- & creative artist wannabe, a mathematics & computer science lecturer of Department of Artificial Intelligence & Data Science (AIDS), School of Technology (SOT), UMT Trường Đại học Quản lý & Công nghệ TP.HCM, Hồ Chí Minh City, Việt Nam.
E-mail: nguyenquanbahong@gmail.com & hong.nguyenquanba@umt.edu.vn. Website: https://nqbh.github.io/. GitHub: https://github.com/NQBH.

```
import math
import numpy as np

# f(x) = x^2 + 5sin x
def f(x):
    return x**2 + 5*np.sin(x)

def df(x): # derivative f'(x) of f(x)
    return 2*x + 5 * np.cos(x)

x = float(input("x = "))
print("f(x) = ", f(x))
print("df(x) = ", df(x))

tol = 1e-3 # tolerance: just a small number

def gradient_descent(x0, eta): # x0: starting point, eta: learning rate
    x = [x0]
    for i in range(100):
        x_new = x[-1] - eta*df(x[-1]) # x_new: x_{t+1}, x[-1]: x_t
        if abs(df(x_new)) < tol:
            break
        x.append(x_new)
    return(x, i)

x0 = float(input("x0 = "))
eta = float(input("eta = "))
if eta <= 0:
    print("error: eta must be positive!")
else:
    print(gradient_descent(x0, eta))
```

Bài toán 1. *Xét hàm số* $f(x) = x^3 + 3x^2 + 5\sin x - 7\cos x + \sqrt{2}e^{-2x}$. *Viết chương trình* C/C++, Python *để: (a) Tính hàm* $f(x), f'(x)$ *với* $x \in \mathbb{R}$ *được nhập từ bàn phím. (b) Viết hàm gradient descent theo công thức*

$$x_{t+1} = x_t - \eta f'(x_t),$$

với $\eta \in (0, \infty)$ *được gọi là* tốc độ học (learning rate).

Chứng minh. Dễ thấy $f(x)$ là 1 hàm liên tục trên $\mathbb{R}$, i.e., $f \in C(\mathbb{R}$, & có đạo hàm $f'(x) = 3x^2 + 6x + 5\cos x + 7\sin x - 2\sqrt{2}e^{-2x}$.
Code Python:

```
# f1(x) = x^3 + 3x^2  + 5sin x - 7cos x + sqrt{2}e^{-2x}
def f1(x):
    return x**3 + 3*x**2 + 5*np.sin(x) - 7*np.cos(x) + np.sqrt(2)*np.exp(-2*x)

def df1(x):
    return 3*x**2 + 6*x + 5*np.cos(x) + 7*np.sin(x) - 2*np.sqrt(2)*np.exp(-2*x)

x = float(input("x = "))
print("f(x) = ", f(x))
print("df(x) = ", df(x))

tol = 1e-3 # tolerance: just a small number

def gradient_descent_f1(x0, eta): # x0: starting point, eta: learning rate
    x = [x0]
    for i in range(100):
        x_new = x[-1] - eta*df1(x[-1]) # x_new: x_{t+1}, x[-1]: x_t
        if abs(df1(x_new)) < tol:
            break
        x.append(x_new)
    return(x, i)
```

```
x0 = float(input("x0 = "))
eta = float(input("eta = "))
if eta <= 0:
    print("error: eta must be positive!")
else:
    print(gradient_descent_f1(x0, eta))
```

□

Remark 1. *Có thể tham khảo các công thức tính đạo hàm ở Wikipedia/tables of derivatives.*

Bài toán 2. *Xét hàm số* $f(x,y) = 2x^3y^2 + \dfrac{\sqrt{x^3}}{y} + \sin(x^2y) + e^{\cos(xy^2)}$. *Viết chương trình* C/C++, Python *để: (a) Tính hàm* $f(x,y), \nabla f(x,y)$ *với* $x, y \in \mathbb{R}$ *được nhập từ bàn phím. (b) Viết hàm gradient descent cho 2 trường hợp:*

$$(x_{t+1}, y_{t+1}) = (x_t, y_t) - \eta \nabla f(x_t, y_t),$$

or

$$\begin{cases} x_{t+1} = x_t - \boldsymbol{\alpha} \cdot \nabla f(x_t, y_t) = x_t - \alpha_1 \partial_x f(x_t, y_t) - \alpha_2 \partial_x f(x_t, y_t), \\ y_{t+1} = y_t - \boldsymbol{\beta} \cdot \nabla f(x_t, y_t) = x_t - \beta_1 \partial_x f(x_t, y_t) - \beta_2 \partial_x f(x_t, y_t), \end{cases}$$

