



GD, SGD, Minibatch SGD

MATHS FOUNDATION FOR COMPUTER SCIENCE (C05097)
Nhóm 02

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

Nội dung

01

Gradient Descent (GD)

02

Stochastic GD (SGD)

03

Mini-Batch

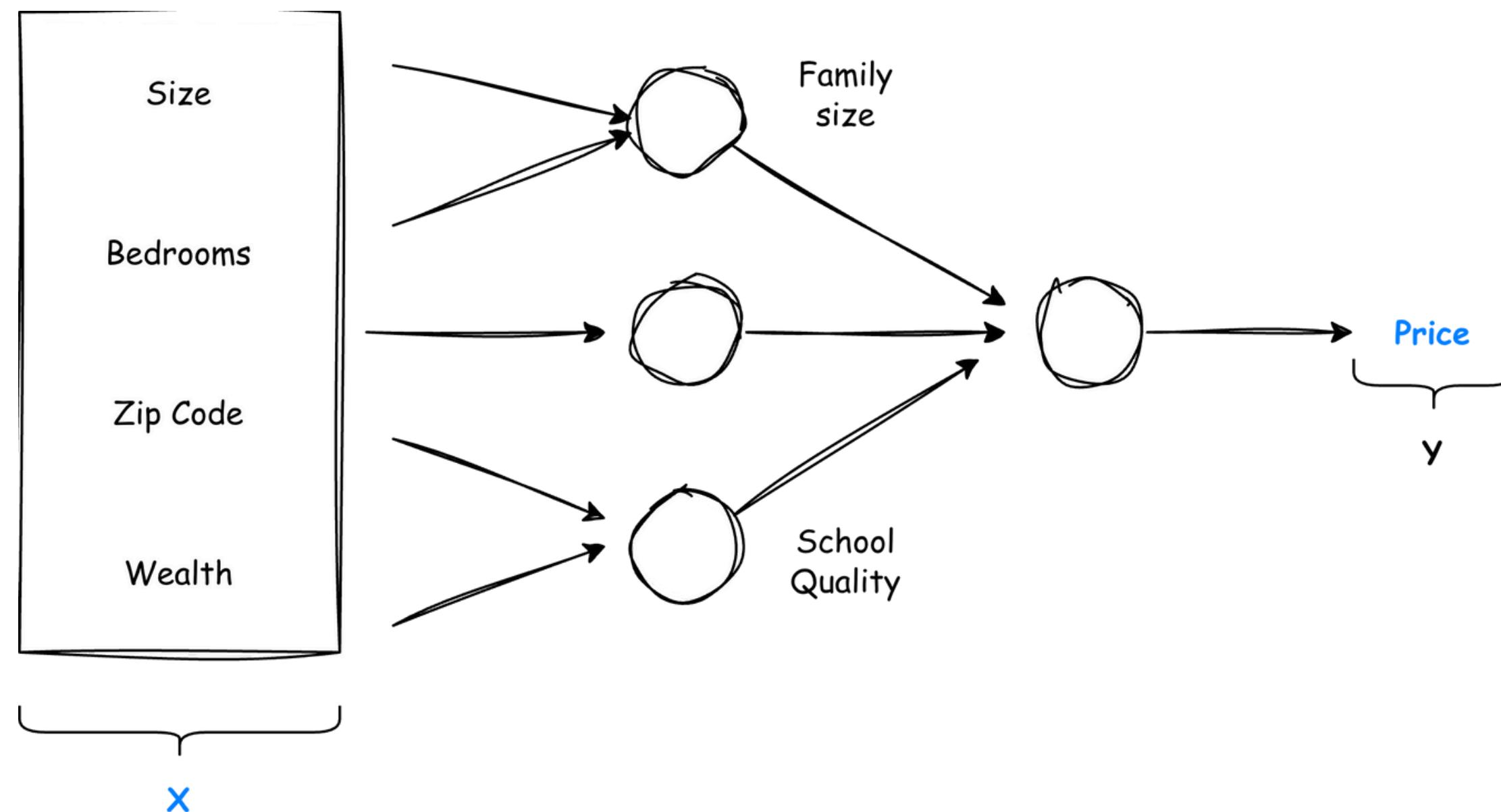
04

So sánh các thuật toán & Kết luận

1. Đặt vấn đề

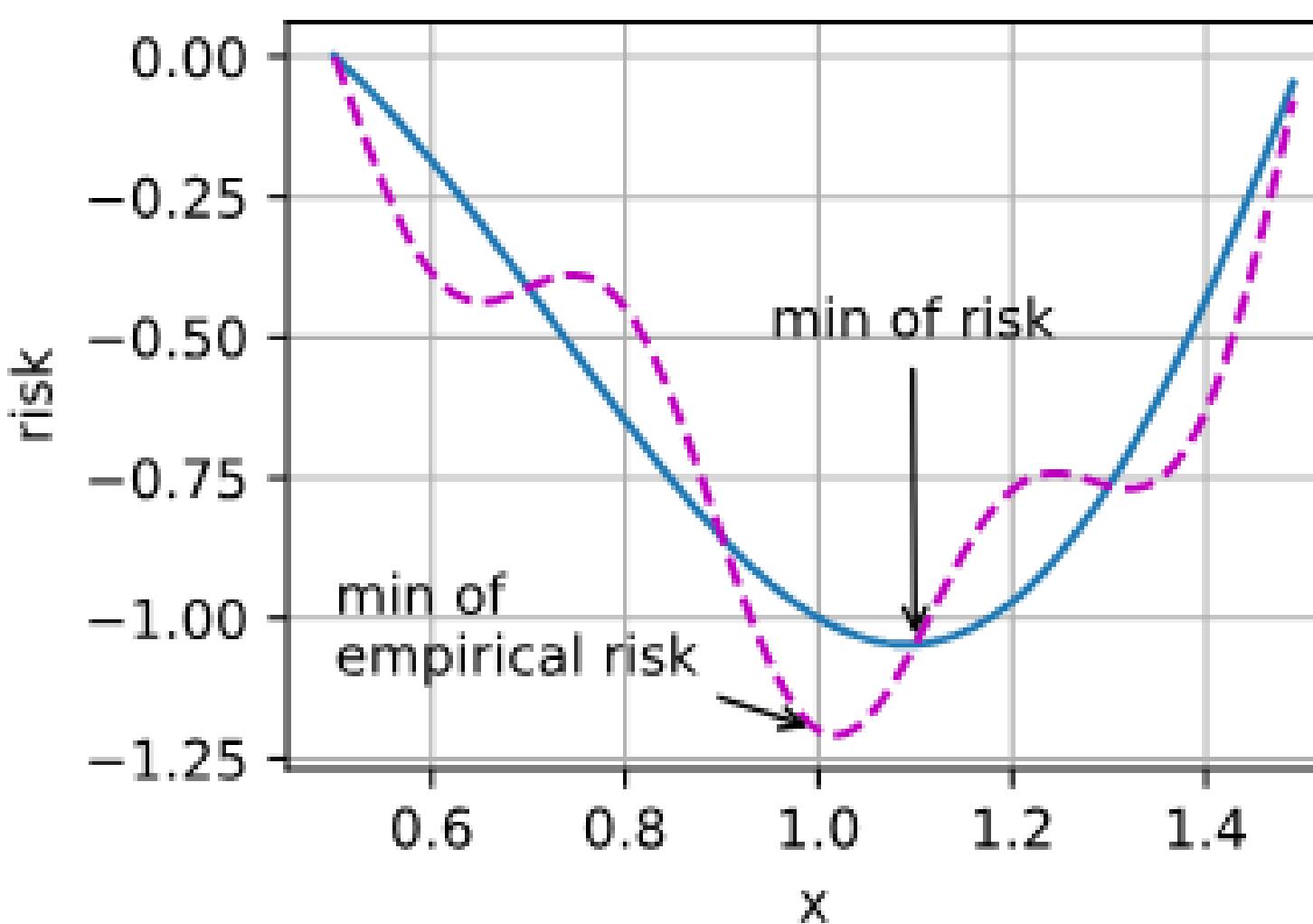
Bài toán tối ưu hóa là gì?

Tối ưu hóa là việc đi tìm phương án "tốt nhất" trong vô vàn các phương án có thể, dựa trên một tiêu chí cụ thể. Trong toán học, bài toán này thường được phát biểu là tìm giá trị nhỏ nhất (**Minimization**) hoặc lớn nhất (**Maximization**) của một hàm số.



