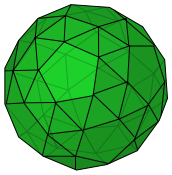
Projects in Mathematics and Applications

TỐI ƯU HÓA

Nhat Pham

23/07/2018



Mục lục

Danh pháp tiếng Anh

Danh sách những khái niệm Toán học trong tài liệu và danh pháp tiếng Anh của chúng để tiện cho những bạn tra cứu online:

1. Cực tiểu (Cực đại) Địa phương: Local Minimum (Maximum)
2. Cực tiểu (Cực đại) Toàn cục: Global Minimum (Maximum)
3. Bị chặn: Bounded
4. Đóng: Closed
5. Điểm Yên ngựa: Saddle Point
6. Phép thử Bậc hai: Second-order Test
7. Xác định dương: Positive Definite
8. Xác định âm: Negative Definite
9. Phương pháp Nhân tử Lagrange: Lagrange Multiplier Method
10. Học Có Giám sát: Supervised Learning
11. Hàm mục tiêu: Objective Function
12. Hàm Giả thuyết: Hypothesis
13. Hàm Mất mát: Loss Function
14. Kết hợp: Associative
15. Tham số hóa: Parameterized
16. Phân phối Xác suất Sinh Dữ liệu: Data-generating Distribution
17. Sai số Tổng quát: Generalization Error
18. Độ tăng: Rate of Increase
19. Ma trận: Matrix
20. Mẫu: Sample
21. Điều chỉnh Tốc độ Học: Learning Rate Schedule
22. Hàm Tốc độ Học Mũ: Exponential Learning Rate
23. Hàm Tốc độ Học Lũy thừa: Power Learning Rate
24. Dừng sớm: Early Stopping
25. Quán tính: Momentum
26. Gia tốc: Acceleration

1 Cực Trị Trong Giải Tích Nhiều Biến

1.1 Không Có Điều Kiện Chặn

Ở đây, ta chủ yếu làm việc với hàm $f : D \to \mathbb{R}$, với $D \subseteq \mathbb{R}^n$. Giả sử f có đạo hàm riêng bậc nhất và bậc hai liên tục trên tập xác định. Ta bắt đầu bằng các định nghĩa sau:

Định nghĩa 1.1 (Cực trị Địa phương)**.** Điểm $\boldsymbol{x} \in D$ được gọi là một **cực tiểu địa phương (local minimum)** của hàm f nếu tồn tại một lân cận

$$B(\boldsymbol{x}, R) = \{\boldsymbol{y} \in \mathbb{R}^n \,|\, ||\boldsymbol{x} - \boldsymbol{y}|| < R\} \subseteq D,$$

với $R > 0$ nào đó, sao cho:

$$f(\boldsymbol{x}) \leq f(\boldsymbol{y}), \quad \forall \boldsymbol{y} \in B(\boldsymbol{x}, R).$$

Đổi chiều bất đẳng thức trên, ta được định nghĩa của **cực đại địa phương (local maximum)**.

Định nghĩa 1.2 (Cực trị Toàn cục)**.** Điểm $\boldsymbol{x} \in D$ được gọi là một **cực tiểu toàn cục (global minimum)** của hàm f nếu:

$$f(\boldsymbol{x}) \leq f(\boldsymbol{y}), \quad \forall \boldsymbol{y} \in D.$$

Đổi chiều bất đẳng thức trên, ta được định nghĩa của **cực đại toàn cục (global maximum)**.

Ta có định lý sau về sự tồn tại của cực trị toàn cục:

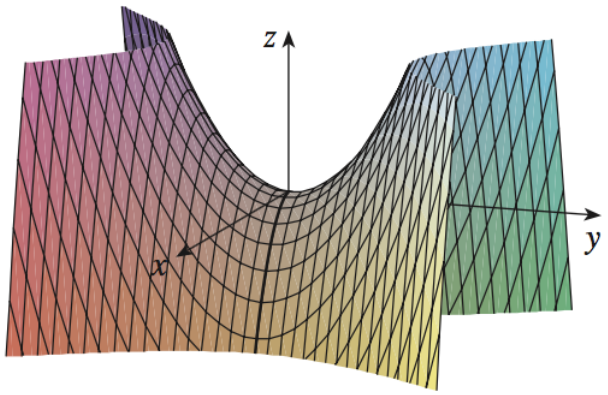
Định lý 1.3. *Nếu D bị chặn (bounded) và đóng (closed, chứa mọi điểm biên), hàm f sẽ đạt được cực đại và cực tiểu toàn cục trên D.*

Định lý sau là mở rộng của một kết quả quen thuộc trong giải tích hàm một biến:

Định lý 1.4. *Giả sử hàm f đạt được cực trị địa phương tại điểm $\boldsymbol{x}$. Khi đó ta có:*

$$\nabla f(\boldsymbol{x}) = \mathbf{0}.$$

Chú ý rằng đây chỉ là điều kiện cần, không phải là điều kiện đủ. Có những điểm $\boldsymbol{x}$ mà tại đây, gradient bằng $\mathbf{0}$, nhưng hàm số không đạt được cực trị; khi đó, điểm $\boldsymbol{x}$ sẽ được gọi là **điểm yên ngựa (saddle point)** (hình từ [8]):



Trong trường hợp hàm f có đạo hàm bậc hai, để xác định chính xác điểm $\boldsymbol{x}$ có $\nabla f(\boldsymbol{x}) = 0$ là loại điểm nào, chúng ta có thể sử dụng **phép thử bậc hai (second-order test)**. Ta tính **ma trận Hessian** của f tại $\boldsymbol{x}$:

$$[\mathbf{H}f(\boldsymbol{x})]_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}(\boldsymbol{x}).$$

Nhận xét 1.5.