Python:

```
# f(x,y) = 2x^3y^2 + sqrt(x^3)/y + sin(x^2y) + e^{cos(xy^2)}

def f(x, y):
    return 2*x**3*y**2 + np.sqrt(x**3)/y + np.sin(x**2 * y) + np.exp(np.cos(x * y**2))

def grad_f(x, y):
    df_dx = 6*x**2 * y**2 + (3/2) * x**0.5 / y + 2*x*y * np.cos(x**2 * y) - y**2 * np.sin(x * y**2) * np.exp(np.
    df_dy = 4*x**3 * y - np.sqrt(x**3) / y**2 + x**2 * np.cos(x**2 * y) - 2*x*y * np.sin(x * y**2) * np.exp(np.c
    return np.array([df_dx, df_dy])

x = float(input("x = "))
y = float(input("y = "))
print("f(x,y) = ", f(x,y))
print("grad f(x,y) = ", grad_f(x,y))
```

2 Minimax

Resources – Tài nguyên.

1. Wikipedia/minimax.

Minimax (sometimes *Minmax, MM, or saddle point*) is a decision rule used in AI, decision theory, game theory, statistics, & philosophy for *minimizing* the possible loss for a worse case (maximum loss) scenario. When dealing with gains, it is referred to as "maximin" – to maximize the minimum gain. Originally formulated for several-player zero-sum game theory, covering both the cases where players take alternate moves & those where they make simultaneous moves, it has also been extended to more complex games & to general decision-making in the presence of uncertainty.

– *Minimax* (đôi khi là *Minmax, MM*, hoặc *điểm yên ngựa*) là 1 quy tắc quyết định được sử dụng trong trí tuệ nhân tạo, lý thuyết quyết định, lý thuyết trò chơi, thống kê và triết học để giảm thiểu tổn thất có thể xảy ra cho một kịch bản xấu nhất (tổn thất tối đa). Khi giải quyết các khoản lợi nhuận, nó được gọi là "maximin" – để tối đa hóa lợi nhuận tối thiểu. Ban đầu được xây dựng cho lý thuyết trò chơi tổng bằng không của nhiều người chơi, bao gồm cả các trường hợp mà người chơi thực hiện các nước đi thay thế và các trường hợp mà họ thực hiện các nước đi đồng thời, nó cũng đã được mở rộng sang các trò chơi phức tạp hơn và ra quyết định chung khi có sự không chắc chắn.

3 Giải Bài Toán Phức Tạp Với Các Thuật Giải Heuristic

3.1 Roster problem – Bài toán phân công

Dạng toán 1. *Cài đặt & đánh giá thực nghiệm 1 thuật giải heuristic cho bài toán phân công công việc (đơn giản), & thuật giải cải tiến.*

Bài toán 3 (Roster – Bài toán phân công đơn giản). *1 đề án gồm $n \in \mathbb{N}^\star$ công việc & các việc sẽ được thực hiện bởi $m \in \mathbb{N}^\star$ máy như nhau. Giả sử biết thời gian để 1 máy thực hiện việc thứ i là t_i. Yêu cầu: Tìm phương án phân công sao cho thời gian hoàn thành toàn bộ công việc là thấp nhất.*

Input. *m: số máy, n: số việc, dãy $t[0], \ldots, t[n-1]$ với $t[i]$: thời gian để 1 máy thực hiện việc i.*

Output. *Bảng phân công tối ưu.*

Sample.

roster.inp	roster.out
3 10 4 9 5 2 7 6 10 8 7 5	

Thuật giải cho bài toán phân công đơn giản – Pseudocode.

Mathematical analyse – Phân tích Toán học. Gọi n công việc là $w_1, \ldots, w_n$ (w: work), gọi m máy là $M_1, \ldots, M_m$ (các máy này có công suất làm việc như nhau). Yêu cầu của bài toán: Phân hoạch tập $\{t_i\}_{i=1}^n$ thành m tập con $T_1, \ldots, T_m$ lần lượt có số phần tử là $n_1, \ldots, n_m$, i.e., $|T_i| = n_i$, $\forall i = 1, \ldots, m$. □

Computer Science analyse – Phân tích Tin học. □

```
for (i = 0; i < n; i++) {
    chọn việc i chưa phân công có thời gian thực hiện cao nhất;
    chọn máy m có thời gian làm việc thấp nhất;
    bố trí việc i cho máy m;
}
```