Bài toán tối ưu hoá là gì?

Hàm mục tiêu $f(x)$ là thước đo để đánh giá mô hình "tốt" hay "xấu".



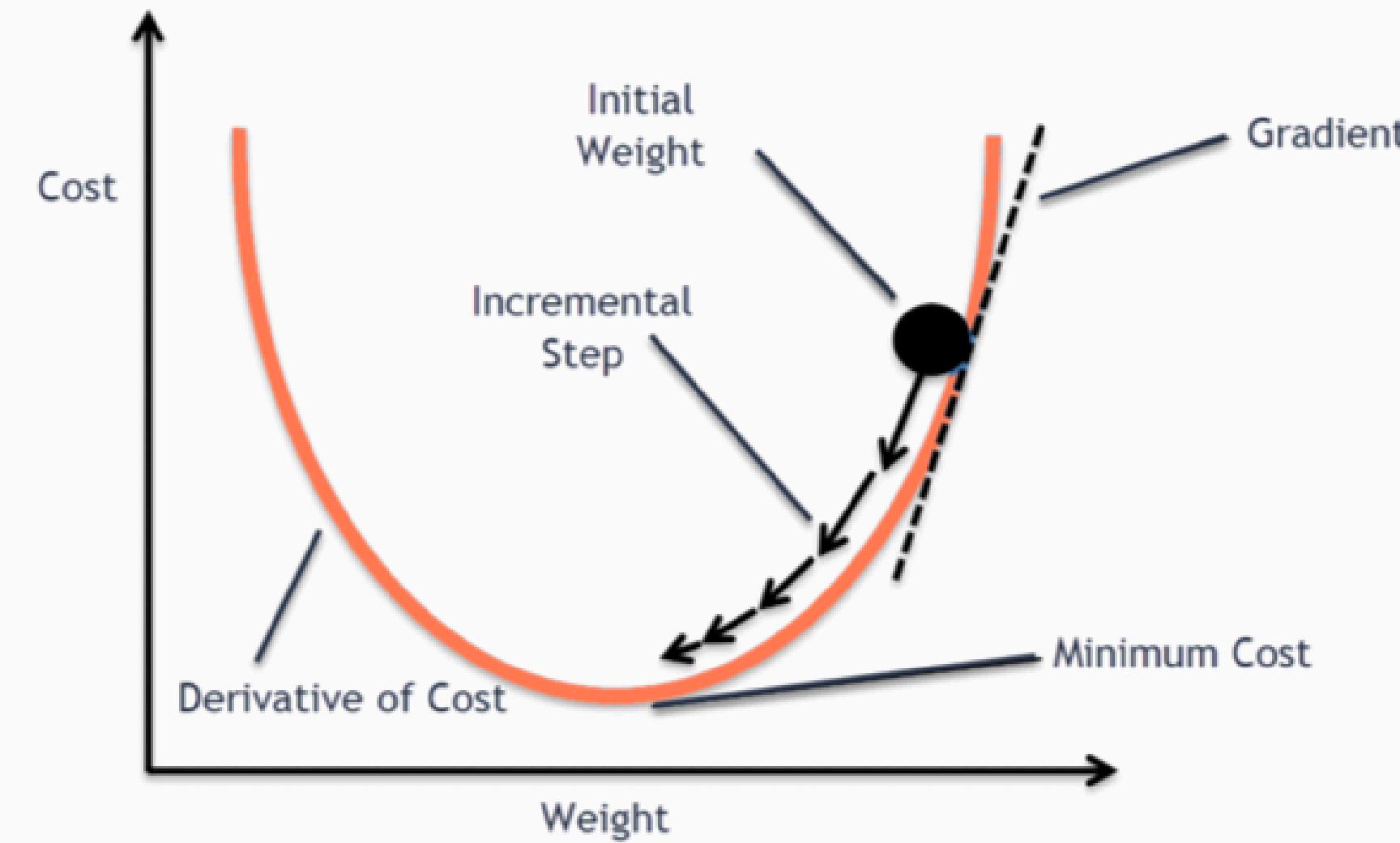
$$\hat{y} = f(x; W, b)$$

$$J(W, b) = \frac{1}{2m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

2. Gradient Descent (GD)

Gradient Descent là gì?

Gradient Descent là một thuật toán tối ưu hóa lặp (iterative) bậc nhất. Nó giải quyết bài toán tối ưu bằng cách liên tục di chuyển các tham số x theo hướng ngược lại với gradient (độ dốc) của hàm mục tiêu tại điểm hiện tại.



GD - Cơ sở lý thuyết

Cơ sở lý thuyết của GD là **Khai triển Taylor (Taylor Series Expansion)**, cho phép chúng ta xấp xỉ một hàm phức tạp bằng một hàm tuyến tính đơn giản tại một điểm cụ thể.

$$f(x) = f(a) + \frac{f'(a)}{1!}(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \frac{f'''(a)}{3!}(x - a)^3 + \dots$$

GD - Cơ sở lý thuyết

Giả sử $f : \mathbb{R}^n \rightarrow \mathbb{R}$ đủ khả vi tại điểm $x = (x_1, x_2, \dots, x_n)^\top$

Vector Gradient ($\nabla f(x)$): Gradient là một vector chứa tất cả các đạo hàm riêng này.

$$\nabla f(x) = \left[\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \dots, \frac{\partial f(x)}{\partial x_d} \right]^T$$

Lúc này khai triển Taylor bậc hai của hàm nhiều biến sẽ là:

$$f(x + \varepsilon) = f(x) + \nabla f(x) \varepsilon^\top + \mathcal{O}(\|\varepsilon\|^2).$$

Xấp xỉ Taylor bậc một cho hàm nhiều biến trở thành:

$$f(x + \epsilon) \approx f(x) + \epsilon^\top \nabla f(x)$$

GD - Cơ sở lý thuyết

Giả sử $f : \mathbb{R}^n \rightarrow \mathbb{R}$ đủ khả vi tại điểm $x = (x_1, x_2, \dots, x_n)^\top$

Ta chọn một bước di chuyển nhỏ ϵ trong đó $\eta > 0$ là hằng số dương nhỏ gọi là tốc độ học (learning rate) với:

$$\epsilon = -\eta \nabla f(x)$$

Khi thay ϵ vào công thức Taylor, ta có:

$$f(x - \eta \nabla f(x)) \approx f(x) + (-\eta \nabla f(x))^T \nabla f(x)$$

$$f(x - \eta \nabla f(x)) \approx f(x) - \eta (\nabla f(x))^T \nabla f(x)$$

$$f(x - \eta \nabla f(x)) \approx f(x) - \eta \|\nabla f(x)\|^2$$

$$x \leftarrow x - \eta \nabla f(x)$$

GD - Chi phí tính toán

Độ phức tạp thời gian của Gradient Descent phụ thuộc vào một số yếu tố: số lượng tham số (n), số lượng dữ liệu (m), và số vòng lặp (iterations)

$$O(k \times m \times n)$$

Trong đó:

- k là số lượng vòng lặp
- m là số lượng mẫu
- n là số lượng tham số

GD - Ví dụ

Ví dụ: $f(x) = 0.5x^2 + 2x + 5$

- Đạo hàm (Gradient): $f'(x) = x + 2$
- Cực tiểu toàn cục: $f'(x) = 0 \Rightarrow x + 2 = 0 \Rightarrow x = -2$
- Công thức cập nhật: $x_{t+1} = x_t - \eta f'(x_t) = x_t - \eta(x_t + 2)$

Chọn điểm bắt đầu $x_0 = 8.0$ và tốc độ học $\eta = 0.5$.