- *Nếu $\mathbf{H}f(\boldsymbol{x})$ là một ma trận xác định dương (positive definite, tức là mọi giá trị riêng của nó đều dương), $\boldsymbol{x}$ là điểm cực tiểu địa phương.*
- *Nếu $\mathbf{H}f(\boldsymbol{x})$ là một ma trận xác định âm (negative definite, tức là mọi giá trị riêng của nó đều âm), $\boldsymbol{x}$ là điểm cực đại địa phương.*
- *Nếu $\mathbf{H}f(\boldsymbol{x})$ vừa có giá trị riêng dương, vừa có giá trị riêng âm (nhưng đều khác 0), $\boldsymbol{x}$ là một điểm yên ngựa.*

Ở các trường hợp khác, chúng ta không kết luận được gì.

1.2 Có Điều Kiện Chặn Dạng Đẳng Thức

Ta xét bài toán tối ưu:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}),$$

thoả mãn:

$$g_i(\boldsymbol{x}) = 0, \quad i \in \{1, 2, ..., m\},$$

với các hàm $f, g_i : \mathbb{R}^n \to \mathbb{R}$ là các hàm khả vi với đạo hàm riêng liên tục.

Để giải quyết bài toán này, ta sẽ phát biểu một điều kiện cần để điểm $\boldsymbol{x}^*$ là một cực trị địa phương của hàm f:

Phương Pháp Nhân Tử Lagrange (Lagrange Multiplier Method)

Ta định nghĩa **Lagrangian** của bài toán tối ưu:

$$\begin{aligned} \mathcal{L} \ : \ \mathbb{R}^n \times \mathbb{R}^m &\to \mathbb{R}, \\ (\boldsymbol{x}, \boldsymbol{\lambda}) &\mapsto f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i g_i(\boldsymbol{x}). \end{aligned}$$

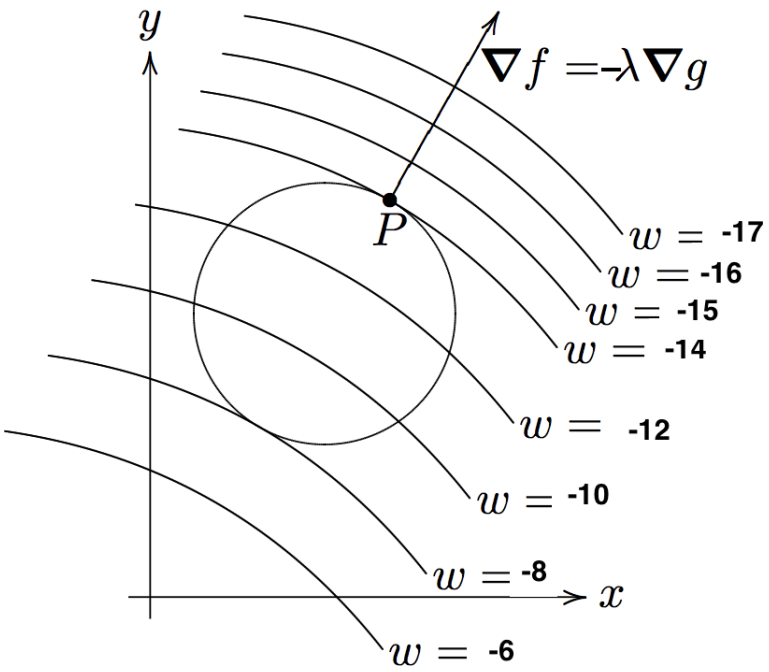
với $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, ..., \lambda_m]$ được gọi là **biến đối ngẫu** hay **vector nhân tử Lagrange**.
Nếu $\boldsymbol{x}^*$ là một điểm cực tiểu địa phương của bài toán tối ưu trên, ta có:

$$\exists \boldsymbol{\lambda}^* \in \mathbb{R}^m : \nabla_{\boldsymbol{x},\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}.$$

Để tìm cực tiểu toàn cục, ta có thể giải hệ phương trình có được từ điều kiện ở trên rồi xét thử tất cả các bộ nghiệm tìm được.

Chứng minh. Tham khảo ở [5]. □

Ý nghĩa hình học: Ở đây ta nêu ý nghĩa hình học trong trường hợp $n = 2, m = 1, g = g_i$ là một đường tròn:
Xét các đường mức $f(\boldsymbol{x}) = w$. Giả sử đường mức này không giao $g(\boldsymbol{x}) = 0$, ta sẽ đi về hướng của $g(\boldsymbol{x}) = 0$. Ngược lại, nếu $f(\boldsymbol{x}) = w$ cắt nhưng không tiếp xúc $g(\boldsymbol{x}) = 0$, ta luôn có thể đi về hướng sao cho giá trị của f giảm, đến khi f tiếp xúc với g. Điều này xảy ra khi $\nabla_{\boldsymbol{x}} f = k\nabla_{\boldsymbol{x}} f$. Đặt $\lambda = -k$, ta có ngay điều kiện $\nabla_{\boldsymbol{x}} f + \lambda g = 0$ (hình chỉnh sửa từ [5]):



Ví dụ 1.6. Giả sử $f, g : \mathbb{R}^2 \to \mathbb{R}$ là hai ánh xạ thoả mãn:

$$\begin{array}{rcl} \mathbb{R} \times \mathbb{R} & \to & \mathbb{R}, \\ (x, y) & \mapsto & f(x, y) = 3x + 4y, \\ (x, y) & \mapsto & g(x, y) = x^2 + y^2 - 1. \end{array}$$

Tìm giá trị nhỏ nhất của f trên $D = \{(x, y) \in \mathbb{R}^2 \mid g(x, y) = 0\}$.