- Input: https://github.com/NQBH/advanced_STEM_beyond/blob/main/AI/Python/roster.inp.
- Output: https://github.com/NQBH/advanced_STEM_beyond/blob/main/AI/Python/roster.out.
- Python: https://github.com/NQBH/advanced_STEM_beyond/blob/main/AI/Python/roster.py.

```
m, n = map(int, input().split())
t = [int(x) for x in input().split()]
d = [0] * m # devices/machines's current accomplished time
t.sort(reverse = True) # descending order
for i in range(n):
    current_max_work = t[0] # current longest work
    t.pop(0) # remove current longest work
    # print(t)
    d.sort() # ascending order
    # print(d)
    d[0] += int(current_max_work) # laziest device takes longest work
    # print(d)
print(max(d))
```

Các bước `print` để mô phỏng quá trình giao công việc cho các máy để tiện hình dung, không bắt buộc.

Bài toán 4 (Extended roster – Bài toán phân công mở rộng). *Có $n \in \mathbb{N}^\star$ công việc & $m \in \mathbb{N}^\star$ máy không đồng nhất. Biết thời gian máy i làm việc j là $t_{ij} = t[i][j]$. Yêu cầu: Lập bảng phân công tối ưu.*

Input. *m: số máy, n: số việc, array 2 chiều $t[i][j]$: thời gian để máy i thực hiện việc j.*

Output. *Bảng phân công tối ưu.*

Sample.

extended_roster.inp	extended_roster.out
3 8 4 5 4 10 8 6 12 8 7 5 7 3 9 7 9 5 10 6 7 12 10 6 5 7	

Cách phát biểu khác của bài toán phân công mở rộng. Có $n \in \mathbb{N}^\star$ công việc sẽ được phân công cho $m \in \mathbb{N}^\star$ người thực hiện, mỗi việc được phân công cho 1 người. Giả sử ta biết thời gian $t_{ij} = t[i][j]$ cần để người thứ i thực hiện công việc thứ j, $\forall i = 1, \ldots, m$, $\forall j = 1, \ldots, n$. Tìm 1 phương pháp phân công sao cho thời gian hoàn thành tất cả các công việc là thấp nhất.

3.2 Bài toán tô màu đồ thị – Graph coloring problem

Bài toán 5 (Bài toán tô màu các đỉnh đồ thị – Graph coloring problem)**.** *Có 1 đồ thị vô hướng đơn giản. Ta muốn tìm cách tô màu cho các đỉnh của đồ thị sao cho 2 đỉnh cạnh nhau phải có màu khác nhau. Yêu cầu: Tìm phương án tô sao cho số màu sử dụng là ít nhất.*

Input. *Đồ thị vô hướng đơn giản.*

Output. *Mỗi đỉnh tô màu gì.*

1 thuật giải heuristic. Sử dụng nguyên lý thứ tự:

```
for (i = 0; i < n; i++) {
    chọn đỉnh s chưa tô có d[s] lớn nhất;
    chọn màu: ưu tiên tô đỉnh s bằng 1 trong các màu đã sử dụng, nếu không được thì sử dụng màu mới;
    sau khi tô màu cho đỉnh s: với mỗi đỉnh x cạnh, giảm d[x]; ???
}
```

$d[x]$: số đỉnh cạnh x mà chưa tô màu. ???

3.3 Shortest path problem – Bài toán đường đi ngắn nhất

Bài toán 6. *Cài đặt & thử nghiệm* A**. So sánh với Dijkstra nếu được.*

Input. $G = (V, E)$ *có trọng số dương, đỉnh xuất phát* a*, đỉnh mục tiêu* z*. Thông tin bổ sung:* $h(x)$*: ước lượng khoảng cách từ* a *đến mục tiêu* z*.*

Output. *Đường đi ngắn nhất shortest path SP từ* a *đến* z*.*

3.4 Traveling salesman problem (TSP) – Bài toán người bán hàng du lịch

Problem 1 (Traveling Salesman Problem (TSP))**.** *The traveling salesman must visit every city in this territory exactly once & then return to the starting point; given the cost of travel between all cities, how should he plan his itinerary for minimum total cost of the entire tour?*

TSP $\in$ NP-Complete.

Remark 2 (Approximate TSP by GAs)**.** *We shall discuss a single possible approach to approximate the TSP by genetic algorithms (GAs).*

4 Miscellaneous

Tài liệu

[Tiệ25] Vũ Khắc Tiệp. *Machine Learning Cơ Bản*. 2025, p. 422.