- **Bước 0 ($t=0$):** $x_0 = 8.0$
- **Bước 1 ($t=1$):**
 - Tính gradient: $f'(x_0) = f'(8.0) = 8.0 + 2 = 10.0$
 - Cập nhật x: $x_1 = x_0 - \eta f'(x_0) = 8.0 - 0.5 \times (10.0) = 8.0 - 5.0 = 3.0$

GD - Ví dụ

Ví dụ: $f(x) = 0.5x^2 + 2x + 5$

- **Bước 2 (t=2):**

- Tính gradient: $f'(x_1) = f'(3.0) = 3.0 + 2 = 5.0$
- Cập nhật x: $x_2 = x_1 - \eta f'(x_1) = 3.0 - 0.5 \times (5.0) = 3.0 - 2.5 = 0.5$

- **Bước 3 (t=3):**

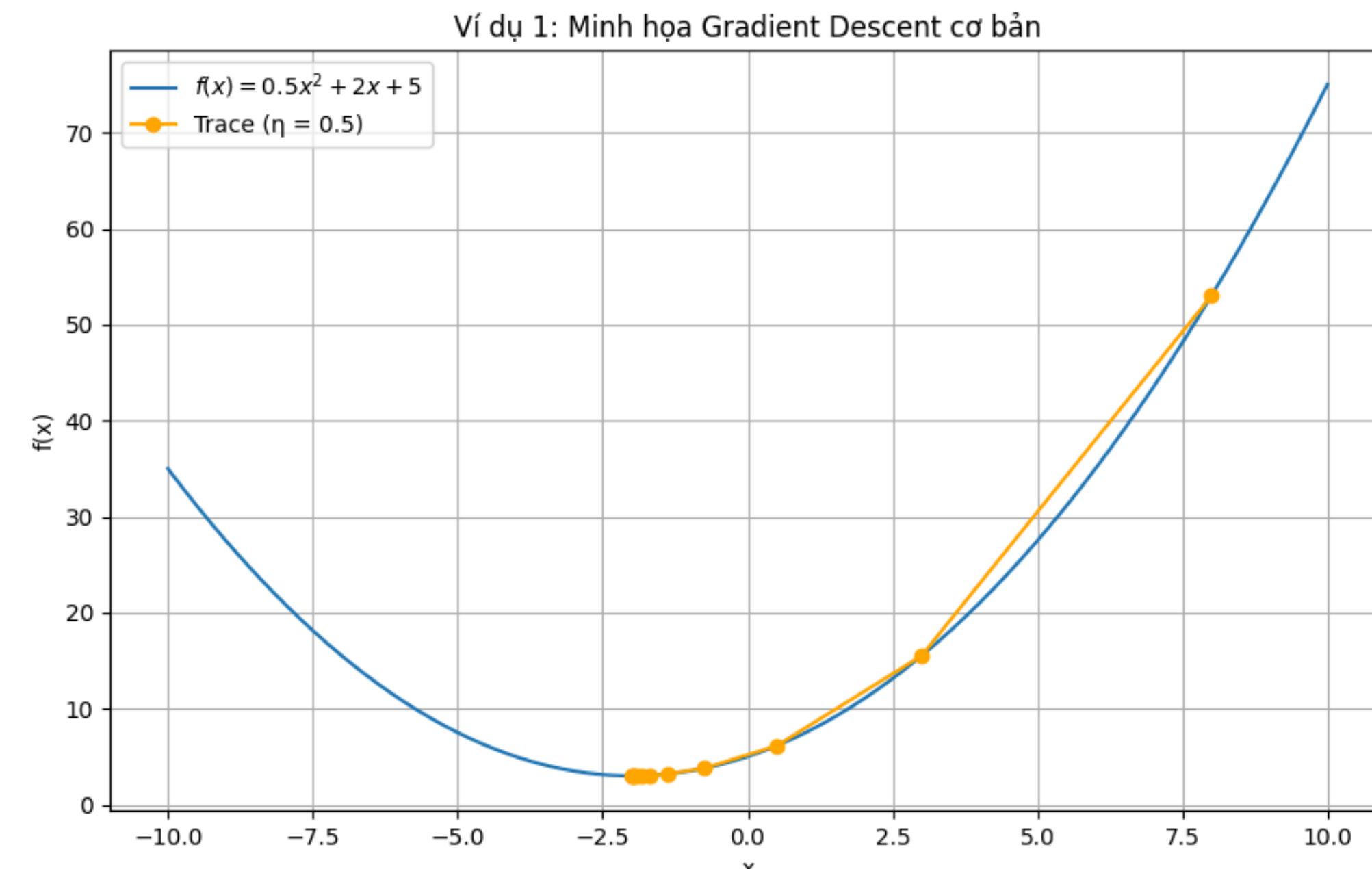
- Tính gradient: $f'(x_2) = f'(0.5) = 0.5 + 2 = 2.5$
- Cập nhật x: $x_3 = x_2 - \eta f'(x_2) = 0.5 - 0.5 \times (2.5) = 0.5 - 1.25 = -0.75$

....

