# Gradient giảm (Gradient Descent)

(Tài liệu nội bộ)



Tháng 3 năm 2020

# Nội dung trình bày

1. Nhắc lại về giải tích nhiều biến
2. Thuật toán GD
3. Thực hành với python
4. Gradient descent ngẫu nhiên
5. Mini-Batch GD

# Nội dung trình bày

1. Nhắc lại về giải tích nhiều biến

# Đạo hàm riêng và Véc tơ gradient

- **Định nghĩa:**
  Hàm số n biến số có tập xác định $D \subset \mathbb{R}^n$ là ánh xạ

$$\begin{aligned} f &: D \to \mathbb{R} \\ \mathbf{x} &\mapsto f(\mathbf{x}), \mathbf{x} = (x_1, x_2, ..., x_n) \end{aligned} \tag{1}$$

  Để cho một hàm n biến, người ta thường viết đơn giản là $y = f(\mathbf{x})$, $x = (x_1, x_2, ..., x_n)$ với miền xác định $D$ ngầm hiểu là miền trong $\mathbb{R}^n$ sao cho biểu thức $f$ có nghĩa.
- **Đạo hàm riêng** Đạo hàm của $f(\mathbf{x})$ theo biến $x_i$, các biến khác xem như hằng số, được gọi là đạo hàm riêng của $f$ theo biến $x_i$, và được ký hiệu là $f'_{x_i}(\mathbf{x})$, hay $\frac{\partial}{\partial x_i} f(x)$

# Đạo hàm riêng và Véc tơ gradient

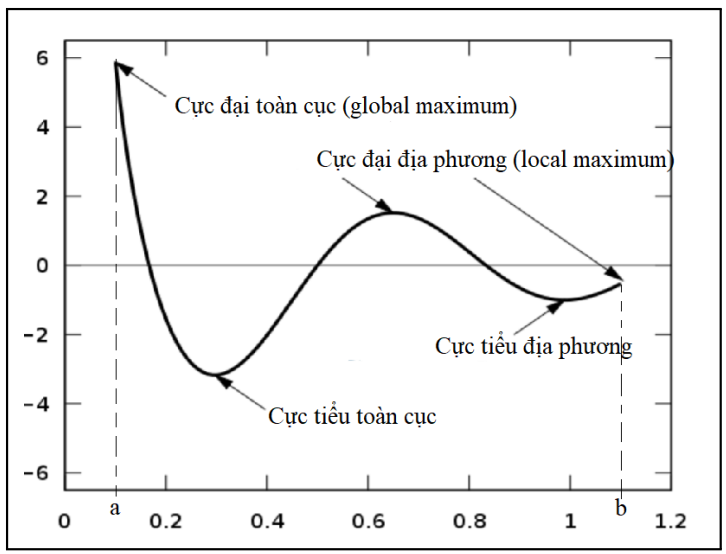
- **Biểu thức vi phân** của hàm hai biến $f(x_1, x_2)$ (để đơn giản):
  Vi phân cấp 1: $df(x_1, x_2) = f'_{x_1} dx_1 + f'_{x_2} dx_2$
  Vi phân cấp 2: $d^2 f(x_1, x_2) = f''_{x_1^2} {dx_1}^2 + 2f''_{x_1 x_2} dx_1 dx_2 + f''_{x_2^2} {dx_2}^2$
- **Véc tơ Gradient** của hàm $n$ biến $f(\mathbf{x})$ được định nghĩa là