Chứng minh. Sử dụng phương pháp nhân tử Lagrange, ta có nghiệm tối ưu (nếu có) của bài toán tối ưu trên sẽ là nghiệm của hệ phương trình:

$$\begin{cases} 3 + 2\lambda x = 0, \\ 2 + \lambda y = 0, \\ x^2 + y^2 = 1. \end{cases} .$$

Giải hệ phương trình trên cho ta hai bộ nghiệm (x, y, λ) là $\left(-\frac{3}{5}, -\frac{4}{5}, \frac{5}{2}\right)$ và $\left(\frac{3}{5}, \frac{4}{5}, -\frac{5}{2}\right)$.
Bộ nghiệm thứ nhất sẽ cho ta giá trị nhỏ nhất cần tìm:

$$\min f(x, y) = -5.$$

□

2 Tối Ưu Trong Bài Toán Học Có Giám Sát

Bài Toán Học Có Giám Sát (Supervised Learning)

Bài toán học có giám sát gồm các yếu tố sau:

- Dữ liệu đầu vào $x \in \mathcal{X}$ và nhãn $y \in \mathcal{Y}$ tương ứng với mỗi x.
- Mối quan hệ tương ứng nêu trên được cho bởi một **hàm mục tiêu** $g : \mathcal{X} \to \mathcal{Y}$, tức là $y = g(x)$ là nhãn tương ứng với x. Chúng ta không biết cách tính hàm số này; chúng ta chỉ biết một số cặp giá trị $(x, g(x))$ cụ thể. Đây là hàm số mà chúng ta muốn ước lượng.
- Tập H các **hàm giả thuyết** $h : \mathcal{X} \to \mathcal{Y}$. Từ tập H chúng ta muốn lựa chọn một hàm h để ước lượng tốt nhất hàm mục tiêu g.

Ví dụ 2.1. Dựa vào chiều cao x của căn nhà để dự đoán giá nhà $y = g(x)$, bằng cách lựa chọn hàm số để ước lượng g trong họ các hàm bậc hai $h(x) = ax^2 + bx + c$.
Để ước lượng mức độ sai lệch của hàm giả thuyết h đối với hàm mục tiêu g tại một điểm x, người ta thường sử dụng một **hàm mất mát** $\mathcal{L}(h(x), y)$, với $h(x)$ là dự đoán của h về nhãn của x, và $y = g(x)$ là nhãn tương ứng thực sự của x.

Ví dụ 2.2. Hàm mất mát **mean squared error** đo khoảng cách hình học giữa $h(x)$ và y:

$$\mathcal{L}(h(x), y) = \frac{1}{2}||h(x) - y||_2^2.$$

Để miêu tả tập H, thông thường người ta chọn một họ các hàm f_θ được **tham số hoá (parameterized)** bởi một vector $\theta \in \Theta$ nào đó, với Θ được gọi là **không gian tham số**.
Quá trình tham số hoá này đưa bài toán tìm kiếm các hàm h trong tập H về bài toán tìm θ trong một không gian tham số Θ, thường được chọn là các tập số quen thuộc như $\mathbb{R}^n$.

Ví dụ 2.3. Họ các đường thẳng $f(x) = ax + b$ trong trường hợp $\mathcal{X} = \mathcal{Y} = \mathbb{R}$. Với mỗi vector tham số $\theta = [a, b]$ thuộc không gian tham số $\Theta = \mathbb{R}^2$, ta có một đường thẳng tương ứng.
Trong thực tiễn, đôi khi một cặp (x_1, y_1) xuất hiện thường xuyên hơn (x_2, y_2). Khi đó sai lầm của hàm giả thuyết trong việc dự đoán (x_1, y_1) sẽ xuất hiện thường xuyên hơn so với sai lầm trong việc dự đoán (x_2, y_2).

Ta có thể thể hiện tần suất xuất hiện của một cặp (x, y) trong thực tiễn bằng cách xem chúng là các biến ngẫu nhiên có không gian mẫu $\mathcal{X}$ và $\mathcal{Y}$, phân phối theo một phân phối xác suất p_{data} nào đó (được gọi là **phân phối xác suất sinh dữ liệu (data-generating distribution)**):

$$(x, y) \sim p_{data}(x, y).$$

Đồng thời, ta cũng bổ sung giả thuyết các cặp (x_i, y_i) được sinh từ p_{data} độc lập với nhau, tức là:

$$p(x_2, y_2|x_1, y_1) = p(x_2, y_2) = p_{data}(x_2, y_2).$$

Việc lựa chọn một mô hình máy học f_θ, tham số hoá bởi θ, có thể được biểu diễn qua việc tối thiểu giá trị kì vọng của một hàm mất mát nào đó (gọi là **sai số tổng quát (generalization error)**) trên phân phối xác suất p_{data}:

$$\theta = \arg\min_{\theta \in \Theta} \mathbb{E}_{(x,y)\sim p_{data}} \mathcal{L}(f_\theta(x), y).$$

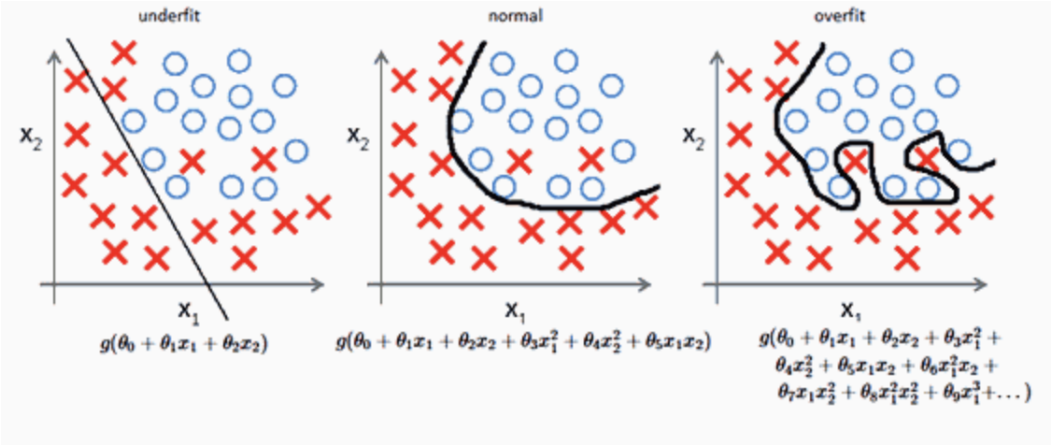
Trong thực tiễn, chúng ta thường không biết p_{data}, mà chỉ có một tập dữ liệu huấn luyện $\mathcal{D} = \{(x_1, y_1), ..., (x_n, y_n)\}$, là một mẫu (sample) từ p_{data}. Chúng ta sẽ ước lượng sai số tổng quát bằng một đại lượng trung bình của hàm mất mát trên $\mathcal{D}$, gọi là **empirical risk**:

$$\mathbb{E}_{(x,y)\sim p_{data}} \mathcal{L}(f_\theta(x), y) \approx J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f_\theta(x_i), y_i).$$