- **Bước 10: $x_{10} \approx -1.992$**

GD - Ví dụ

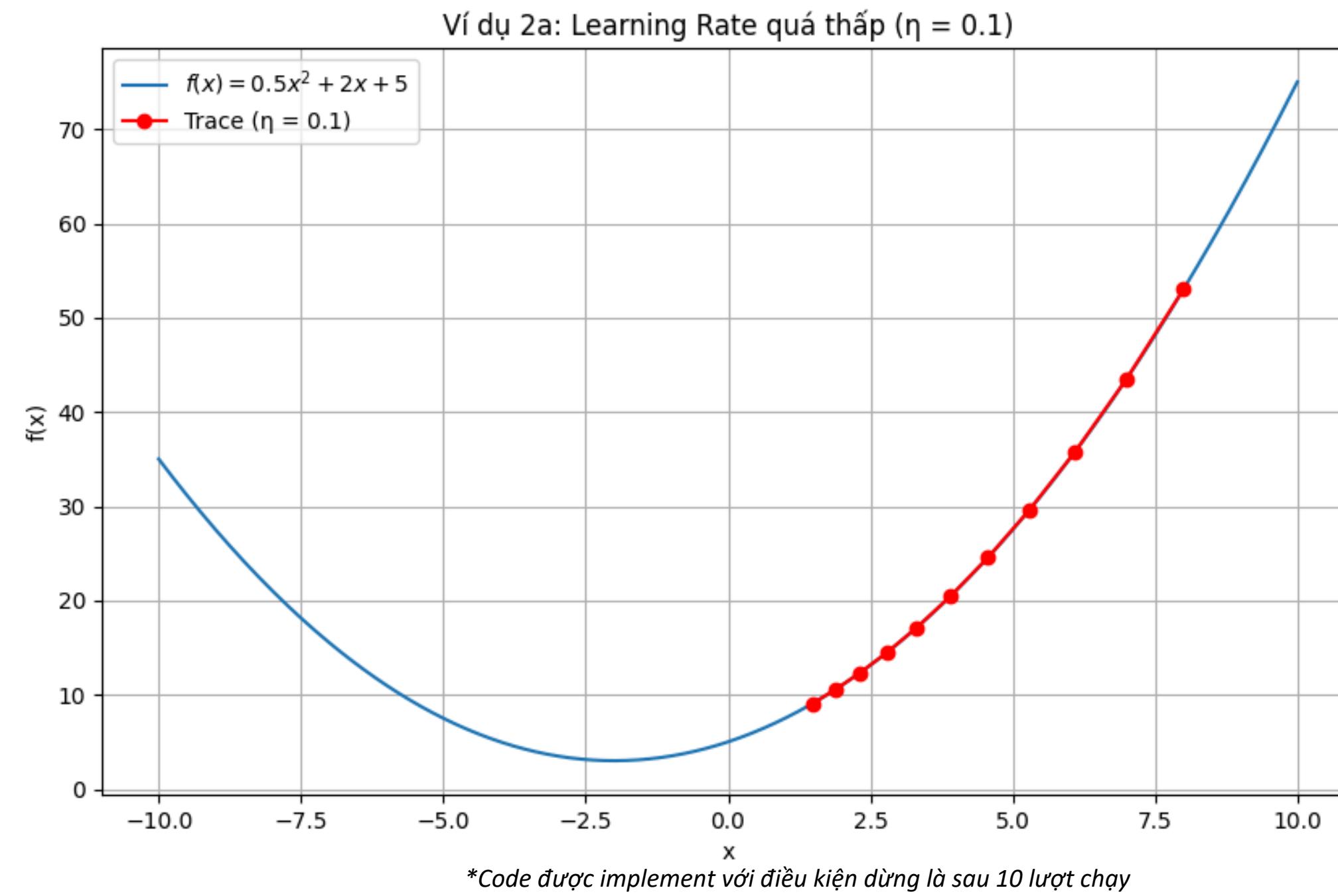
Ví dụ: $f(x) = 0.5x^2 + 2x + 5$



*Code được implement với điều kiện dừng là sau 10 lượt chạy

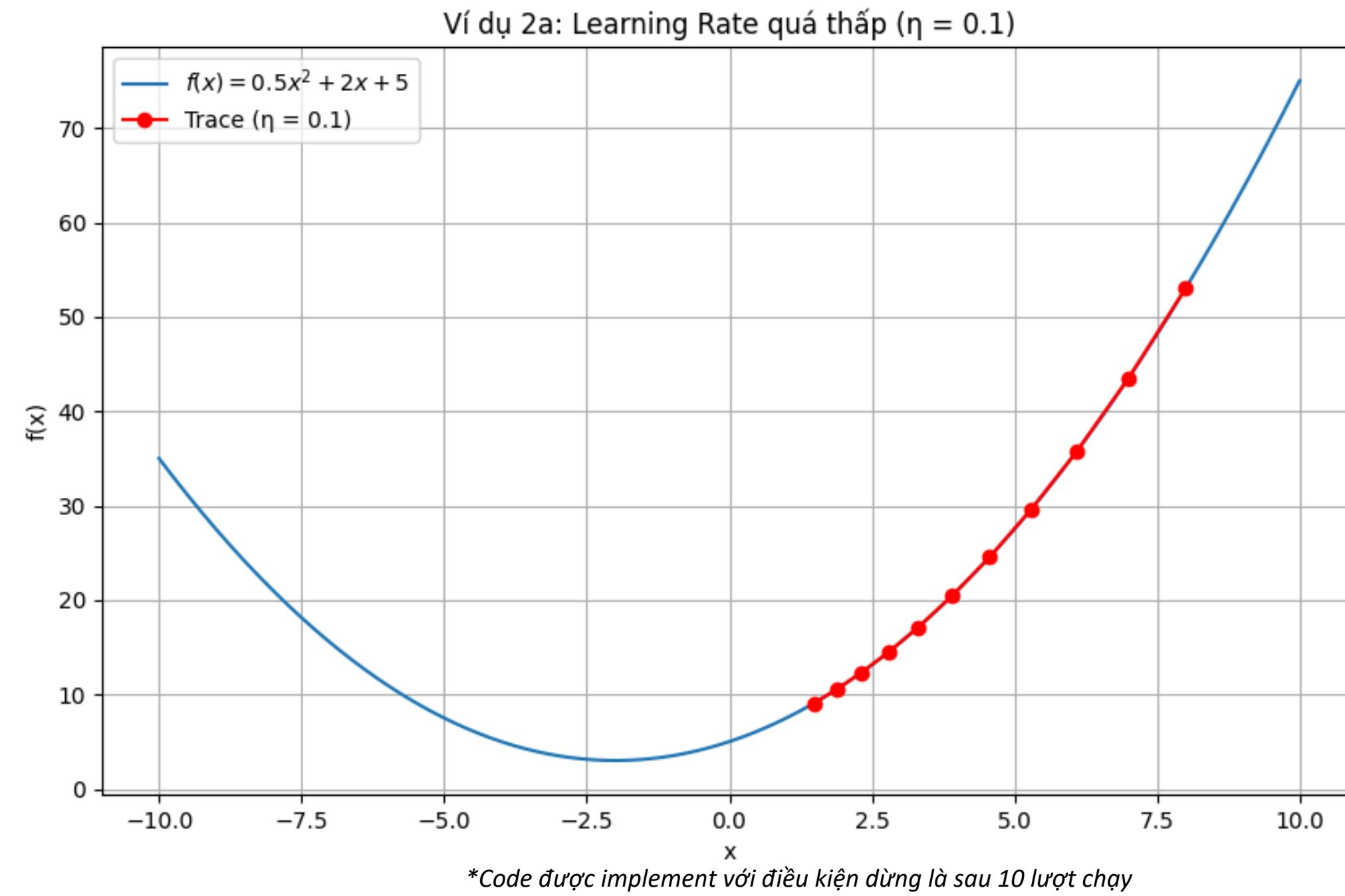
GD - Ví dụ

Ví dụ: $f(x) = 0.5x^2 + 2x + 5$. Chọn điểm bắt đầu $x_0 = 8.0$ và tốc độ học $\eta = 0.1$



GD - Ví dụ

Ví dụ: $f(x) = 0.5x^2 + 2x + 5$. Chọn điểm bắt đầu $x_0 = 8.0$ và tốc độ học $\eta = 0.1$



3. Stochastic GD (SGD)

SGD - Vấn đề của GD

Trong Gradient Descent, gradient được tính theo công thức:

$$\nabla f(x) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(x)$$

GD phải tính gradient trên toàn bộ n mẫu ở mỗi bước lặp dẫn tới chi phí tính toán $O(k \times m \times n)$. Với tập dữ liệu lớn, GD trở nên tốn kém.

SGD - Cơ sở lý thuyết

Stochastic Gradient Descent là một biến thể của Gradient Descent. Thay vì tính toán trên toàn bộ tập dữ liệu, ở mỗi lần lặp, ta chỉ xét một mẫu i ngẫu nhiên theo phân phối đều, và tính gradient để cập nhật x .

Chi phí tính toán: $O(k \times 1 \times n)$

SGD - Cơ sở lý thuyết

Khi lấy ngẫu nhiên mẫu i theo phân phối đồng đều trên tập dữ liệu n , ta có i là biến ngẫu nhiên mà $\nabla f_i(x)$ lại phụ thuộc vào i dẫn đến $\nabla f_i(x)$ cũng là biến ngẫu nhiên.
Lại có $p(i) = 1/n$.

Từ 2 điều trên ta được:

$$\mathbb{E}_i[\nabla f_i(x)] = \sum_{i=1}^n \nabla f_i(x)p(i) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(x)$$

SGD - Hạn chế của SGD

Mỗi bước SGD chỉ dùng 1 mẫu để tính gradient dẫn đến việc tạo nhiễu.