$$\nabla f = \left( f'_{x_1}, f'_{x_2}, \ldots, f'_{x_n} \right) \tag{2}$$

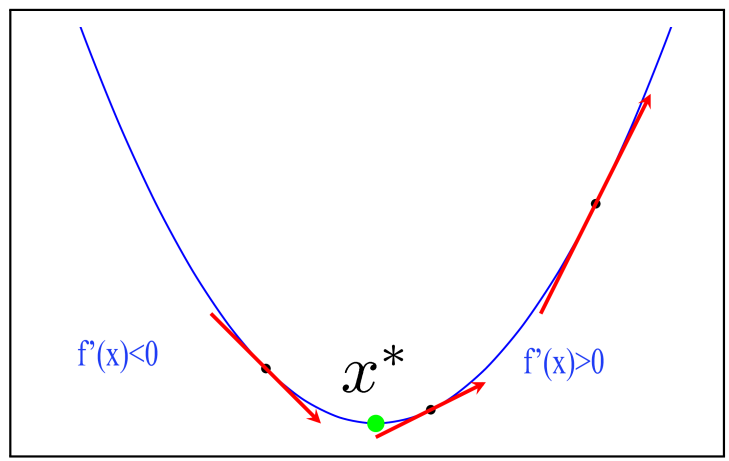
# Bài toán cực trị

Xét bài toán tìm cực trị tự do (cực đại hoặc cực tiểu địa phương) của hàm số $f(\mathbf{x})$, thuật toán là:

- Tìm tập xác định D
- Tính đạo hàm và giải phương trình đạo hàm bằng 0 để tìm các điểm dừng
- Tại mỗi điểm dừng $M$, xét dấu của vi phân cấp 2 $d^2(f(M))$, nếu $> 0$ ($< 0$) thì $M$ là điểm cực tiểu (cực đại)




Hình 1: Cực trị của hàm một biến




Hình 2: Dấu của đạo hàm quanh điểm cực tiểu $x^*$

# Nội dung trình bày

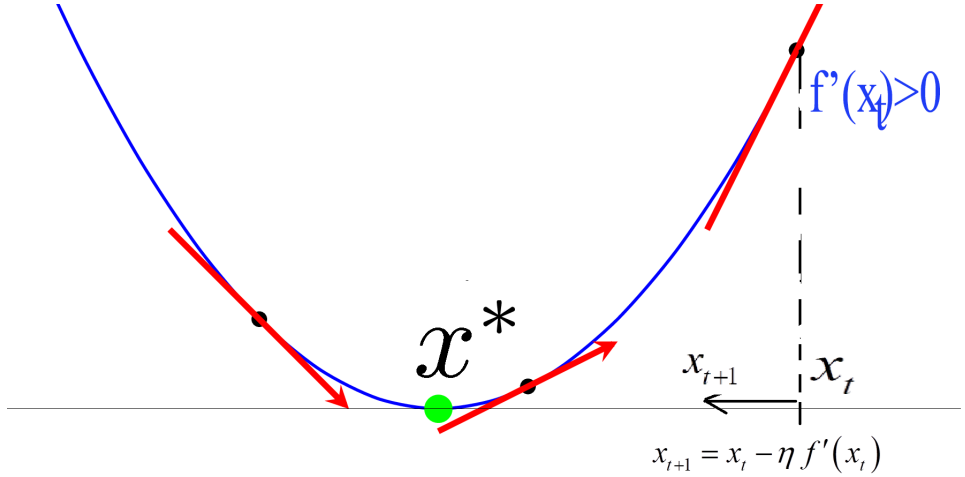
❷ Thuật toán GD

Trong Machine Learning, ta thường xuyên phải tìm giá trị nhỏ nhất của một hàm số nào đó (như cực tiểu hàm mất mát trong hồi quy tuyến tính). Bài toán này thường khá phức tạp. Tính phức tạp có thể đến từ một số lý do sau:

- Biểu thức hàm số phức tạp nên khó giải phương trình đạo hàm bằng 0
- Số chiều của dữ liệu lớn
- Quá nhiều điểm dữ liệu

Do vậy, người ta thường tiếp cận gần đúng bài toán tìm giá trị nhỏ nhất như sau: xuất phát từ một điểm mà ta coi là gần với nghiệm của bài toán, sau đó dùng một phép toán lặp để tiến dần đến nghiệm cần tìm, tức đến khi đạo hàm gần với 0. Gradient Descent (viết gọn là GD) và các biến thể của nó là một trong những phương pháp gần đúng theo cách tiếp cận trên và được dùng nhiều nhất. Thuật toán dựa vào chiều cập nhật nghiệm ngược với chiều của đạo hàm.