Việc tìm hàm f tối thiểu hàm J định nghĩa như trên, với hi vọng rằng hàm f cũng sẽ có sai số tổng quát thấp, được gọi là bài toán **empirical risk minimization**:

$$\theta = \arg\min_{\theta \in \Theta} J(\theta).$$

Lưu ý rằng việc giải bài toán tối ưu trên không đảm bảo mô hình sẽ hoạt động tốt khi thử nghiệm trên dữ liệu không có trong tập huấn luyện. Có những trường hợp mô hình sẽ 'học vẹt', quá chú trọng vào những chi tiết không quan trọng của tập huấn luyện (chẳng hạn như random noise hay sampling error), ảnh hưởng đến khả năng tổng quát hoá của mô hình khi được áp dụng vào thực tiễn. Hiện tượng này được gọi là **overfitting**, và thường xảy ra với các mô hình phức tạp có quá nhiều tham số (hình minh hoạ từ [6]):



Vì vậy, quá trình huấn luyện hệ thống máy học đôi khi không dừng khi hội tụ tại một điểm cực tiểu (địa phương hay toàn cục) của hàm $J(\theta)$, mà phụ thuộc vào một số điều kiện dừng sớm (xem phần 2.3).

Để đánh giá khả năng tổng quát hoá của mô hình từ tập dữ liệu được huấn luyện sang dữ liệu bất kì sinh từ p_{data}, người ta thường thử sử dụng mô hình trên một tập dữ liệu test riêng biệt (mà không được sử dụng để huấn luyện trước đó), cũng được xem là một mẫu từ p_{data}. Giá trị trung bình hàm mất mát (hoặc một metric nào đó) của mô hình trên tập dữ liệu test này sẽ là một "phép thử", cho ta thấy khả năng của mô hình đối với những dữ liệu chưa được học (unseen data).

Chú ý rằng đôi khi hàm mất mát mà chúng ta sử dụng cho empirical risk khác với hàm mất mát được sử dụng cho sai số tổng quát. Điều này thường xảy ra khi hàm mất mát chúng ta muốn tối ưu là một hàm khó tối ưu (chẳng hạn như không khả vi). Khi đó, hàm mất mát ở

empirical risk được gọi là một **hàm mất mát thay thế (surrogate loss)**.
Trong nhiều trường hợp, việc sử dụng hàm mất mát thay thế còn giúp cho cải thiện mô hình được chọn, vì nó có thể sẽ đặt thêm các điều kiện cho mô hình, khiến nó trở nên tốt hơn. Chẳng hạn, nếu ta sử dụng hàm mất mát **hinge loss** thay cho hàm mất mát $0-1$ (0 nếu đúng, 1 nếu sai), ta sẽ không chỉ khuyến khích mô hình dự đoán đúng, mà còn khuyến khích mô hình cải thiện mức độ chắc chắn của dự đoán của mình:

$$\mathcal{L}(f_\theta(x), y) = \sum_{i \neq y} \max(0, \Delta - z_y + z_i),$$

với z_i là điểm của lớp (class) thứ i.
Chú ý rằng khi đánh mô hình qua tập test, ta vẫn sử dụng hàm loss ban đầu, hoặc một metric nào đó được chọn trước (mà không cần quan tâm đến tính khả vi).

Ví dụ 2.4 (Hồi Quy Tuyến Tính)**.** Ở bài toán hồi quy tuyến tính, ta có dữ liệu đầu vào $\mathcal{D} = \{(x_1, y_1), ..., (x_n, y_n)\}$, với $x_i, y_i \in \mathbb{R}$. Chúng ta muốn tìm tham số $\theta = [a, b] \in \mathbb{R}^2$ sao cho siêu phẳng $\hat{y} = f_\theta(x) = ax + b$ 'dự đoán tốt' nhãn y từ giá trị của x.
Khái niệm 'dự đoán tốt' được thể hiện qua hàm mất mát:

$$\mathcal{L}(f_\theta(x), y) = \frac{1}{2}|ax + b - y|^2.$$

Việc lựa chọn tham số phù hợp có thể được đưa về bài toán tối ưu:

$$(a, b) = \arg\min_{(a,b)} J(a, b) = \arg\min_{(a,b)} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2}|ax_i + b - y_i|^2.$$

3 Gradient Descent

Thuật toán tối ưu thường dùng nhất trong các bài toán máy học có hàm mất mát khả vi là thuật toán gradient descent. Ý tưởng của thuật toán nằm ở nhận xét rằng vector gradient chỉ về hướng mà hàm có độ tăng (rate of increase) lớn nhất. Nhận xét này xuất phát từ công thức tính đạo hàm có hướng $\vec{u}$ (với $\vec{u}$ là một vector đơn vị) của hàm J tại điểm θ theo vector gradient:

$$D_{\vec{u}} J(\theta) = \nabla J(\theta) \cdot \vec{u} = ||\nabla J(\theta)||\,||\vec{u}|| \cos(\nabla J(\theta), \vec{u}) = ||\nabla J(\theta)|| \cos(\nabla J(\theta), \vec{u}).$$

Đại lượng trên đạt giá trị lớn nhất khi $\vec{u}$ cùng hướng với vector gradient, và đạt giá trị nhỏ nhất khi $\vec{u}$ ngược hướng với vector gradient.
Nhận xét trên cho ta thuật toán sau:

Thuật Toán Gradient Descent

Dữ liệu đầu vào: Một hàm $J(\theta)$ cần tìm cực tiểu theo biến θ, và một giá trị η quyết định độ dài của bước đi (step size) theo hướng gradient.
Bước 1: Khởi tạo giá trị ngẫu nhiên cho biến θ.
Bước 2: Cập nhật cho đến khi hội tụ hoặc đạt đến điều kiện dừng:

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta).$$

Ví dụ 3.1 (Hồi Quy Tuyến Tính)**.** Tìm ma trận hệ số W sao cho:

$$W = \arg\min_W J(W) = \arg\min_W \frac{1}{2n}||XW - Y||_2^2,$$

với n là số điểm dữ liệu, $X \in \mathbb{R}^{n\times m}$, $W \in \mathbb{R}^{m\times 1}$ và $Y \in \mathbb{R}^{n\times 1}$.

Chứng minh. Trước hết ta tính gradient:

$$\nabla_W J(W) = \frac{1}{n}(X^\top XW - X^\top Y).$$

Từ đây, chúng ta có công thức cập nhật:

$$W_{t+1} \leftarrow W_t - \frac{\eta}{n}(X^\top XW_t - X^\top Y).$$

□

3.1 Mini-batch và Stochastic Gradient Descent

Nhắc lại rằng khi tính gradient $\nabla_\theta J(\theta)$, chúng ta sử dụng toàn bộ dataset $\mathcal{D}$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{(x,y)\in\mathcal{D}} \nabla_\theta \mathcal{L}(f_\theta(x), y) = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \nabla_\theta \mathcal{L}(f_\theta(x), y).$$

Trong trường hợp dataset quá lớn, việc sử dụng toàn bộ dataset cho mỗi lần cập nhật là bất khả thi. Ta có nhận xét rằng nếu ta lấy một batch $\mathcal{D}_i$ rồi tính giá trị trung bình của hàm mất mát trên $\mathcal{D}_i$:

$$\frac{1}{|\mathcal{D}_i|} \sum_{(x,y)\in\mathcal{D}_i} \nabla_\theta \mathcal{L}(f_\theta(x), y),$$

ta sẽ được một ước lượng khá tốt[1] của $\nabla_\theta J(\theta)$, nếu các batch $\mathcal{D}_i$ có kích cỡ (gần) bằng nhau. Từ nhận xét trên, ta sẽ phân hoạch dataset thành từng batch nhỏ $\mathcal{D} = \bigcup_{i=1}^{N} \mathcal{D}_i$, với mỗi batch có độ lớn gần bằng và hai batch bất kì không chứa điểm chung nào, rồi lần lượt cập nhật cho mỗi batch:

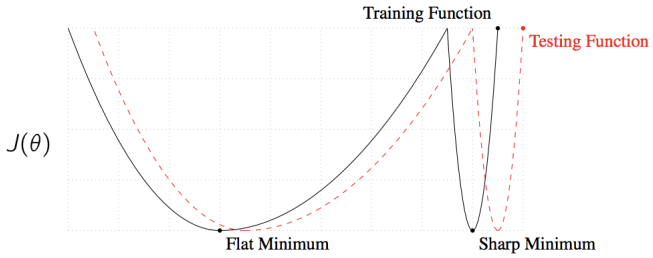
Thuật Toán Mini-Batch Gradient Descent

Dữ liệu đầu vào: Một hàm $J(\theta)$ cần tìm cực tiểu theo biến θ, và một giá trị η quyết định độ dài của bước đi (step size) theo hướng gradient.
Bước 1: Khởi tạo giá trị ngẫu nhiên cho biến θ.
Bước 2: Lặp lại cho đến khi hội tụ hoặc đạt đến điều kiện dừng.
Bước 2a: Phân hoạch ngẫu nhiên dataset $\mathcal{D}$ thành N batch nhỏ, mỗi batch có số dữ liệu (gần) bằng nhau.
Bước 2b: Với $i = 1, 2, ..., N$:

$$\theta \leftarrow \theta - \eta \frac{1}{|\mathcal{D}_i|} \sum_{(x,y)\in\mathcal{D}_i} \nabla_\theta \mathcal{L}(f_\theta(x), y).$$

1 Cụ thể là một **unbiased estimator**

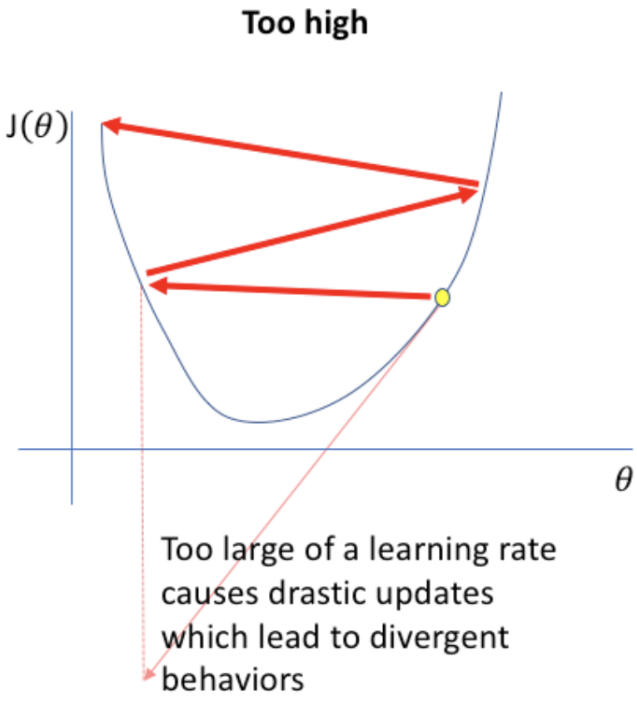
Khi độ lớn của mỗi batch bằng 1, thuật toán được gọi là Stochastic Gradient Descent.
Ngoài việc giảm gánh nặng về bộ nhớ, thực tiễn cho thấy batch size càng nhỏ, khả năng tổng quát hoá của mô hình máy học càng tốt, đặc biệt là với các mô hình lớn. Keskar và các đồng nghiệp đưa ra giả thuyết rằng thuật toán stochastic gradient descent dẫn đến một điểm tối thiểu "phẳng" hơn, khiến cho mô hình hoạt động tốt hơn ở giai đoạn test, khi mà loss surface có thể bị xê dịch so với loss surface ở giai đoạn train [4]:



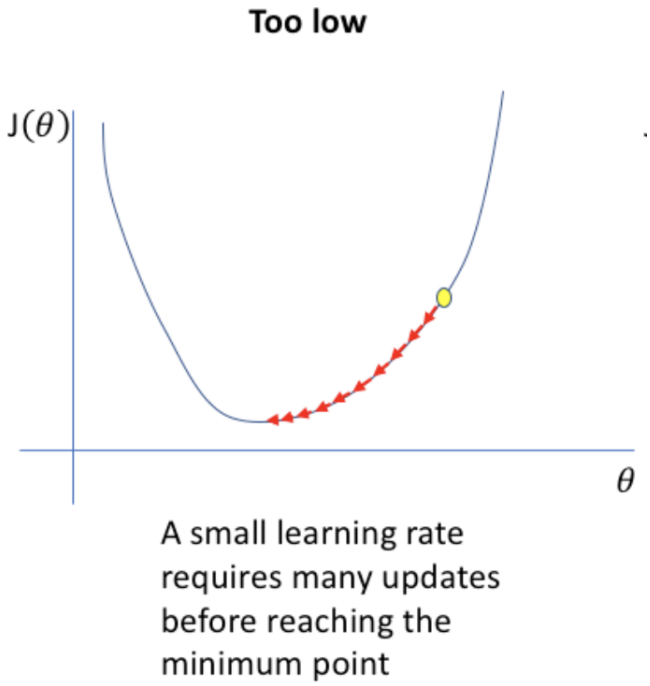
Samuel Smith và Quoc V. Le giải thích rằng chính những noise xuất hiện qua việc ước lượng vector gradient trên mini-batch giúp thuật toán tránh được những điểm cực tiểu địa phương "nhọn" [7].
Tuy nhiên, việc sử dụng batch size nhỏ làm tăng phương sai của ước lượng vector gradient, khiến cho thuật toán trở nên thiếu ổn định hơn. Điều này buộc chúng ta phải sử dụng tốc độ học nhỏ, có thể làm cho quá trình học trở nên chậm hơn.
Đồng thời, với các model lớn, việc sử dụng batch size lớn sẽ giúp cho quá trình training bằng GPU hiệu quả hơn. Chú ý rằng với các thiết kế GPU hiện tại, batch size là luỹ thừa của 2 sẽ là hiệu quả nhất.

3.2 Điều Chỉnh Tốc Độ Học (Learning Rate Schedule)

Thuật toán gradient descent khá nhạy cảm với tốc độ học η. Nếu η quá lớn, thuật toán sẽ dễ bỏ lỡ cực tiểu và thậm chí không hội tụ (hình minh hoạ từ [3]):



Ngược lại, nếu η quá nhỏ, thuật toán sẽ hội tụ rất chậm và khó rời khỏi cực trị địa phương hay điểm yên ngựa (hình minh hoạ từ [3]):



Thay vì sử dụng một tốc độ học η cố định cho thuật toán gradient descent, chúng ta có thể sử dụng một hàm $\eta(t)$ để điều chỉnh độ lớn của tốc độ học theo thời gian t (số lần cập nhật đã thực hiện).
Hàm $\eta(t)$ cần có giá trị lớn ban đầu để thuật toán nhanh chóng tìm ra các khu vực dễ có cực tiểu, và giảm dần để đảm bảo thuật toán hội tụ đến cực tiểu. Một số lựa chọn cho hàm tốc độ học là:

- Hàm tốc độ học mũ (exponential learning rate):

$$\eta(t) = \eta_0 10^{-\frac{t}{r}},$$

  với η_0 là tốc độ học ban đầu, và r là hệ số giảm.

- Hàm tốc độ học luỹ thừa (power learning rate):

$$\eta(t) = \eta_0 \left(1 + \frac{t}{r}\right)^{-c}.$$

3.3 Dừng Sớm (Early Stopping)

Như đã nhắc ở trên, đôi khi việc tìm θ tối thiểu $J(\theta)$ sẽ không cho ta một mô hình tốt nhất. Để tránh hiện tượng overfit, sau mỗi k bước cập nhật chúng ta có thể thử (validate) mô hình đang được huấn luyện trên một tập dữ liệu (thường được gọi là **validation set** hay **dev set**). Nếu giá trị hàm loss trên tập dữ liệu này không không thấp hơn giá trị ở các lần thử trước vượt quá p lần thử liên tiếp (p được gọi là **patience**), ta dừng quá trình huấn luyện lại.
Chú ý rằng tập validation này thường được rút ra từ tập huấn luyện. Điều này cũng đồng nghĩa

với việc trong quá trình huấn luyện, chúng ta sẽ phải sử dụng ít dữ liệu hơn. Thông thường, người ta sẽ huấn luyện hai lần:

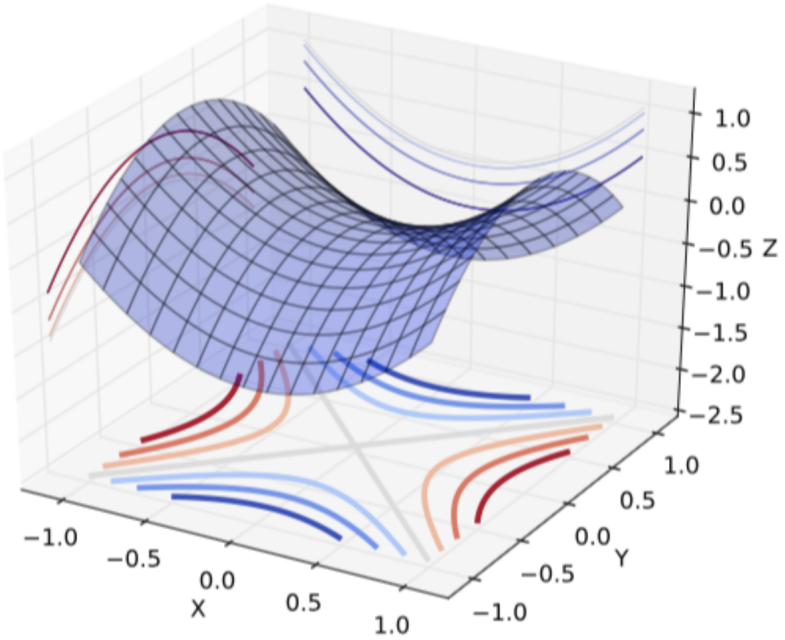
- Lần thứ nhất với tập validation để xác định số lần cập nhật hoặc loss có thể đạt được mà không overfit.
- Lần thứ hai với toàn bộ dữ liệu huấn luyện cho đến khi đạt được số lần cập nhật (hoặc loss giảm đến mức đạt được ở lần đầu tiên).

Bạn đọc có thể xem thêm ở phần 7.8 của [2] để biết thêm chi tiết.

4 Một Số Phiên Bản Cải Tiến Của Gradient Descent

4.1 Các Khó Khăn Khi Tối Ưu Bằng Thuật Toán Gradient Descent

Thuật toán gradient descent sẽ hội tụ khi vector gradient (rất gần) bằng $\mathbf{0}$. Tuy nhiên, điều này không đảm bảo rằng điểm hội tụ sẽ là một điểm cực tiểu toàn cục. Đôi khi, thuật toán sẽ hội tụ tại một điểm cực tiểu địa phương. Nếu không may, giá trị hàm J tại điểm cực tiểu địa phương này sẽ quá lớn, khiến cho mô hình không đủ tốt. Trường hợp tệ nhất là thuật toán sẽ hội tụ tại một điểm yên ngựa (hình từ [9]):



Một vấn đề khác là sự nhạy cảm của thuật toán gradient descent đối với tốc độ học. Chúng ta đã đưa ra một biện pháp thủ công để điều chỉnh tốc độ học, tuy nhiên thực tiễn cho ta thấy việc điều chỉnh tốc độ học một cách hợp lý phụ thuộc hoàn toàn vào hàm mất mát và không gian tham số của bài toán và mô hình mà chúng ta đang tối ưu. Thông tin này chỉ có thể được tìm hiểu phần nào qua vector gradient ở từng lần cập nhật.
Hai vấn đề trên sẽ đưa ta đến với các biến thể sau của thuật toán gradient descent:

4.2 Gradient Descent với Momentum

Ý tưởng: Chúng ta sẽ lưu thông tin các lần update trước vào một vector m. Điều này sẽ giúp cho thuật toán vượt qua những vùng của không gian hệ số mà gradient quá nhỏ để thuật toán có thể thoát ra khỏi (thường được gọi là các "valley"), hay các local optima:

$$\begin{aligned} m &\leftarrow m + \eta \nabla_\theta J(\theta), \\ \theta &\leftarrow \theta - m. \end{aligned}$$

Vector m thường được hiểu là momentum (quán tính) hay acceleration (gia tốc). Tên gọi này xuất phát từ cách hiểu rằng chúng ta sử dụng "đà" từ các lần cập nhật trước để vượt qua những khu vực khó tối ưu.
Hệ số β quyết định mức độ quan trọng của momentum trong quá khứ đối với bước cập nhật hiện tại. β càng nhỏ, mức độ quan trọng càng thấp; khi $\beta = 0$ thì thuật toán tương đương với gradient descent cơ bản. Theo [1], β thường được chọn bằng 0.9.

4.3 Thuật toán RMSProp

Ý tưởng: Phương nào càng "dốc" (thể hiện qua độ lớn của thành phần tương ứng của vector gradient), tốc độ học càng phải nhỏ.
Thuật toán RMSProp tinh chỉnh bước gradient theo nhận xét này bằng cách sử dụng một vector lưu trữ để lưu lại thông tin về độ lớn của vector gradient ở mỗi hướng ở các lần lặp trước:

$$s \leftarrow \beta s + (1-\beta) \nabla J(\theta) \otimes \nabla J(\theta).$$

Chú ý $\otimes$ ở đây là phép nhân theo phần tử. Hệ số phân rã β có tác dụng làm "dịu" lại tốc độ tăng của vector s, và thường được chọn là $\beta = 0.9$ [1].
Công thức cập nhật của RMSProp tương tự như công thức cập nhật chuẩn của thuật toán Gradient Descent, nhưng vector được cộng thêm sẽ được chia (theo phần tử) cho một vector được tính dựa trên vector lưu trữ:

$$\theta \leftarrow \theta - \eta \nabla J(\theta) \oslash (\sqrt{s} + \varepsilon).$$

Phần tử ε được thêm vào để tránh phép chia cho 0, và thường là các giá trị nhỏ như 10^{-8} [1]. Như vậy, nếu phần tử thứ i của vector gradient càng lớn ở các lần lặp trước thì tốc độ học tương ứng của nó ở lần lặp này sẽ càng nhỏ và ngược lại.

4.4 Thuật toán Adam

Thuật toán Adam là sự kết hợp của thuật toán momentum và RMSProp:

$$\begin{aligned} m &\leftarrow \beta_1 m + (1-\beta_1) \nabla J(\theta), \\ s &\leftarrow \beta_2 s + (1-\beta_2) \nabla J(\theta) \otimes \nabla J(\theta), \\ m' &\leftarrow \frac{m}{1-\beta_1^t}, \\ s' &\leftarrow \frac{s}{1-\beta_2^t}, \\ \theta &\leftarrow \theta - \eta m' \oslash (\sqrt{s'} + \varepsilon). \end{aligned}$$

Với t là số lần lặp tính đến lần lặp hiện lại (tính từ 1). Chú ý rằng bước 3 và bước 4 đảm bảo để thuật toán bắt đầu bằng những bước lớn hơn (do m và s thường được khởi tạo bằng 0).

Thuật toán này là một trong những thuật toán thông dụng nhất khi huấn luyện các mạng neuron sâu (Deep Neural Network).
Theo [1], các giá trị $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$ là các giá trị thường được chọn.

5 Phương Pháp Newton (Newton's Method)

Ý tưởng: Phương pháp Newton sử dụng các thông tin địa phương bậc 2 để tìm điểm cực trị. Xét bài toán tối ưu hàm một chiều $f(x)$, khai triển Taylor bậc 2 của hàm này quanh một điểm x_t là:

$$\begin{aligned} f(x) \approx f_{x_t}(x) &= f(x_t) + f'(x_t)(x - x_t) + \frac{1}{2} f''(x_t)(x - x_t)^2 \\ &= f(x_t) + f'(x_t)\Delta x + f''(x_t)\frac{1}{2}\Delta x^2. \end{aligned}$$

Ta muốn tìm bước $\Delta x = x - x_t$ sao cho $x = x_t + \Delta x$ là một điểm thoả $f'(x) = 0$. Điều này tương đương với:

$$\begin{aligned} & 0 = \frac{df(x)}{dx} \approx \frac{df_{x_t}(x)}{dx} = \frac{df_{x_t}(x)}{d\Delta x} = f'(x_t) + f''(x_t)\Delta x, \\ \iff \quad & \Delta x = -\frac{f'(x_t)}{f''(x_t)}. \end{aligned}$$

Mở rộng ra với hàm $J(\theta)$ không gian nhiều chiều, ta được công thức cập nhật sau để tìm một điểm cực trị cho hàm J theo biến θ, với tốc độ học η:

$$\theta_{t+1} \leftarrow \theta_t - \eta[\mathbf{H}J(\theta_t)]^{-1}\nabla_\theta J(\theta_t).$$

Với $\mathbf{H}J(\theta)$ là ma trận Hessian của hàm $J(\theta)$:

$$[\mathbf{H}J(\theta)]_{i,j} = \frac{\partial^2 J}{\partial\theta_i.\partial\theta_j}(\theta).$$

Chú ý rằng điểm cực trị mà bài thuật toán tìm được có thể là tối thiểu, tối ưu, hay là điểm yên ngựa, tuỳ thuộc vào tính chất của ma trận Hessian. Thuật toán chỉ hội tụ về điểm tối thiểu nếu ma trận Hessian là ma trận xác định dương (positive definite). Một cách để "ép" thuật toán tiến về cực tiểu là thực hiện theo công thức biến đổi sau:

$$\theta_{t+1} \leftarrow \theta_t - \eta[\mathbf{H}J(\theta_t) + \alpha I]^{-1}\nabla_\theta J(\theta_t).$$

Việc cộng thêm αI vào ma trận Hessian sẽ giúp cho mỗi giá trị riêng của ma trận Hessian tăng thêm một giá trị α, để giúp đảm bảo tính xác định dương của ma trận được nghịch đảo. Các giá trị riêng của ma trận Hessian càng âm, giá trị α càng phải lớn. Tuy nhiên, nếu α quá lớn, ma trận được nghịch đảo sẽ gần như là tích của một hằng số và ma trận đơn vị, khiến cho thuật toán trở thành một trường hợp đặc biệt của thuật toán gradient descent.
Trong thực tiễn, một vấn đề khá lớn của thuật toán là tính nghịch đảo của ma trận Hessian $\mathbf{H}J(\theta)$. Thông thường, người ta tính trực tiếp giá trị bước $\Delta\theta$ là nghiệm của hệ phương trình tuyến tính sau:

$$[\mathbf{H}J(\theta)]\Delta\theta = -\nabla_\theta J(\theta).$$

Các thuật toán ước lượng trực tiếp $[\mathbf{H}J(\theta)]^{-1}$ hay $\Delta J(\theta)$ được gọi là các thuật tựa - Newton (quasi - Newton) (thuật toán BFGS, v.v).

Tài liệu

[1] Aurélien Geron. *Hands on Machine Learning with Scikit-Learn and Tensorflow*. O'Reilly Media, 2017.

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville. *Deep Learning*. MIT Press, 2016.

[3] Jeremy Jordan. Setting the learning rate of your neural network. https://www.jeremyjordan.me/nn-learning-rate/.

[4] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. arXiv preprint arXiv:1609.04836 (2016).

[5] MIT OpenCourseware. Proof of Lagrange Multipliers. https://ocw.mit.edu/courses/mathematics/18-02sc-multivariable-calculus-fall-2010/2.-partial-derivatives/part-c-lagrange-multipliers-and-constrained-differentials/session-40-proof-of-lagrange-multipliers/MIT18_02SC_notes_22.pdf.

[6] ML Wiki. Overfitting. http://mlwiki.org/index.php/Overfitting.

[7] Samuel L. Smith, Quoc V. Le. A Bayesian Perspective on Generalization and Stochastic Gradient Descent. arXiv preprint arXiv:1710.06451 (2018).

[8] James Stewart. *Calculus: Early Transcendental, 8th Edition*. Cengage Learning, 2015.

[9] Daniel Takeshi. Gradient Descent Converges to Minimizers. https://danieltakeshi.github.io/2016-03-26-gradient-descent-converges-to-minimizers/.

[10] VietAI. "Giới Thiệu Tổng Quát Về Machine Learning. Thư Viện Tensorflow."