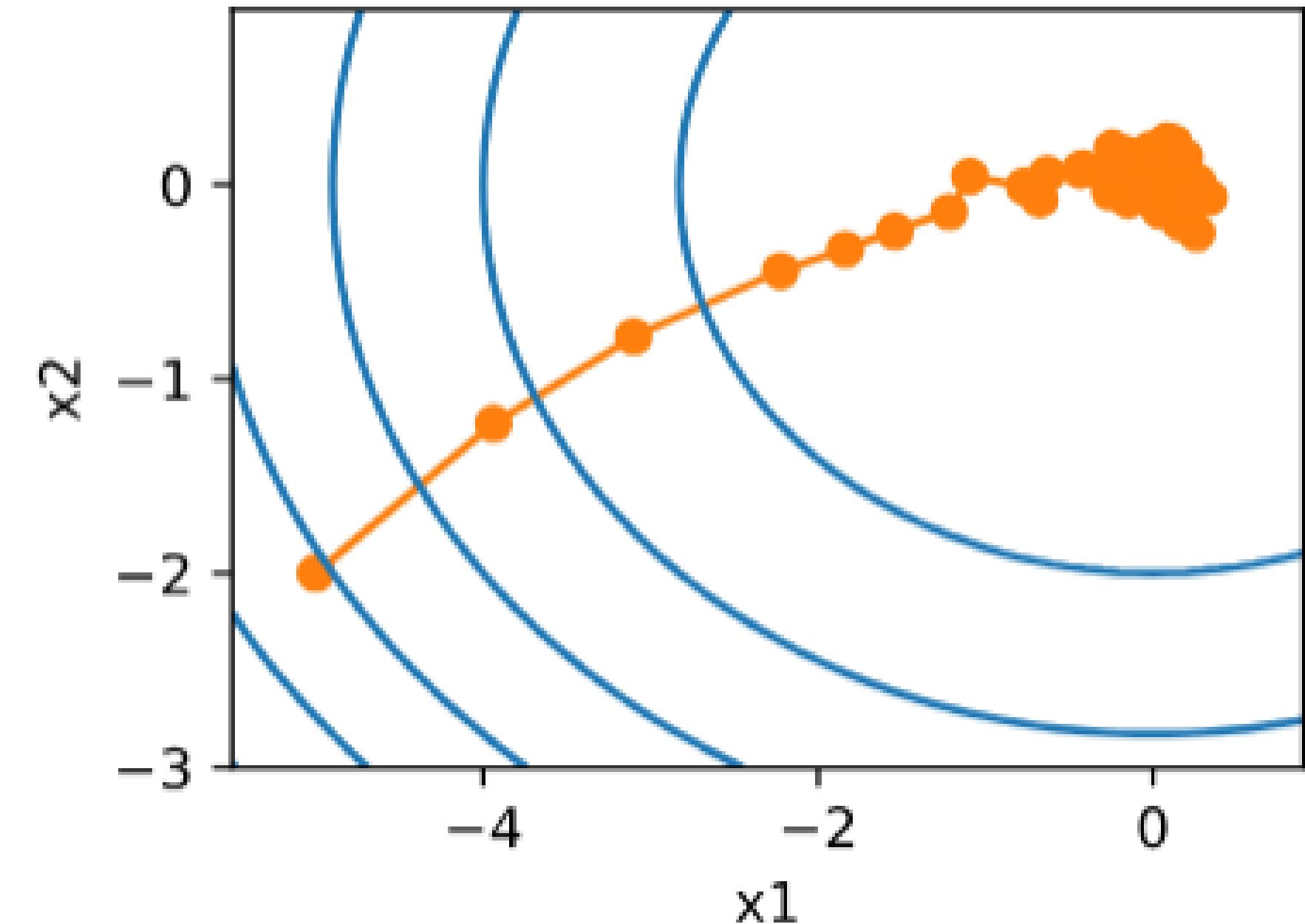
Nhiều này không chêch

$$\nabla f(x) = \nabla f_i(x) + \epsilon$$

$$\mathbb{E}_i[\epsilon] = \mathbb{E}_i[\nabla f_i(x)] - \nabla f(x) = 0.$$

→ SGD khớp với GD trong kỳ vọng, nhưng nhiễu khi xét từng bước.

SGD - Hạn chế của SGD



SGD - Tốc độ học động

Trong SGD, bước cập nhập là

$$-\eta \nabla f_i(x).$$

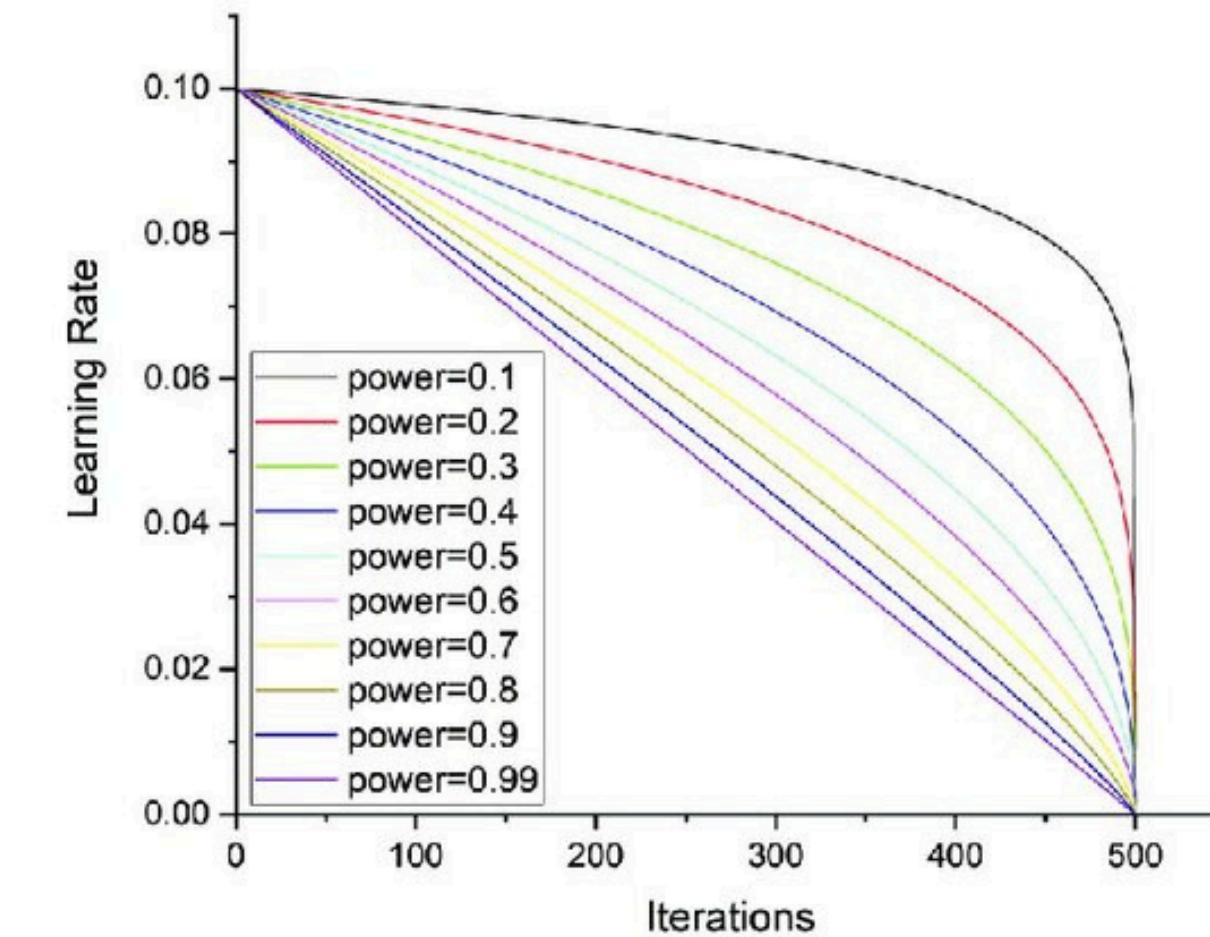
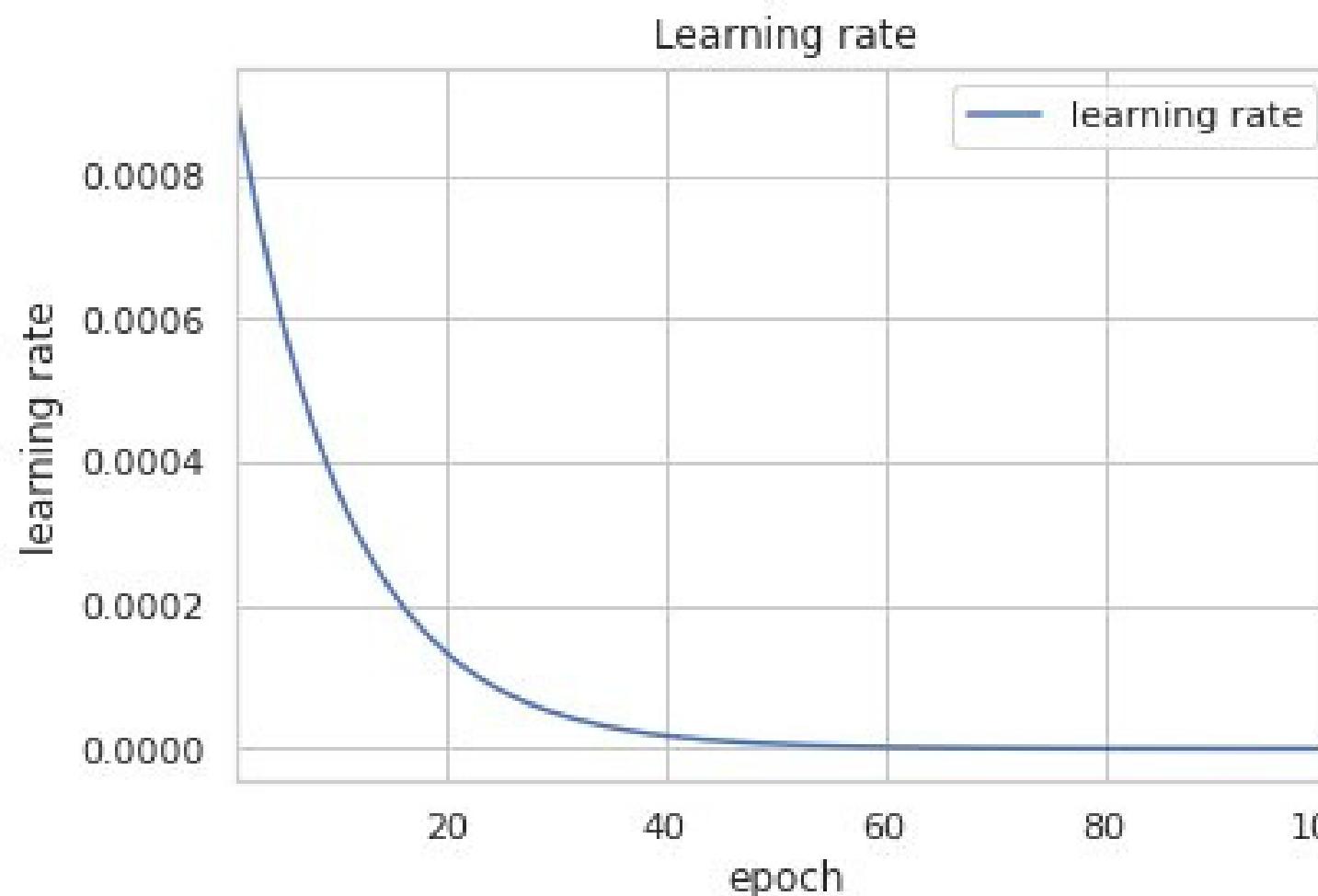
hiệu quả tối ưu phụ thuộc vào tốc độ học η . Nếu η quá nhỏ thì học chậm, η quá lớn thì dao động mạnh không có được kết quả tốt.