# GD cho hàm một biến




Hình 3: GD cho hàm một biến

Thuật toán GD:

- Khởi tạo điểm bắt đầu $x \in D$
- Chọn giá trị cho bước nhảy $\eta > 0$ ($\eta$ còn được gọi là **độ học** (learning rate) của phương pháp GD)
- Vòng lặp: Khi điều kiện dừng chưa thỏa thì thực hiện cập nhật điểm $x = x - \eta f'(x)$

Điều kiện dừng có thể là số vòng lặp ấn định trước hoặc ấn định độ lớn của đạo hàm bé hơn một số $\varepsilon > 0$: $|f'(x)| < \varepsilon$

# Cài đặt thuật toán GD

Cho hàm một biến $f(x) = x^2 + 4x + 3$. Thấy ngay hàm số đạt cực tiểu tại điểm $x^* = -2$ (Tại sao?).

a) Viết chương trình máy tính sử dụng thuật toán GD tìm giá trị gần đúng của $x^*$.

b) Thay đổi nghiệm (điểm cực tiểu) ban đầu và nhận xét.

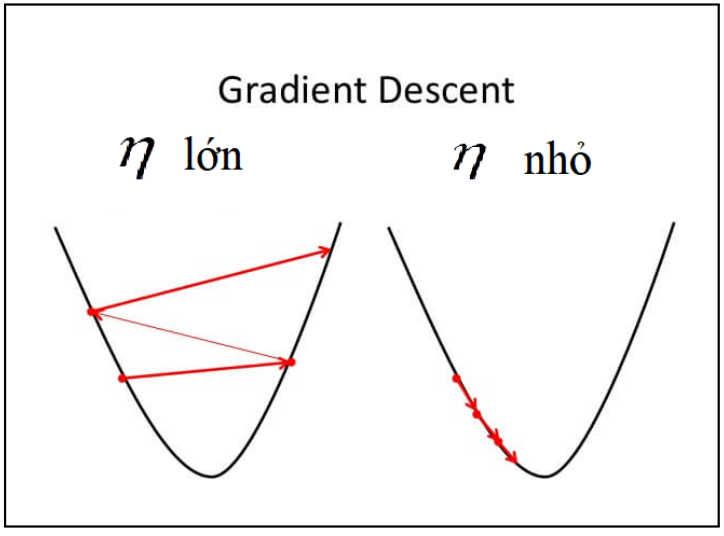
c) Thay đổi giá trị bước nhảy $\eta$ và nhận xét.

d) Bạn có thể chạy thuật toán GD bằng "sức người" cho 2 vòng lặp được không? 

# Tốc độ và độ chính xác của GD

Phụ thuộc vào:

- Chọn giá trị nghiệm ban đầu (tại sao?)
- Việc chọn $\eta$?
  Nếu $\eta$ quá lớn thì ta không hội tụ được về đích, nhưng nếu $\eta$ quá nhỏ thì ta lại mất rất nhiều thời gian để chạy giải thuật này, thậm chí không tìm được điểm cực tiểu mong muốn (khi hàm số có nhiều hơn một điểm cực tiểu).




Hình 4: Ảnh hưởng của độ học $\eta$

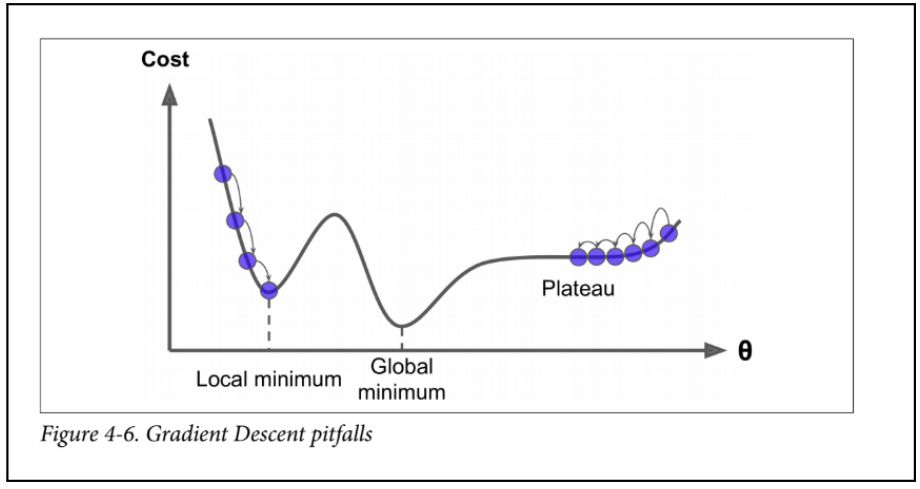



*Figure 4-6. Gradient Descent pitfalls*

Hình 5: Ảnh hưởng của độ học $\eta$

# Tốc độ hội tụ

Hình dưới, bên nào hội tụ nhanh hơn?

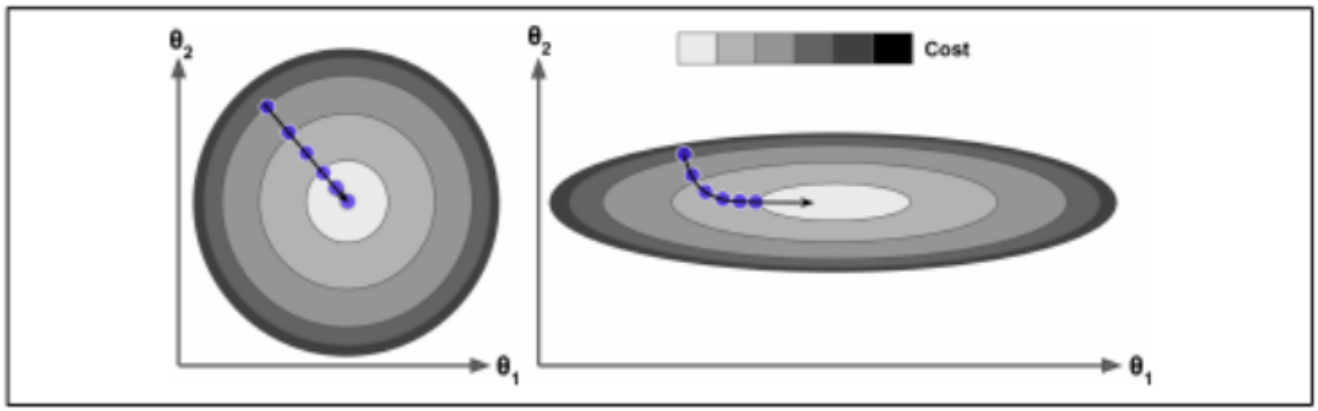



*Figure 4-7. Gradient Descent with and without feature scaling*

Tốc độ hội tụ: Nói chung khi hàm cần tìm cực trị là hàm lồi (convex function) thì chắc chắn ta tìm được cực trị với giá trị độ học nào đó với số vòng lặp là $O(1/\varepsilon)$ trong đó $\varepsilon$ phụ thuộc vào hình dáng của hàm mất mát ($\varepsilon$ là độ chính xác của nghiệm, còn gọi là dung sai (tolerance). Nghĩa là nếu ta muốn độ chính xác là $\varepsilon/10$ thì số vòng lặp sẽ nhiều hơn 10 lần.

# GD cho hàm nhiều biến

Thuật toán GD:

- Khởi tạo điểm bắt đầu $\mathbf{x} \in D \subset \mathbb{R}^N$
- Chọn giá trị cho bước nhảy $\eta > 0$
- Vòng lặp: Khi điều kiện dừng chưa thỏa thì thực hiện cập nhật điểm

$$x = x - \eta \nabla f(x) \tag{3}$$

Điều kiện dừng có thể là số vòng lặp ấn định trước hoặc ấn định độ lớn của véc tơ gradient bé hơn một số $\varepsilon > 0$ cho trước:

$$||\nabla f(\mathbf{x})||_2 = \sqrt{\sum_{i=1}^{N} \left[\frac{\partial}{\partial x_i} f(\mathbf{x})\right]^2} < \varepsilon, \tag{4}$$

Phương pháp GD này còn gọi là GD toàn phần (Batch GD(BGD))

# Cài đặt GD cho hàm 2 biến

Cho hàm hai biến $f(x_1, x_2) = x_1^2 + 3x_2^2 + 1$ có điểm cực trị là $\mathbf{x}^* = (0, 0)$ (Tại sao?). Hãy cài đặt thuật toán GD tìm nghiệm gần đúng với $\mathbf{x}^*$. Thay đổi các giá trị khởi tạo cho $\mathbf{x}$ và $\eta$ và nhận xét.

**Bạn có thể chạy thuật toán GD bằng "sức người" cho 2 vòng lặp được không?** 

# BGD cho mô hình hồi quy tuyến tính

Từ hàm mất mát

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{2m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^{\intercal}\mathbf{x}^{(i)} - y^{(i)}\right)^2 \tag{5}$$

Lấy đạo hàm riêng hàm $J(\boldsymbol{\theta})$ theo $\theta_j$ ta được

$$J'_{\theta_j}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^{\intercal}\mathbf{x}^{(i)} - y^{(i)}\right)x_j^{(i)} \tag{6}$$

Do đó vec tơ gradient của $J$ là

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = (J'_{\theta_0}(\boldsymbol{\theta}), \ldots, J'_{\theta_{n-1}}(\boldsymbol{\theta}))^T = \frac{1}{m}\mathbf{X}^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) \tag{7}$$

Như vậy, công thức lặp cho $\theta$ là

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{8}$$

* **TH-t124**: Chạy thử đoạn CT trong TL, thay đổi $\eta$ để được các hình trong tr 125

# Nội dung trình bày

3. Thực hành với python

- BT1. Chạy lại đoạn CT tr 124 với dữ liệu demo-data.xls
- BT2. (a) Bỏ số 2 trong (2/m) của đoạn CT BT1, chạy lại CT và nhận xét. (b) Bỏ ”m”, chạy lại CT và nhận xét.
- BT3. Thay đổi $\eta$, giá trị khởi tạo $\theta$ và nhận xét (Nên thay các giá trị có sự khác biệt lớn).
- BT4. Thay điều kiện dừng bằng điều kiện ”gardient $<$ epsilon ”

# Nội dung trình bày

4. Gradient descent ngẫu nhiên

# Gradient descent ngẫu nhiên (Stochastic Gradient descent)

- BGD: Mỗi lần cập nhật thì dùng toàn bộ dữ liệu huấn luyện (Các khó khăn có thể có là gì?)
  - Về khối lượng tính
  - Về tốc độ hội tụ
  - Tính ”thời sự” (online)
- SGD: Thay vì sử dụng toàn bộ tập dữ liệu để cập nhập tham số thì ta sử dụng từng dữ liệu một để cập nhập. Phương pháp như vậy được gọi là GD ngẫu nhiên. Về cơ bản ở mỗi lần cập nhập tham số, ta duyệt toàn bộ cặp mẫu $(\mathbf{x}_i, \mathbf{y}_i)$ và cập nhập tương tự như BGD như sau:

$$\boldsymbol{\theta}^{Next} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \tag{9}$$

Với phương trình hồi quy thì gradient tại một điểm dữ liệu là

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \mathbf{x}^{(i)}; y^{(i)}) = \mathbf{x}^{(i)T}(\mathbf{x}^{(i)}\boldsymbol{\theta} - y^{(i)}) \tag{10}$$

# SGD

- Tại một thời điểm ta chỉ tính đạo hàm trên một mẫu và cập nhật ma trận trọng số
- Mỗi lần duyệt qua tất cả các điểm trên toàn bộ dữ liệu ta gọi là epoch
  - Với BGD: Mỗi epoch là 1 lần cập nhật ma trận trọng số
  - Với SGD: Mỗi epoch tương ứng với m lần cập nhật trọng số
- Việc cập nhật từng điểm có thể giảm tốc độ thực hiện 1 epoch
- SGD chỉ yêu cầu một lượng epoch nhỏ và thường phù hợp với bài toán có dữ liệu lớn
- Thứ tự chọn điểm dữ liệu: sau mỗi epoch ta cần trộn thứ tự của các điểm dữ liệu để đảm bảo tính ngẫu nhiên
- Giải thuật SGD hội tụ nhanh hơn BGD

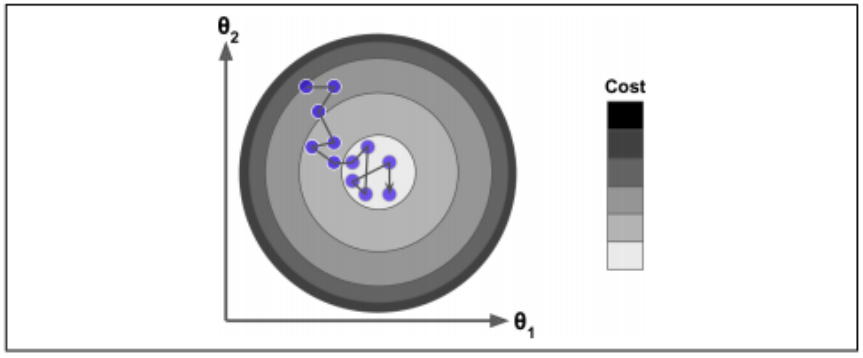



*Figure 4-9. Stochastic Gradient Descent*

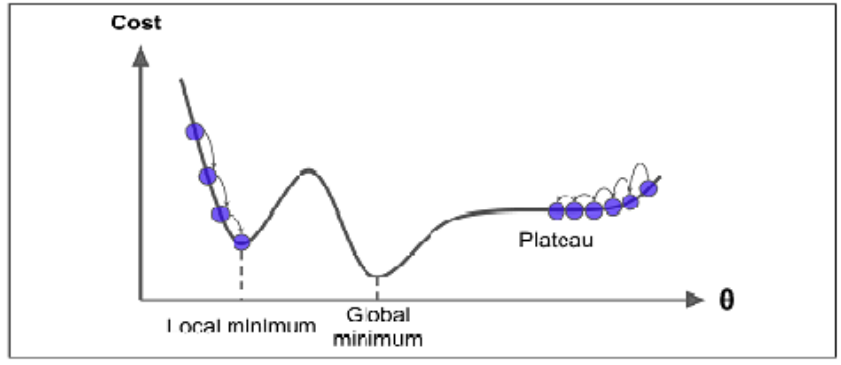



*Figure 4-6. Gradient Descent pitfalls*

Thực tế cho thấy chỉ lấy vài điểm là ta đã có thể xác định được gần đúng phương trình đường thẳng cần tìm rồi. Đây chính là ưu điểm của SGD - hội tụ rất nhanh.

# Bài tập với python

- BT1. Chạy vài lần CT SGD ở tr 127 với số liệu data-demo.xls và nhận xét. Bạn có so sánh gì giữa SGD và BGD ở trang 124
- BT2. Hàm ”learning-schedule” đóng vai trò gì trong BT1? Nếu bỏ qua hàm này thì CT ở BT1 sẽ thế nào?
- BT3. Chạy vài lần đoạn CT sử dụng hàm SGDRegressor của thư viện sklearn và nhận xét. Chú thích ý nghĩa các tham số của hàm này.

# Nội dung trình bày

5. Mini-Batch GD

Mini-batch GD là thuật toán pha trộn giữa BGD và SGD. Mini-batch GD bắt đầu mỗi epoch bằng việc xáo trộn ngẫu nhiên dữ liệu rồi chia toàn bộ dữ liệu thành các nhóm $n$ điểm (trừ mini-batch cuối có thể có ít hơn nếu m không chia hết cho n) và sử dụng mỗi nhóm Ở mỗi bước cập nhật tham số. Công thức cập nhật có thể viết dưới dạng:

$$\theta = \theta - \eta \nabla_\theta J(\theta; \mathbf{x}^{(i:i+n)}; \mathbf{y}^{(i:i+n)}) \tag{11}$$

- Khi nào Mini-Batch GD trở thành
  SGD
  BGD
- Kích thước thường được chọn của mini-batch là $2^k$, chẳng hạn: 64, 128, 256, 512, 1024,...

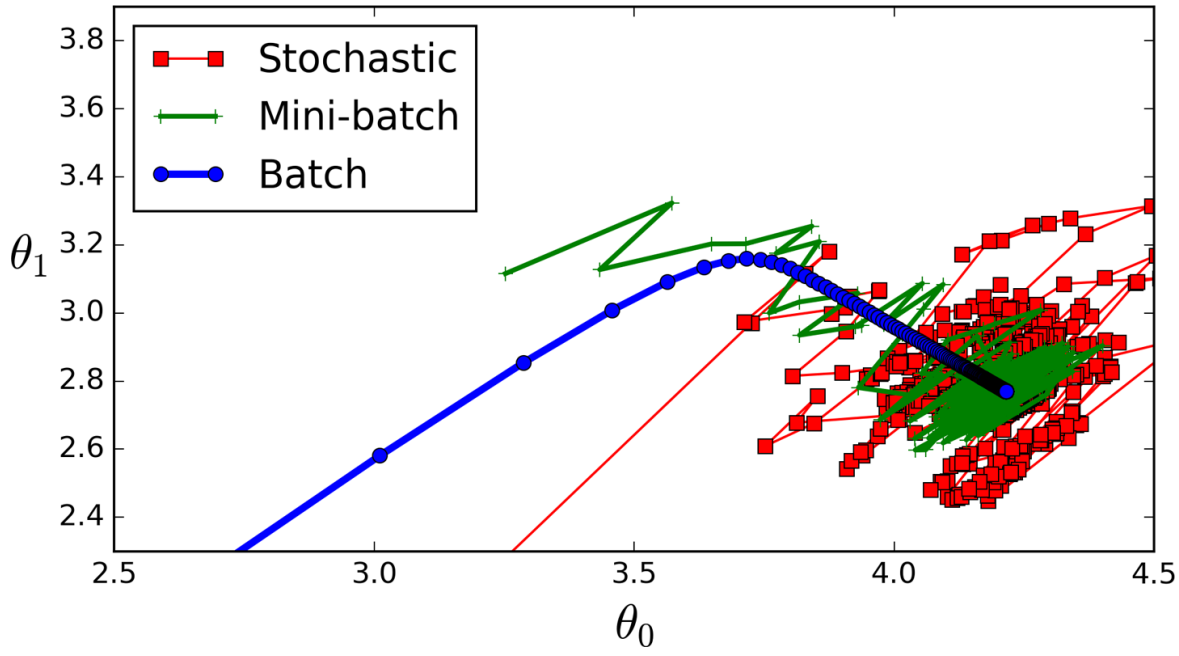



*Table 4-1. Comparison of algorithms for Linear Regression*

| Algorithm | Large *m* | Out-of-core support | Large *n* | Hyperparams | Scaling required | Scikit-Learn |
|---|---|---|---|---|---|---|
| Normal Equation | Fast | No | Slow | 0 | No | n/a |
| SVD | Fast | No | Slow | 0 | No | `LinearRegression` |
| Batch GD | Slow | No | Fast | 2 | Yes | `SGDRegressor` |
| Stochastic GD | Fast | Yes | Fast | ≥2 | Yes | `SGDRegressor` |
| Mini-batch GD | Fast | Yes | Fast | ≥2 | Yes | `SGDRegressor` |

Bài tập

- BT1. Cài đặt thuật toán Mini-Batch GD
- BT2. Sử dụng thư viện sklearn

# Tài liệu tham khảo


- Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, 2nd Edition của tác giả Aurélien Géron.

# Ghi chú về việc tính đạo hàm

Việc tính đạo hàm bằng công thức giải tích đôi khi rất phức tạp. Khi đó ta tính gần đúng đạo hàm tại một điểm như sau:Từ định nghĩa đạo hàm

$$f'(x) = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x)}{\varepsilon} \tag{12}$$

Ta xấp xỉ giá trị đạo hàm với $\varepsilon$ rất nhỏ:

$$f'(x) \approx \frac{f(x+\varepsilon) - f(x)}{\varepsilon} \tag{13}$$