→ Giảm dần tốc độ học theo thời gian

SGD - Tốc độ học động

Các chiến thuật phổ biến

- Piecewise Constant: Ta giảm tốc độ học mỗi khi tiến trình tối ưu bị ngưng trệ.
- Exponential Decay: Vì tính chất giảm mạnh ngay từ ban đầu nên có thể dẫn tới trường hợp ngưng trệ sớm trước khi đạt tối ưu.
- Polynomial Decay: Giảm chậm hơn, ổn định hơn về cuối quá trình huấn luyện



SGD - Lấy mẫu trong môi trường hữu hạn

- Lý thuyết SGD: lấy mẫu ngẫu nhiên có hoàn lại từ phân phối nền.
- Thực tế: dữ liệu hữu hạn
 - duyệt ngẫu nhiên toàn bộ tập dữ liệu.
 - mỗi mẫu dùng đúng 1 lần / epoch.

SGD - Lấy mẫu hoàn lại và không hoàn lại

Giả sử lấy mẫu có hoàn lại từ tập n mẫu. Với xác suất chọn phần tử i ngẫu nhiên là $1/n$, để có thể chọn i ít nhất 1 lần:

$$P(\text{choose } i) = 1 - P(\text{omit } i) = 1 - \prod_{i=1}^n [1 - P(\text{choose } i \text{ once})] = 1 - [1 - \frac{1}{n}]^n$$

Lại có:

$$\lim_{n \rightarrow +\infty} \left(1 + \frac{x}{n}\right)^n = e^x$$

Khi ta có n tiến về vô cùng, ta sẽ được:

$$P(\text{choose } i) = \left(1 + \frac{(-1)}{n}\right)^n \approx 1 - e^{-1} \approx 0.63$$

SGD - Lấy mẫu hoàn lại và không hoàn lại

Ngoài ra, để có thể chọn i đúng 1 lần:

$$\begin{aligned} P(\text{choose } i \text{ exactly once}) &= \binom{n}{1} P(\text{choose } i \text{ once}) P(\text{omit } i \text{ } n-1) = \frac{n!}{1!(n-1)!} \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} \\ &= \frac{\left(1 - \frac{1}{n}\right)^n}{1 - \frac{1}{n}} = \frac{n}{n-1} \left(1 - \frac{1}{n}\right)^n \approx e^{-1} \\ &\approx 0.37 \end{aligned}$$

SGD - Lấy mẫu hoàn lại và không hoàn lại

Ta có thể thấy với lấy mẫu hoàn lại:

- Một số mẫu bị lặp lại nhiều lần.
- Một số mẫu không bao giờ được sử dụng.

→ Giảm hiệu quả sử dụng dữ liệu

Trong khi đó, với mẫu không hoàn lại:

- Đảm bảo mỗi mẫu chỉ được sử dụng 1 lần trong 1 epoch.

SGD - Ví dụ

Xét bài toán hồi quy tuyến tính đơn giản: $y = wx$, trong đó x là biến đầu vào, y là giá trị mục tiêu và w là tham số cần học.

Tập dữ liệu cung cấp: $\{(1,2), (2,4), (3,6), (4,8), (5,10)\}$

Tham số tối ưu cần tìm: $w^* = 2$

Siêu tham số: $\eta=0.1, w_0=0$

Hàm loss:

$$L(w) = \frac{1}{2n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \frac{1}{2n} \sum_{i=1}^n (y_i - wx_i)^2$$

SGD - Ví dụ

Đạo hàm hàm mất mát theo w ta được:

$$\frac{\partial L}{\partial w} = -\frac{1}{n} \sum_{i=1}^n x_i(y_i - wx_i)$$

Trong SGD, ta chọn một mẫu ngẫu nhiên để tính xấp xỉ cho gradient toàn phần.

$$\frac{\partial L}{\partial w} \approx -x_i(y_i - wx_i) = g(i)$$

SGD - Ví dụ

Bước 0($t = 0$): $w_0 = 0.0$.

Bước 1($t = 1$):

- Ta rút 1 phần tử trong tập dữ liệu (x, y) được $(1, 2)$.
- Tính $g(1) = -x_1(y_1 - w_0 * x_1) = -2$.
- Cập nhập $w_1 = w_0 - \eta g(1) = 0.2$.

Bước 2 ($t=2$):

- Ta rút 1 phần tử trong tập dữ liệu (x, y) được $(3, 6)$.
- Tính $g(3) = -x_3(y_3 - w_1 * x_3) = -16.2$.
- Cập nhập $w_2 = w_1 - \eta g(3) = 1.82$.

Bước 3 ($t=3$):

- Ta rút 1 phần tử trong tập dữ liệu (x, y) được $(4, 8)$.
- Tính $g(4) = -x_4(y_4 - w_2 * x_4) = -2.88$.
- Cập nhập $w_3 = w_2 - \eta g(4) = 2.108$.

SGD - Ví dụ

Bước 4 ($t=4$):

- Ta rút 1 phần tử trong tập dữ liệu (x,y) được $(2,4)$.
- Tính $g(2) = -x_2(y_2 - w_3x_2) = 0.432$
- Cập nhập $w_4 = w_3 - \eta g(2) = 2.0648$.

Bước 5 ($t=5$):

- Ta rút 1 phần tử trong tập dữ liệu (x,y) được $(5,10)$.
- Tính $g(5) = -x_5(y_5 - w_4x_5) = 1.62$
- Cập nhập $w_5 = w_4 - \eta g(5) = 1.9028$

Kết quả sau một epoch là:

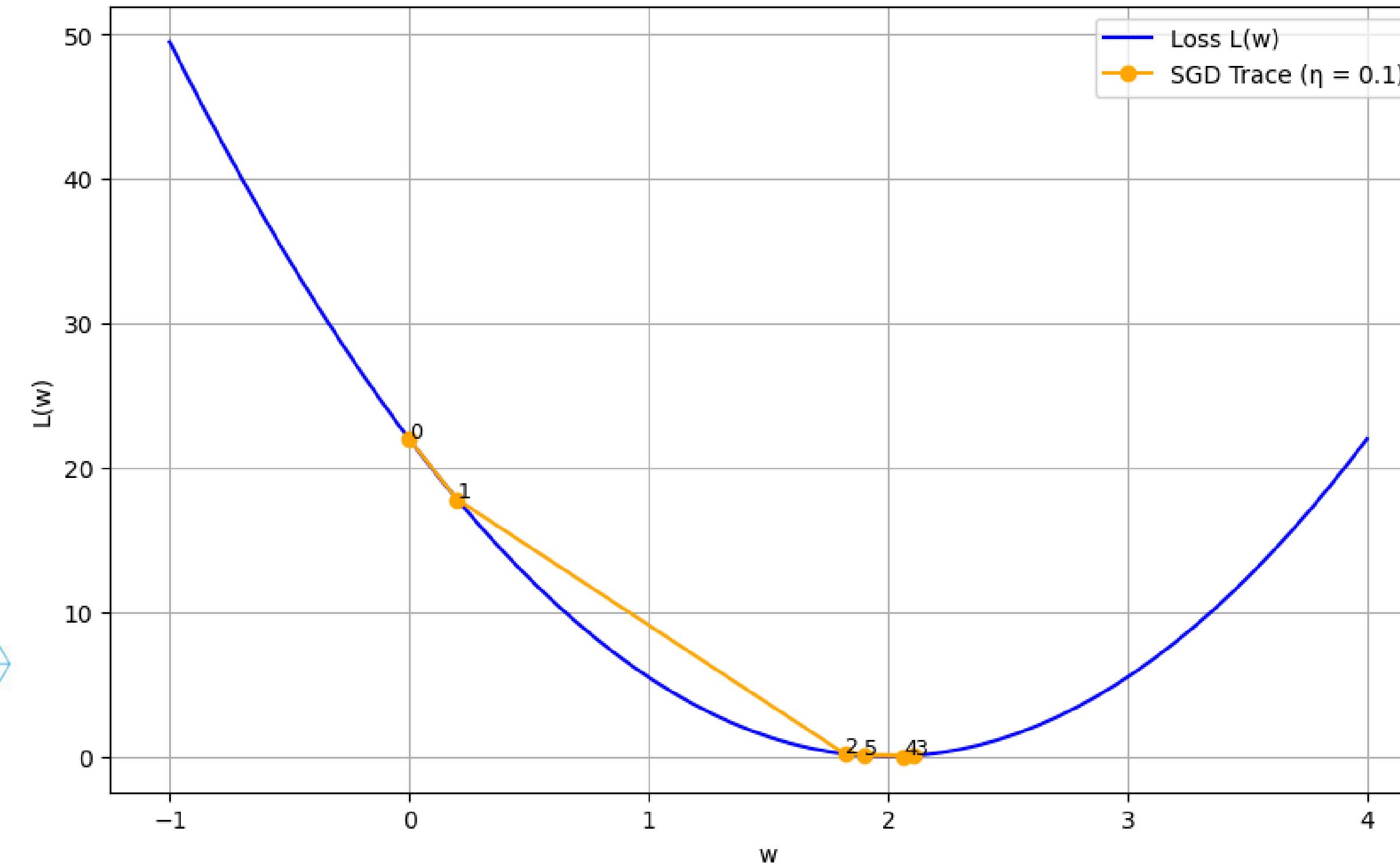
- $w = 1.9028$

→ SGD di chuyển không trơn tru như GD. Giá trị w dao động quanh $w^* = 2$ do nhiều.

Đây là đặc trưng của SGD khi dùng learning rate cố định.

SGD - Ví dụ

Minh họa SGD với lấy mẫu không thay thế (1 epoch)



4. Mini-Batch



Vấn đề của GD và SGD

- Vấn đề của các phương pháp cũ:
- Gradient Descent (GD): Dùng toàn bộ n mẫu.
 - Chi phí tính toán quá lớn, không khả thi với dữ liệu lớn.
- Stochastic Gradient Descent (SGD): Dùng 1 mẫu duy nhất.
 - Giá trị hàm Chi phí hỗn loạn (high variance), đường đi tối ưu hình zigzag, không ổn định.
- Mini Batch GD:
 - Sử dụng một nhóm nhỏ dữ liệu (Batch-size) để ước tính gradient.
 - Mục tiêu: Tạo sự cân bằng tối ưu giữa hiệu suất tính toán và độ ổn định hội tụ.

Tại sao MGD phù hợp với phần cứng hiện đại?

- Vector hóa (Vectorization): Tận dụng khả năng tính toán song song
 - Cơ chế: CPU/GPU hiện đại xử lý 8, 16, 32... lệnh đồng thời trong một chu kỳ.
 - Lợi ích:
 - Kích thước batch-size lớn vừa phải, tận dụng tốt sức mạnh tính toán.
 - Thay thế vòng lặp (for, while) bằng phép toán ma trận/vector.
 - Thực tế:
 - Các thư viện hỗ trợ tối ưu cực tốt cho phép toán vector.
 - Vector hóa là điều kiện cần thiết để tính toán trên các CPU/GPU hiện đại.

Tại sao MGD phù hợp với phần cứng hiện đại?

- Bộ nhớ đệm (Cache): Tối ưu hóa truy xuất dữ liệu
 - Tốc độ bộ nhớ: Cache (L1, L2 - nhanh, nhỏ) > RAM (chậm, lớn).
 - Nguyên lý hoạt động tối ưu: Dữ liệu cần nằm trong Cache càng lâu càng tốt để tránh "Cache miss" (phải truy xuất xuống RAM dẫn đến thắt cổ chai).
- Ưu điểm của MGD:
- Xử lý các khối dữ liệu liền kề (mini-batch) giúp tăng tỷ lệ "Cache hit".
 - Dữ liệu được xử lý gói gọn trong L1/L2 Cache (CPU) hoặc VRAM (GPU) giúp tăng tốc độ huấn luyện đáng kể.

MGD - Cơ sở lý thuyết

- Gradient trên Mini-batch: Thay vì tính trên 1 mẫu, ta tính trung bình trên lô B:

$$\mathbf{g}_t = \partial_{\mathbf{w}} \left(\frac{1}{|\mathcal{B}_t|} \sum_{i \in \mathcal{B}_t} f(\mathbf{x}_i, \mathbf{w}) \right)$$

trong đó: B là kích thước batch (batch-size).

- MGD giúp giảm chi phí tính toán do gọi hàm và tận dụng phép nhân ma trận/véc-tơ hiệu quả hơn so với xử lý từng mẫu hoặc toàn bộ mẫu lớn quá dẫn đến quá tải.

MGD - Chiến lược chọn Batch-size

Như vậy có phải Batch càng lớn càng tốt?

Chiến lược thực tế:

- Chọn mini-batch vừa đủ lớn để tận dụng tối đa vec-tor hóa của GPU/CPU
- Chọn mini-batch vừa đủ để dữ liệu xử lý vẫn nằm trong bộ nhớ (VRAM/Cache).
- Cân bằng giữa tốc độ tính toán mỗi bước và tốc độ hội tụ của thuật toán.

MGD - Ví dụ tính toán

Xét mô hình hồi quy tuyến tính $\hat{y} = wx$:

Giả sử mô hình tuyến tính với phương trình $y = 2x$. Thủ nghiệm với dữ liệu tính toán gồm 8 điểm, để dễ tính toán tay không xáo trộn các batch:

$$(x_i, y_i) = (1, 2); (2, 4); (3, 6); (4, 8); (5, 10); (6, 12); (7, 14); (8, 16)$$

MGD - Ví dụ tính toán

Ta có:

- Sai số từng điểm:

$$\hat{y} - y = wx - y = wx - 2x = (w - 2)x$$

- Hàm mất mát hồi quy tuyến tính:

$$\begin{aligned} L(w) &= \frac{1}{2n} \sum_{i=1}^{n=8} (wx_i - y_i)^2 = \frac{1}{2n} \sum_{i=1}^{n=8} (w - 2)^2 x_i^2 = \frac{(w - 2)^2}{2n} \sum_{i=1}^{n=8} x_i^2 \\ &= \frac{(w - 2)^2}{2 \cdot 8} (1^2 + 2^2 + \dots + 8^2) = \frac{(w - 2)^2}{16} \cdot 204 = 12.75 \cdot (w - 2)^2 \end{aligned}$$

- Gradient của hàm Loss:

$$\begin{aligned} g(w) &= \frac{\partial L}{\partial w} = \frac{\partial}{\partial w} \left[\frac{1}{2n} \sum_{i=1}^n (wx_i - y_i)^2 \right] = \frac{1}{2n} \sum_{i=1}^n \frac{\partial}{\partial w} [(wx_i - y_i)^2] \\ &= \frac{1}{2n} \sum_{i=1}^n \left[2(wx_i - y_i) \cdot \frac{\partial}{\partial w} (wx_i - y_i) \right] = \frac{2}{2n} \sum_{i=1}^n (wx_i - y_i) x_i = \frac{1}{n} \sum_{i=1}^n (wx_i - y_i) x_i \end{aligned}$$

MGD - Ví dụ tính toán

- Vậy Gradient trên Batch-size có kích thước $|B|$ là:

$$g_B(w) = \frac{1}{|B|} \sum_{i \in B} (wx_i - y_i)x_i = \frac{1}{|B|} \sum_{i \in B} (w - 2)x_i^2$$

- Cập nhật tham số:

$$w_{\text{mới}} = w_{\text{cũ}} - \eta g_B(w_{\text{cũ}})$$

- Giả sử các giá trị ban đầu:

$$\eta = 0.02; \quad w_0 = 0; \quad L_{\text{Full}}(w_0) = 12.75 \cdot (w_0 - 2)^2 = 12.75 \cdot (0 - 2)^2 = 12.75 \cdot 4 = 51$$

MGD - Ví dụ tính toán

Trường hợp 2: MDG với Batch-size = 2, tính tương tự ta có $w_0 = 0, L_{full}(w_0) = 51$ ta có:

$$B_1 = \{(1, 2), (2, 4)\}; B_2 = \{(3, 6), (4, 8)\}; B_3 = \{(5, 10), (6, 12)\}; B_4 = \{(7, 14), (8, 16)\}$$

$$g_B(w) = \frac{1}{|B|} \sum_{i \in B} (w - 2)x_i^2 \Rightarrow g_2(w) = \frac{1}{2} \sum_{i \in 2} (w - 2)x_i^2$$

- Tính Epoch 1:
- Tại $B_1 = \{(1, 2), (2, 4)\}, w_0 = 0$:

$$g_2(w_0) = \frac{1}{2} \sum_{i \in 2} (w_0 - 2)x_i^2 = \frac{1}{2} \cdot (0 - 2)(1^2 + 2^2) = -5$$

$$w_1 = w_0 - \eta \cdot g_2(w_0) = 0 - 0,02 \cdot (-5) = 0.1$$

MGD - Ví dụ tính toán

- Tại $B_2 = \{(3, 6), (4, 8)\}$, $w_1 = 0.1$:

$$g_2(w_1) = \frac{1}{2} \sum_{i \in B} (w_1 - 2)x_i^2 = \frac{1}{2} \cdot (0, 1 - 2)(3^2 + 4^2) = -23.75$$

$$w_2 = w_1 - \eta \cdot g_2(w_1) = 0.1 - 0.02 \cdot (-23.75) = 0.1 + 0.475 = 0.575$$

Tính tương tự cho B_3 và B_4 ta được kết quả:

Batch	$g(w)$	w
1	-5	0.1
2	-23.75	0.575
3	-43.4625	1.44425
4	-31.399875	2.0722475

Vậy tham số w cuối Epoch 1 của Batch-size = 2: $w_{E1} \approx 2.0722$ và $L_{\text{Full}}(w_{E1}) = 12.75 \cdot (w_{E1} - 2)^2 = 12.75 \cdot (2.0772 - 2)^2 \approx 0.06655$

5. So sánh các thuật toán & Kết luận

So sánh đặc điểm hội tụ, chi phí tính toán và tính phù hợp trong huấn luyện mô hình học sâu.

Tổng quan so sánh

So sánh trong trường hợp lấy mẫu hoàn lại

- Gradient Descent (GD):
 - Cập nhật bằng toàn bộ dữ liệu → hội tụ mượt và ổn định, nhưng chi phí tính toán cao, tốc độ chậm khi dữ liệu lớn.
- Stochastic GD (SGD):
 - Cập nhật sau từng mẫu → bước đi rất nhanh, nhưng gradient nhiễu mạnh, đường hội tụ dễ zig-zag và khó ổn định.
- Mini-Batch SGD:
 - Sử dụng một lô nhỏ dữ liệu mỗi bước → giảm nhiễu so với SGD, nhẹ hơn GD về chi phí, đạt cân bằng tối ưu giữa tốc độ và ổn định.

Chi tiết so sánh

	Tốc độ hội tụ	Chi phí tính toán	Ổn định & nhiễu
Gradient Descent (GD)	hội tụ ổn định nhưng chậm	$O(n)$ mỗi bước	không nhiễu
Stochastic GD (SGD)	nhiều dao động, khó ổn định	$O(1)$ mỗi bước	nhiễu mạnh → zig-zag
Mini-Batch SGD	hội tụ nhanh và mượt hơn	$O(b)$ nhưng tối ưu GPU/CPU	nhiễu giảm đáng kể

Kết luận từ phần so sánh

- Gradient Descent (GD): ổn định nhưng chậm, ít dùng trong DL.
 - SGD: cập nhật nhanh nhưng nhiễu mạnh, đường hội tụ dao động.
 - Mini-Batch SGD: dùng lô dữ liệu nhỏ, giảm nhiễu, tận dụng GPU.
- Mini-Batch SGD là tiêu chuẩn trong deep learning.
→ Cân bằng tối ưu giữa tốc độ, ổn định và hiệu quả tính toán.
→ Được sử dụng mặc định trong PyTorch, TensorFlow, JAX.

For-loop & Vectorization

Mục tiêu: So sánh tốc độ nhân Xw bằng for-loop và vectorization:

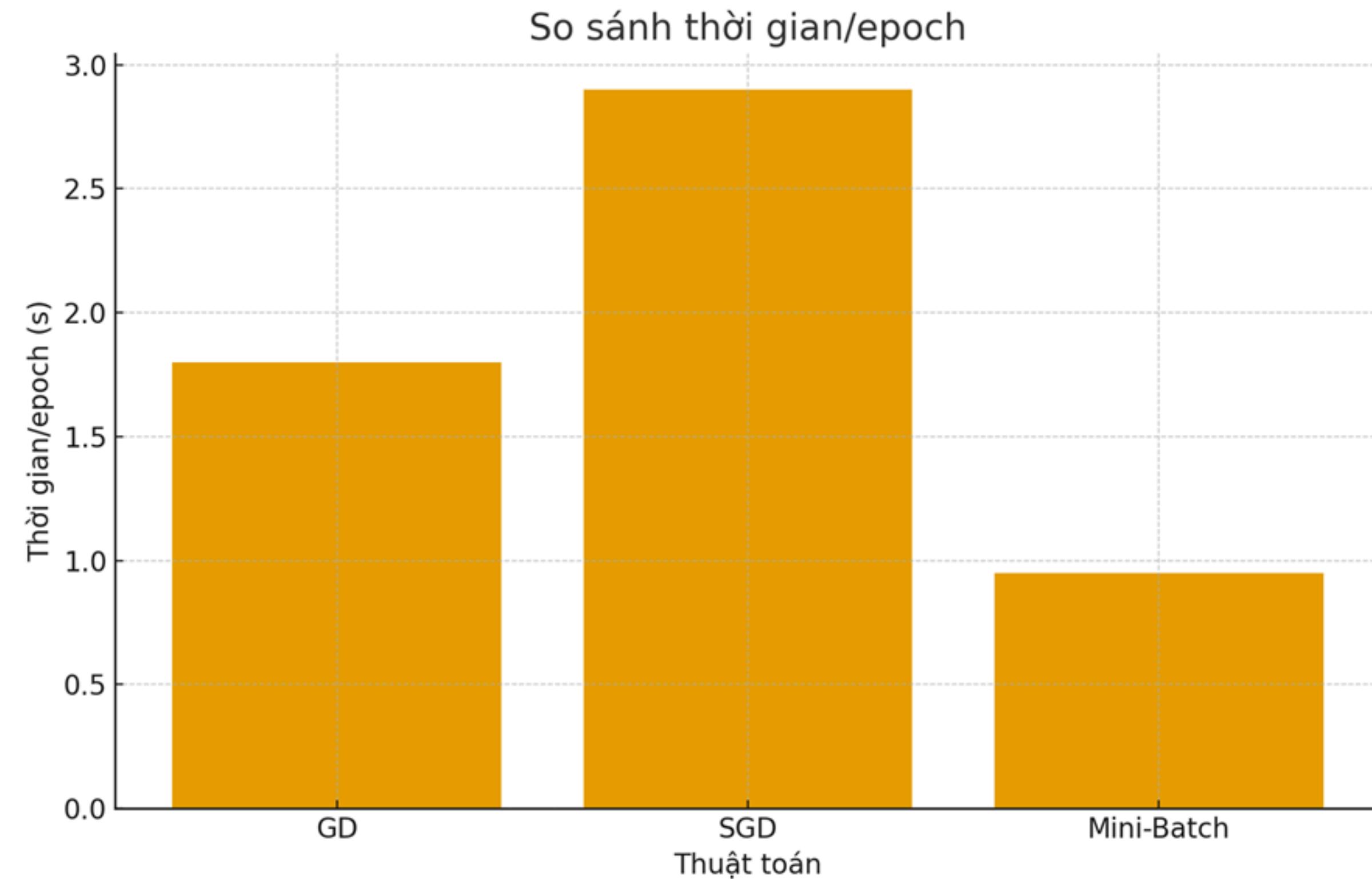
- Vectorization nhanh hơn ~ 88 lần.
- Kết quả thực tế trên CPU:
 - For-loop: ~ 3.47 s
 - Vectorized: ~ 0.039 s
 - Speedup: $\sim 88\times$

Kết luận: Phải dùng vectorization để đạt hiệu năng.

Table 1: Benchmark for-loop vs vectorization trên CPU (Colab)

Thiết lập	For-loop (s)	Vectorized (s)	Speedup
CPU	3.4728	0.0394	88.2×

Benchmark Vectorization & Mini-Batch





Thank you

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA