Chương 4
Tính toán số

- Mục lục
  - 4.1 Overflow và Underflow
  - 4.2 Tính kém điều hòa
  - 4.3 Tối ưu dựa trên gradient
    - 4.3.1 Ma trận Jacobi và ma trận Hesse
  - 4.4 Tối ưu có ràng buộc
  - 4.5 Ví dụ: Bình phương cực tiểu tuyến tính

Các thuật toán ML thường yêu cầu một khối lượng tính toán rất lớn. Các thuật toán này giải các bài toán bằng cách cập nhật các tham số qua rất nhiều vòng lặp thay vì đưa ra một công thức cho kết quả cuối cùng. Các phép toán chủ yếu là tối ưu hoá và giải các hệ phương trình tuyến tính. Ngay cả việc đánh giá một hàm số đơn giản liên quan tới số thực trên máy tính cũng là một bài toán khó, vì số thực không thể được biểu diễn chính xác với một lượng bộ nhớ hữu hạn.

4.1 Overflow và Underflow

Khó khăn cơ bản khi thực hiện các phép toán liên tục trên máy tính là ta cần biểu diễn các số thực vô hạn với một lượng bit hữu hạn. Điều đó có nghĩa là với hầu hết các số thực, ta chấp nhận một lượng sai số khi biểu diễn xấp xỉ trên máy tính. Trong nhiều trường hợp, sai số này chính là sai số khi làm tròn. Tuy nhiên, việc làm tròn số lại gây ra nhiều vấn đề, đặc biệt là khi kết hợp nhiều bước tính toán. Nhiều sai số nhỏ kết hợp gây ra sai số đủ lớn để khiến các thuật toán thất bại trong thực tế, dù nó hoàn toàn đúng trên lý thuyết nếu ta bỏ qua vấn đề này khi thiết kế.

Có một dạng lỗi làm tròn rất nguy hiểm là *underflow*. Underflow xảy ra khi các số gần 0 được làm tròn thành 0. Nhiều hàm sẽ thay đổi tính chất một cách căn bản khi đầu vào là 0, thay vì là một số dương rất nhỏ. Ví dụ, ta thường tránh phép chia cho 0 (một số môi trường phần mềm sẽ sinh ra exception khi trường hợp này xảy ra, một số khác sẽ trả về giá trị NaN - Not a Number) hoặc phép lấy

logarit của 0 (đầu ra thường được coi là $-\infty$, sau đó nó sẽ trở thành NaN nếu nó được sử dụng trong các phép tính số học tiếp theo đó).

Một dạng lỗi cũng rất nguy hiểm khác là *overflow*, xảy ra khi các số với giá trị rất lớn bị xấp xỉ thành ∞ hoặc $-\infty$. Các phép tính số học sau đó sẽ quy các giá trị vô hạn này thành NaN.

Một ví dụ về một hàm số cần phải được xem xét để tránh underflow và overflow là *hàm softmax*. Hàm softmax thường được dùng để dự đoán các xác suất trong một một phân phối multinoulli, và được định nghĩa như sau:

$$\mathrm{softmax}(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \qquad (4.1)$$

Xét trường hợp khi tất cả x_i có giá trị bằng một hằng số c. Về mặt giải tích, ta dễ thấy tất cả đầu ra của hàm đều bằng $\frac{1}{n}$. Nhưng về mặt số học, điều này có thể không xảy ra khi c có trị tuyệt đối cực lớn. Nếu c rất âm, thì e^c sẽ bị underflow. Điều này dẫn tới việc mẫu số của hàm softmax sẽ bằng 0 và kết quả của hàm là vô định. Khi c rất dương, e^c sẽ bị overflow và kết quả của hàm cũng là vô định. Cả hai vấn đề này đều có thể giải quyết được bằng cách tính gián tiếp hàm $\mathrm{softmax}(\boldsymbol{x})$ thông qua hàm $\mathrm{softmax}(\boldsymbol{z})$ với $\boldsymbol{z} = \boldsymbol{x} - \max_i x_i$. Một vài biến đổi đại số đơn giản chỉ ra rằng giá trị của hàm softmax không bị thay đổi về mặt toán học khi ta cộng hoặc trừ vector đầu vào với một đại lượng vô hướng (chẳng hạn như hằng số). Khi trừ các phần tử của vector cho $\max_i x_i$, giá trị đối số lớn nhất của hàm e sẽ là 0. Điều này loại bỏ nguy cơ bị overflow. Tương tự, khi mẫu số của hàm có ít nhất một thành phần có giá trị bằng 1 (ứng với trường hợp đối số của hàm e bằng 0 nói trên), ta sẽ loại bỏ khả năng mẫu số bị underflow gây bởi phép chia cho 0.

Tuy nhiên, vẫn tồn tại một vấn đề nhỏ. Underflow ở tử số vẫn có thể khiến giá trị của cả biểu thức bằng 0. Có nghĩa là nếu ta tính toán giá trị của hàm $\log \mathrm{softmax}(\boldsymbol{x})$ bằng cách tính giá trị hàm softmax trước rồi thực hiện phép lấy logarit, ta có thể sẽ thu được kết quả sai là $-\infty$. Thay vào đó, ta cần cài đặt một hàm riêng để tính toán hàm log softmax theo một cách bình ổn kết quả về mặt số học hơn. Ta có thể bình ổn hàm log softmax bằng kĩ thuật tương tự đã dùng với hàm softmax.

Trong phần lớn cuốn sách này, ta sẽ không liệt kê cụ thể tất cả vấn đề liên quan tới tính toán số khi thực thi các thuật toán nêu ra trong sách. Những lập trình viên dùng ngôn ngữ bậc thấp thường quan tâm tới các vấn đề liên quan tới tính toán

số khi lập trình các thuật toán DL. Hầu hết bạn đọc có thể đơn giản chỉ dựa vào các framework bậc thấp đã cung cấp các cách cài đặt ổn định. Trong vài trường hợp, ta có thể cài đặt một thuật toán mới và các thư viện bậc thấp sẽ tự bình ổn kết quả của thuật toán đó. Ví dụ, Theano [Bergstra et al., 2010; Bastien et al., 2012] là một thư viện tự động phát hiện và bình ổn nhiều biểu thức không ổn định về mặt tính toán số thường gặp khi nghiên cứu DL.

4.2 Tính kém điều hòa

Tính điều hòa, nói đến mức độ thay đổi đầu ra của hàm tương ứng khi biến đầu vào có những thay đổi nhỏ. Những hàm thay đổi nhanh khi biến đầu vào chỉ biến động một chút sẽ gây ra vấn đề trong hệ thống tính toán, bởi các sai số làm tròn của đầu vào có thể dẫn đến những thay đổi lớn ở đầu ra.

Xét hàm $f(\boldsymbol{x}) = \boldsymbol{A}^{-1}\boldsymbol{x}$. Khi ma trận $\boldsymbol{A} \in \mathbb{R}^{\mathrm{n}\times\mathrm{n}}$ có một phép phân tích giá trị riêng, *hệ số điều hòa* (*condition number*) của nó là:

$$\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right| \qquad (4.2)$$

Đây là tỉ số giữa giá trị tuyệt đối của *giá trị riêng* (eigenvalue) lớn nhất và nhỏ nhất. Khi tỉ số này lớn, phép lấy ma trận nghịch đảo cực kì nhạy cảm với sai số của đầu vào.

Sự nhạy cảm này là tính chất nội tại của ma trận, chứ không phải kết quả của sai số làm tròn khi nghịch đảo ma trận. Các ma trận kém điều hòa sẽ khuếch đại những sai số tồn tại trước đó khi ta thực hiện phép nhân với ma trận nghịch đảo thực sự. Trong thực tế, sai số sẽ càng nghiêm trọng hơn bởi các sai số liên quan tới tính toán số trong quá trình tính nghịch đảo.

4.3 Tối ưu dựa trên gradient

Phần lớn các thuật toán DL đều liên quan tới tối ưu theo cách này hay cách khác. Tối ưu là tác vụ cực tiểu hóa hoặc cực đại hóa hàm $f(\boldsymbol{x})$ bằng cách thay đổi $\boldsymbol{x}$. Ta thường đưa các bài toán tối ưu về dạng cực tiểu hóa hàm $f(\boldsymbol{x})$. Ta có thể giải quyết bài toán tìm cực đại thông qua một thuật toán tìm cực tiểu bằng cách cực tiểu hoá hàm $-f(\boldsymbol{x})$.

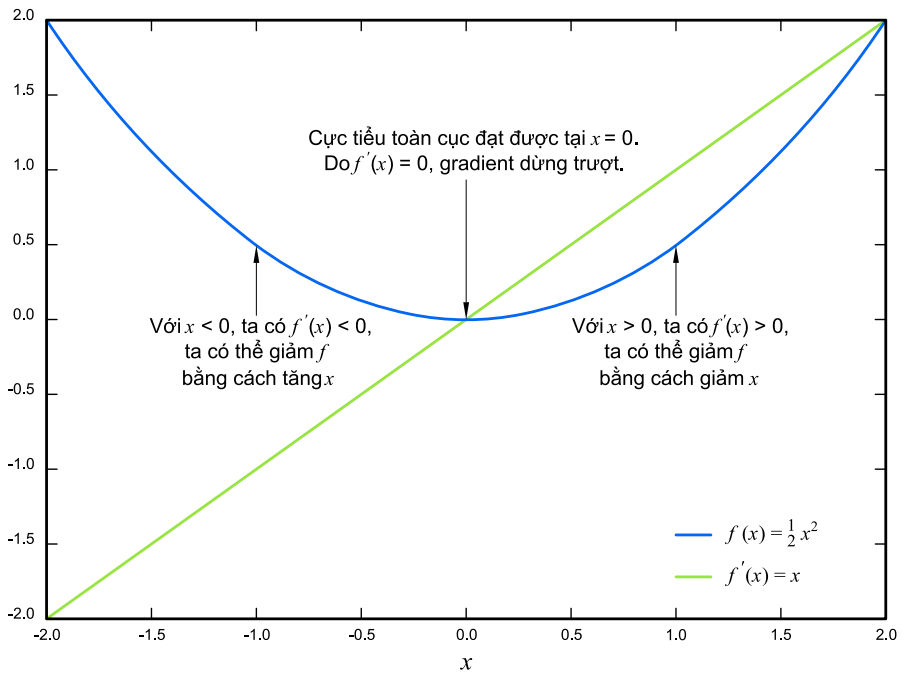
Các hàm ta cần cực tiểu hóa hoặc cực đại hóa được gọi là *hàm mục tiêu* (*objective function*), hoặc *tiêu chuẩn* (*criterion*). Khi ta cực tiểu hoá chúng, hàm này còn có thể gọi là *hàm chi phí* (*cost function*), *hàm mất mát* (*loss function*) hay *hàm sai số* (*error function*). Trong cuốn sách này, các thuật ngữ trên có thể được dùng thay thế nhau nhưng đều mang ý nghĩa tương đương, mặc dù một số tài liệu khác về ML sử dụng những thuật ngữ này với ý nghĩa khác nhau.

Ta thường kí hiệu giá trị của đối số để hàm số đạt cực đại hoặc cực tiểu bằng dấu $*$. Ví dụ, ta có thể kí hiệu $\boldsymbol{x}^* = \arg\min f(\boldsymbol{x})$.

Chúng tôi ngầm định rằng bạn đọc đã quen thuộc với các phép tính vi tích phân, nhưng chúng tôi vẫn sẽ điểm qua một số khái niệm về các phép tính vi tích phân liên quan tới bài toán tối ưu hóa.

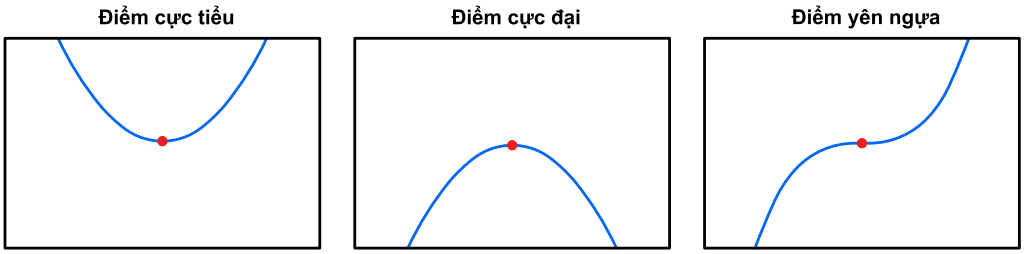
Giả sử, ta có hàm $y = f(x)$ với x, y đều là số thực. *Đạo hàm* (derivative) của hàm này được kí hiệu là $f'(x)$ hay $\frac{dy}{dx}$. Đạo hàm $f'(x)$ cho ta biết độ dốc của hàm $f(x)$ tại điểm x. Nói cách khác, nó cho ta biết sự thay đổi của hàm tương ứng với sự thay đổi rất nhỏ của biến: $f(x + \epsilon) \approx f(x) + \epsilon f'(x)$

Do đó, đạo hàm có ích trong việc tối thiểu hóa một hàm vì nó cho ta biết nên thay đổi x thế nào để cải thiện hàm tối ưu một lượng nhỏ. Ví dụ, ta biết rằng với ϵ đủ nhỏ thì $f(x - \epsilon \operatorname{sign}(f'(x)))$ sẽ nhỏ hơn $f(x)$. Từ đây, ta có thể giảm giá trị hàm $f(x)$ bằng cách tăng hoặc giảm x một lượng nhỏ về hướng ngược với hướng của của đạo hàm. Kỹ thuật này được gọi là *gradient descent* (gradient descent). Hình 4.1 minh họa cho kỹ thuật này.



> Hình 4.1: gradient descent. Một minh họa về cách Gradient Descent sử dụng các đạo hàm của một hàm để đi theo dốc của hàm tiến về một điểm cực tiểu.

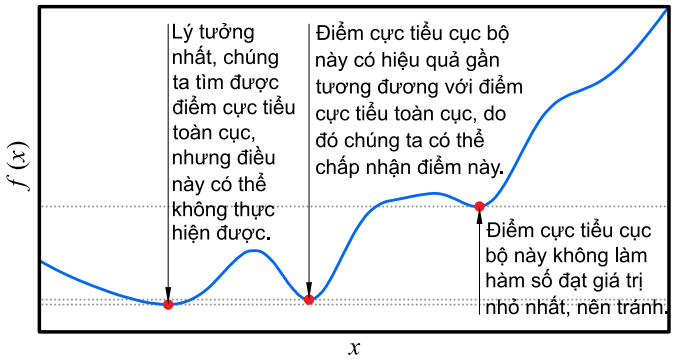
Khi $f'(x) = 0$, ta không biết được nên thay đổi x như thế nào nếu chỉ dựa vào đạo hàm. Ta gọi các điểm mà $f'(x) = 0$ là các *điểm tới hạn* (*critical point*) hoặc *điểm dừng* (*stationary point*). Điểm *cực tiểu cục bộ* (*local minimum*) là điểm mà giá trị $f(x)$ nhỏ hơn giá trị của hàm tại tất cả các điểm lân cận, do đó, ta không thể giảm giá trị của hàm $f(x)$ bằng cách thay đổi x một lượng vô cùng nhỏ như trước. Tương tự, điểm *cực đại cục bộ* (*local maximum*) là điểm mà $f(x)$ có giá trị lớn hơn giá trị của hàm tại tất cả các điểm lân cận, do đó, ta không thể tăng giá trị của hàm $f(x)$ bằng cách thay đổi x một lượng vô cùng nhỏ như trước. Có một vài điểm tới hạn không phải là cực đại hay cực tiểu. Các điểm đó ta gọi là các *điểm yên ngựa* (*saddle point*). Hình 4.2 minh họa từng loại điểm tới hạn.



> Hình 4.2 Các loại điểm tới hạn. Ví dụ về ba loại điểm tới hạn trong không gian một chiều. Điểm tới hạn là điểm có độ dốc bằng 0. Một điểm như vậy có thể là một điểm cực tiểu cục bộ, giá trị hàm số tại điểm này có giá trị thấp hơn tại các điểm lân cận; hoặc một điểm cực đại cục bộ, giá trị hàm số tại điểm này có giá trị cao hơn tại các điểm lân cận; hoặc cũng có thể là một điểm yên ngựa, giá trị hàm số tại điểm này có giá trị cao hơn tại một số điểm lân cận nhưng lại thấp hơn tại một số các điểm lân cận khác.

Một điểm mà hàm $f(x)$ đạt giá trị nhỏ nhất được gọi là *cực tiểu toàn cục* (*global optimum*) của hàm. Một hàm số có thể có một hoặc nhiều điểm cực tiểu toàn cục. Hàm cũng có thể có những điểm cực tiểu cục bộ mà không phải cực tiểu toàn cục. Trong phạm vi của DL, ta thường gặp bài toán tối ưu mà ở đó hàm có nhiều cực tiểu cục bộ nhưng không phải toàn cục, hoặc hàm có nhiều điểm yên ngựa được bao quanh bởi các vùng phẳng. Tất cả những vấn đề này làm cho việc tối ưu hóa trở nên khó khăn, đặc biệt là khi các biến của hàm số là biến nhiều chiều. Do đó, ta thường chỉ cần tìm một giá trị mà ở đó hàm f có giá trị

nhỏ nhưng không nhất thiết phải là cực tiểu theo đúng nghĩa. Xem một ví dụ minh họa trong hình 4.3.



> Hình 4.3. Xấp xỉ giá trị cực tiểu. Các thuật toán tối ưu có thể không tìm được cực tiểu toàn cục khi tồn tại nhiều cực tiểu cục bộ hoặc các miền phẳng (plateaus). Trong phạm vi của DL, ta cũng thường chấp nhận những nghiệm như vậy ngay cả khi nó không thực sự là các điểm cực tiểu, miễn là nó tương ứng với các giá trị rất nhỏ của hàm chi phí.

ta thường cực tiểu hóa các hàm đa biến: $f : \mathbb{R}^n \to \mathbb{R}$. Để khái niệm "tối ưu hóa" trở nên hợp lý, đầu ra của hàm phải là một đại lượng vô hướng duy nhất.

Với các hàm đa biến, ta phải sử dụng khái niệm *đạo hàm riêng* (*partial derivative*). Đạo hàm riêng $\frac{\partial}{\partial x_i} f(x)$ là đại lượng đo sự biến thiên của hàm số f khi x_i tăng tại điểm $\boldsymbol{x}$. *Gradient* là khái niệm tổng quát của đạo hàm cho một vector: gradient của hàm số f là vector chứa tất cả đạo hàm riêng của hàm số, kí hiệu: $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$. Phần tử i của gradient biểu diễn đạo hàm riêng phần của f tại x_i. Trong không gian đa chiều, những điểm tới hạn là những điểm mà tại đó mọi phần tử của gradient đều bằng 0.

Đạo hàm theo hướng (*directional derivative*) theo phương $\boldsymbol{u}$ (một vector đơn vị) là độ dốc của hàm số f theo phương u. Nói cách khác, đạo hàm theo hướng là đạo hàm của hàm số $f(\boldsymbol{x} + \alpha \boldsymbol{u})$ đối với α tại $\alpha = 0$. Sử dụng quy tắc chuỗi, ta có thể nhận ra $\frac{\partial}{\partial \alpha} f(\boldsymbol{x} + \alpha \boldsymbol{u})$ bằng với $\boldsymbol{u}^\intercal \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ khi $\alpha = 0$.

Để tìm giá trị nhỏ nhất của f, ta cần tìm hướng sao cho hàm f giảm nhanh nhất. Ta có thể thực hiện điều này bằng cách sử dụng đạo hàm theo hướng:

$$\min_{\boldsymbol{u}, \boldsymbol{u}^\intercal \boldsymbol{u} = 1} \boldsymbol{u}^\intercal \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{4.3}$$

$$= \min_{\boldsymbol{u}, \boldsymbol{u}^\intercal \boldsymbol{u} = 1} ||\boldsymbol{u}||_2 ||\nabla_{\boldsymbol{x}} f(\boldsymbol{x})||_2 \cos\theta \tag{4.4}$$

với θ là góc giữa $\boldsymbol{u}$ và gradient. Thay $||\boldsymbol{u}||_2 = 1$ và bỏ qua các đại lượng không phụ thuộc vào $\boldsymbol{u}$, bài toán trở thành tìm $\min_{\boldsymbol{u}} \cos\theta$. Hàm số này đạt giá trị cực

tiểu khi u có hướng ngược lại so với gradient. Nói cách khác, gradient chỉ theo hướng đi lên và phương nghịch của gradient chỉ theo hướng đi xuống. Ta có thể giảm f bằng cách di chuyển theo phương nghịch của gradient. Phương pháp này được gọi là *gradient dốc nhất* (steepest gradient) hoặc *gradient descent* (gradient descent).

gradient descent đưa ra một điểm mới:

$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \qquad (4.5)$$

trong đó ϵ là *learning rate*, một đại lượng vô hướng dương xác định kích thước của bước nhảy. Ta có thể lựa chọn ϵ theo nhiều cách khác nhau. Một cách phổ biến nhất cho ϵ nhận giá trị là hằng số rất nhỏ, và đôi khi ta có thể tìm được bước nhảy khiến cho đạo hàm theo hướng bị triệt tiêu. Một cách tiếp cận khác là tìm giá trị của $f(\boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}))$ với một số giá trị khác nhau của ϵ và lựa chọn giá trị dẫn đến giá trị nhỏ nhất của hàm mục tiêu. Chiến lược này còn được gọi là *dò đường* (line search).

Gradient descent hội tụ khi mọi phần tử của gradient bằng 0 (hoặc, trong thực tế, rất gần 0). Trong một vài trường hợp, ta có thể tránh việc chạy thuật toán theo vòng lặp mà nhảy trực tiếp đến điểm tới hạn bằng cách giải phương trình $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = 0$ để tìm $\boldsymbol{x}$.

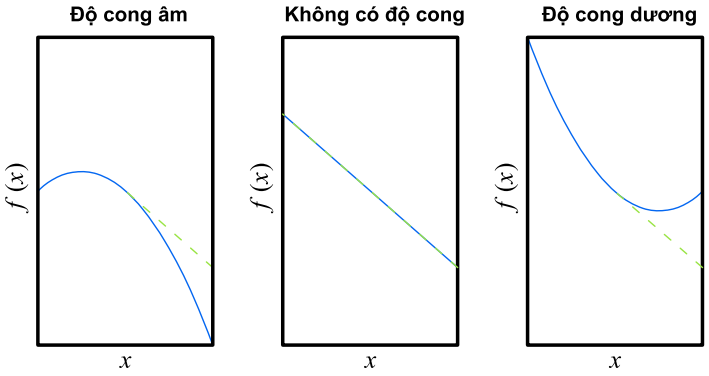
Mặc dù gradient descent bị giới hạn để tối ưu hóa trong các không gian liên tục, ý tưởng chung của việc lặp lại các bước di chuyển nhỏ (mà có thể coi là bước di chuyển nhỏ tốt nhất) để hướng tới một kết quả tốt hơn có thể được khái quát hóa cho các không gian rời rạc. Quá trình tăng một hàm mục tiêu của các tham số rời rạc được gọi là quá trình *leo đồi* (hill climbing) [Russel and Norvig, 2003].

4.3.1 Ma trận Jacobi và ma trận Hesse

Đôi khi ta cần tìm tất cả các đạo hàm riêng của một hàm có đầu vào và đầu ra đều là các vector. Ma trận chứa tất cả các đạo hàm riêng như vậy được gọi là *ma trận Jacobi* (*Jacobian matrix*). Cụ thể, nếu ta có một hàm $f : \mathbb{R}^m \to \mathbb{R}^n$, thì ma trận Jacobi $\boldsymbol{J} \in R^{n \times m}$ của f được xác định như sau $\boldsymbol{J}_{i,j} = \frac{\partial}{\partial x_j} f(\boldsymbol{x})_i$.

Đôi khi ta cũng quan tâm đến một đạo hàm của một đạo hàm. Ta gọi nó là *đạo hàm bậc hai* (*second derivative*). Ví dụ, đối với một hàm $f : \mathbb{R}^n \to \mathbb{R}$, đạo hàm đối với x_i của đạo hàm của f đối với x_j được ký hiệu là $\frac{\partial^2}{\partial x_i \partial x_j} f(x)$. Trong trường hợp một chiều, ta có thể kí hiệu $\frac{d^2}{dx^2} f(x)$ bởi $f''(x)$. Đạo hàm bậc hai cho ta biết đạo

hàm bậc một sẽ biến thiên như thế nào khi ta thay đổi đầu vào. Đây là một tính năng quan trọng bởi nó cho ta biết liệu một bước cập nhật gradient có cải thiện hàm số như ta mong đợi khi chỉ dựa trên gradient hay không. Ta có thể coi đạo hàm bậc hai như là phép đo *độ cong* (*curvature*). Giả sử ta có hàm bậc hai (nhiều hàm phát sinh trong thực nghiệm không phải là bậc hai nhưng có thể xấp xỉ tốt bởi dạng bậc hai, ít nhất là một cách cục bộ). Nếu một hàm như vậy có đạo hàm bậc hai bằng 0 thì nó không có độ cong. Nó hoàn toàn là một đường thẳng, và chỉ cần sử dụng gradient là ta có thể dự đoán được giá trị nó. Nếu gradient là 1, thì ta có thể thực hiện một bước cập nhật có kích thước ϵ dọc theo vector đối của gradient, và hàm chi phí sẽ giảm cùng lượng ϵ. Nếu đạo hàm bậc hai là số âm, hàm sẽ cong xuống dưới, do đó hàm chi phí thực tế sẽ giảm nhiều hơn ϵ. Cuối cùng, nếu đạo hàm bậc hai là dương, thì đường cong của hàm hướng lên trên, do đó, hàm chi phí có thể giảm ít hơn ϵ. Xem hình 4.4 để thấy các dạng cong khác nhau ảnh hưởng đến mối quan hệ giữa giá trị dự đoán của hàm chi phí sử dụng gradient với giá trị thực của hàm như thế nào.



> Hình 4.4: Đạo hàm bậc hai mô tả độ cong của hàm. Ở đây chúng tôi biểu diễn các hàm bậc hai với các độ cong khác nhau. Đường nét đứt biểu thị giá trị của hàm chi phí mà ta kỳ vọng khi thực hiện một bước giảm gradient, trong trường hợp ta chỉ biết thông tin về gradient. Với độ cong có giá trị âm, hàm chi phí thực sự giảm nhanh hơn lượng giảm gradient dự đoán. Với độ cong bằng 0, gradient dự đoán mức giảm chính xác. Với độ cong dương, hàm giảm chậm hơn dự kiến và cuối cùng bắt đầu tăng, vì vậy các bước nhảy quá lớn có thể vô tình làm tăng hàm số.

Khi hàm có biến số nhiều chiều, sẽ có nhiều đạo hàm bậc hai. Ta có thể thu thập các đạo hàm này lại với nhau thành một ma trận gọi là *ma trận Hesse* (Hessian matrix). Ma trận Hesse $\boldsymbol{H}(f)(\boldsymbol{x})$ được xác định như sau:

$$\boldsymbol{H}(f)(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(x) \qquad (4.6)$$

Nói cách khác, ma trận Hesse chính là ma trận Jacobi của gradient.

Tại bất kỳ điểm nào mà các đạo hàm riêng bậc hai là liên tục, thì các toán tử vi phân có tính chất giao hoán; có nghĩa là, thứ tự của chúng có thể được hoán đổi:

$$\frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x}) = \frac{\partial^2}{\partial x_j \partial x_i} f(\boldsymbol{x}) \qquad (4.7)$$

Điều này dẫn tới $\boldsymbol{H}_{i,j} = \boldsymbol{H}_{j,i}$, do đó ma trận Hesse là đối xứng tại những điểm như vậy.

Hầu hết các hàm số ta gặp phải trong DL đều có Hesse đối xứng tại hầu hết mọi nơi (almost everywhere). Bởi ma trận Hessie là thực và đối xứng, nên ta có thể phân tách nó thành một tập các giá trị riêng thực và một cơ sở trực giao của các *vector riêng* (eigenvector). Đạo hàm bậc hai theo một hướng cụ thể biểu diễn bởi một vector đơn vị $\boldsymbol{d}$ được cho bởi $\boldsymbol{d}^\intercal \boldsymbol{H} \boldsymbol{d}$. Khi $\boldsymbol{d}$ là một vector riêng của $\boldsymbol{H}$, đạo hàm bậc hai theo hướng đó chính là giá trị riêng tương ứng. Đối với các hướng khác của $\boldsymbol{d}$, đạo hàm bậc hai theo hướng là một trung bình có trọng số của tất cả các trị riêng, với các trọng số nằm giữa 0 và 1, và các vector riêng có góc nhỏ hơn với $\boldsymbol{d}$ nhận trọng số lớn hơn. Trị riêng lớn nhất chính là giá trị lớn nhất của đạo hàm bậc hai (theo hướng) và trị riêng nhỏ nhất chính là giá trị nhỏ nhất của đạo hàm bậc hai.

Đạo hàm bậc hai (theo hướng) cho ta biết ta có thể kỳ vọng một bước gradient descent thực hiện tốt như thế nào. Ta có thể tạo một xấp xỉ bởi chuỗi Taylor bậc hai của hàm $f(\boldsymbol{x})$ xung quanh điểm hiện tại $\boldsymbol{x}^{(0)}$:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\intercal \boldsymbol{g} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\intercal \boldsymbol{H} (\boldsymbol{x} - \boldsymbol{x}^{(0)}) \qquad (4.8)$$

trong đó $\boldsymbol{g}$ là gradient và $\boldsymbol{H}$ là ma trận Hesse tại $\boldsymbol{x}^{(0)}$. Nếu ta sử dụng tốc độ học ϵ, thì điểm mới $\boldsymbol{x}$ sẽ là $\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}$. Thay vào công thức xấp xỉ trên, ta thu được

$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^\intercal \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^\intercal \boldsymbol{H} \boldsymbol{g}. \qquad (4.9)$$

Ba số hạng trong công thức trên theo thứ tự là: giá trị ban đầu của hàm $f(\boldsymbol{x}^{(0)})$, mức giảm kỳ vọng do độ dốc của hàm và hiệu chỉnh ta đưa vào để tính đến độ cong của hàm. Khi đại lượng cuối cùng quá lớn, bước gradient descent thực tế có thể di chuyển lên trên. Khi $\boldsymbol{g}^\intercal \boldsymbol{H} \boldsymbol{g}$ là số 0 hoặc số âm, xấp xỉ chuỗi Taylor dự đoán rằng tăng ϵ mãi mãi sẽ giảm f mãi mãi. Trong thực tế, chuỗi Taylor không

còn là một xấp xỉ tốt khi ϵ lớn, vì vậy người ta phải sử dụng một vài kĩ thuật dựa trên kinh nghiệm để chọn ϵ trong trường hợp này. Khi $\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}$ là một số dương, giải phương trình tìm giá trị bước nhảy tối ưu ϵ để giảm chuỗi xấp xỉ Taylor của hàm chi phí một lượng lớn nhất, ta có

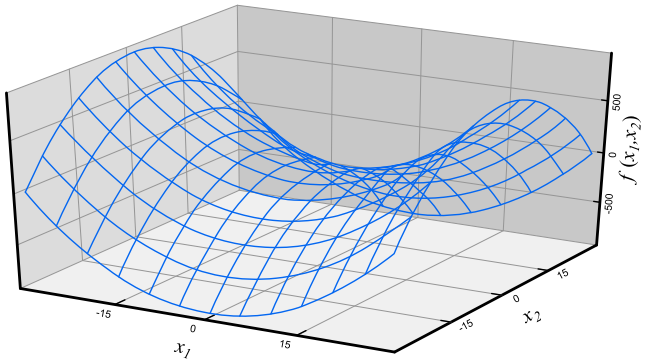
$$\epsilon^* = \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}}. \qquad (4.10)$$

Trong trường hợp xấu nhất, khi $\boldsymbol{g}$ có cùng hướng với vector riêng của $\boldsymbol{H}$ tương ứng với giá trị riêng tối đa λ_{max}, thì độ lớn của bước nhảy tối ưu này chính là $\frac{1}{\lambda_{max}}$. Trong phạm vi mà hàm số ta đang cực tiểu hoá có thể được xấp xỉ tốt bởi hàm bậc hai, do đó, các giá trị riêng của ma trận Hesse sẽ xác định độ lớn của tốc độ học.

Đạo hàm bậc hai có thể được sử dụng để xác định một điểm tới hạn là cực đại cục bộ, cực tiểu cục bộ hay điểm yên ngựa. Nhớ lại rằng tại một điểm tới hạn, $f'(x) = 0$. Khi đạo hàm bậc hai $f''(x) > 0$, đạo hàm bậc một $f'(x)$ tăng khi x di chuyển về bên phải và giảm khi x di chuyển về bên trái. Điều này có nghĩa là $f'(x - \epsilon) < 0$ và $f'(x + \epsilon) > 0$ với ϵ đủ nhỏ. Nói cách khác, khi ta di chuyển sang phải thì độ dốc hướng lên về bên phải, và khi ta di chuyển sang trái thì độ dốc hướng lên về bên trái. Do đó, khi $f'(x) = 0$ và $f''(x) > 0$, ta có thể kết luận rằng x là một cực tiểu cục bộ. Tương tự, khi $f'(x) = 0$ và $f''(x) < 0$, ta có thể kết luận rằng x là một cực đại cục bộ. Suy luận này được gọi là *phép thử đạo hàm bậc hai* (second derivative test). Khi $f''(x) = 0$, ta sẽ không thể kết luận gì về đặc điểm của x. Trong trường hợp này, x có thể là một điểm yên ngựa hay là một phần của một vùng phẳng.

Trong không gian nhiều chiều, ta cần khảo sát toàn bộ các đạo hàm bậc hai của hàm. Sử dụng phép phân tích riêng của ma trận Hesse, ta có thể tổng quát hóa phép thử đạo hàm bậc hai cho trường hợp nhiều chiều. Tại một điểm tới hạn, tức $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = 0$, ta có thể khảo sát các trị riêng của ma trận Hesse để xác định điểm tới hạn là cực đại cục bộ, cực tiểu cục bộ hay điểm yên ngựa. Nếu ma trận Hesse là ma trận xác định dương (tất cả các trị riêng đều dương), điểm tới hạn đang xét là một cực tiểu cục bộ. Phát biểu này đúng nhờ vào quan sát sau: tất cả các đạo hàm bậc hai theo bất kì hướng nào đều phải dương, và từ đây tham chiếu đến trường hợp đạo hàm bậc hai đơn biến. Tương tự, nếu ma trận Hesse là ma trận xác định âm (tất cả các trị riêng đều âm) thì điểm tới hạn đang xét là một cực đại cục bộ. Trong không gian nhiều chiều, ta có thể tìm thấy những bằng chứng tích cực về điểm yên ngựa trong một số trường hợp. Nếu có ít nhất

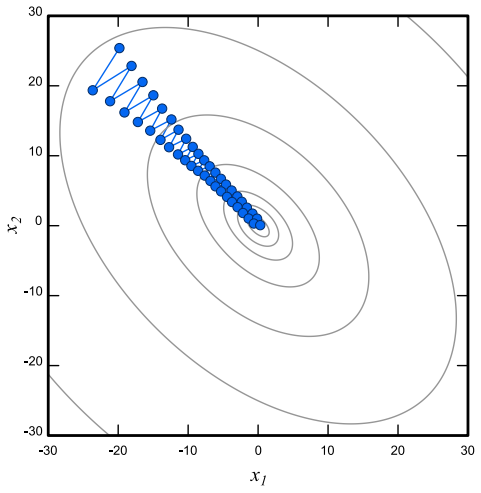
một trị riêng là âm và có ít nhất một trị riêng là dương, ta biết được rằng $\boldsymbol{x}$ là một cực đại cục bộ trên một mặt cắt (cross section) của f nhưng lại là một cực tiểu cục bộ trên một mặt cắt khác. Xem một ví dụ minh họa cho trường hợp này trong hình 4.5. Cuối cùng, phép thử đạo hàm bậc hai của hàm nhiều biến cũng có thể không cho ta một nhận định chắc chắn về $\boldsymbol{x}$, cũng giống như trong trường hợp đơn biến. Phép thử sẽ không cho ta kết luận gì nếu tất cả các trị riêng khác không cùng dấu (âm hoặc dương) nhưng có ít nhất một trị riêng bằng không. Nguyên nhân là bởi ta không thể kết luận được gì từ phép thử đạo hàm bậc hai đơn biến trong mặt cắt tương ứng với trị riêng bằng không.



> Hình 4.5: Một điểm yên ngựa gồm cả độ cong âm và dương. Phương trình trong ví dụ này là $f(\boldsymbol{x}) = x_1^2 - x_2^2$. Theo trục tương ứng với x_1, đồ thị hàm số cong lên. Trục này là một vector riêng của ma trận Hesse có trị riêng dương. Theo trục tương ứng với x_2, đồ thị hàm số cong xuống. Trục theo hướng này là một vector riêng của ma trận Hesse với trị riêng âm. Cái tên "điểm yên ngựa" bắt nguồn từ hình dáng giống như yên ngựa của phương trình này. Đây là một ví dụ hết sức tinh tế về một phương trình với một điểm yên ngựa. Trong không gian nhiều chiều, một điểm không nhất thiết phải có trị riêng bằng 0 mới là điểm yên ngựa: Một điểm yên ngựa chỉ cần có cả trị riêng âm và trị riêng dương. Ta có thể coi một điểm yên ngựa có các trị riêng âm và dương là một cực đại cục bộ trong một mặt cắt nhưng nó lại là một cực tiểu cục bộ trong một mặt cắt khác.

Trong không gian nhiều chiều, tại một điểm sẽ có một đạo hàm bậc hai khác nhau cho mỗi chiều. Hệ số điều hòa của ma trận Hesse tại điểm này thể hiện mức độ khác nhau giữa các đạo hàm bậc hai. Nếu ma trận Hesse có hệ số điều hòa cao, Gradient Descent sẽ kém hiệu quả. Bởi theo một hướng nào đó, đạo

hàm tăng nhanh chóng, nhưng theo một hướng khác, nó tăng rất chậm. Gradient Descent không nhận thức được sự thay đổi này của đạo hàm, nên nó sẽ không biết rằng nên ưu tiên đi theo hướng mà đạo hàm âm nhiều hơn. Hệ số điều hòa cao còn làm cho việc chọn bước nhảy trở nên khó khăn. Bước nhảy phải đủ nhỏ để tránh đi qua giá trị cực tiểu và đi ngược lên hướng có độ cong dương. Điều này thường có nghĩa là, theo những hướng khác ít cong hơn, thì bước nhảy này lại là quá nhỏ để tạo ra những bước tiến đáng kể đến điểm tới hạn. Xem một ví dụ minh họa trong hình 4.6.



Hình 4.6: Thuật toán gradident descent không thể khai thác được thông tin về độ cong từ ma trận Hesse. Ở đây ta sử dụng gradient descent để tìm giá trị nhỏ nhất của hàm bậc hai $f(\boldsymbol{x})$, hàm này có hệ số điều hòa là 5. Điều này có nghĩa là hướng có độ cong lớn nhất sẽ cong gấp 5 lần hướng có độ cong nhỏ nhất. Trong trường hợp này, độ cong lớn nhất nằm ở hướng $[1, 1]^\intercal$ và độ cong nhỏ nhất nằm ở hướng $[1, -1]^\intercal$. Những đoạn màu đỏ thể hiện đường đi của Gradient Descent. Đồ thị này kéo dài và có những đỉnh nhọn trông giống như một hẻm núi dài vậy. Gradient Descent đã lãng phí thời gian để liên tiếp đi xuống những sườn núi bởi chúng có độ dốc lớn nhất. Nguyên nhân gây ra hiện tượng trên là bởi bước nhảy quá lớn và có khả năng sẽ vượt quá điểm thấp nhất cần tìm nên Gradient Descent đã phải điều chỉnh đường đi bằng cách đi theo sườn của đỉnh đối diện trong lần lặp tiếp theo. Trị riêng có giá trị lớn của ma trận Hesse tương ứng với vector riêng của hướng này có đạo hàm theo hướng tăng rất nhanh, vì thế một thuật toán tối ưu dựa trên ma trận Hesse có thể dự đoán rằng đi theo sườn dốc nhất chưa chắc đã là một hướng tìm kiếm tốt trong trường hợp này.

Vấn đề này có thể được giải quyết bằng cách sử dụng thông tin từ ma trận Hesse để dẫn dắt quá trình tìm kiếm. Phương pháp đơn giản nhất được biết đến để thực hiện điều này là *phương pháp Newton*. Phương pháp Newton dựa trên khai triển chuỗi Taylor bậc hai để xấp xỉ $f(\boldsymbol{x})$ tại điểm gần với $\boldsymbol{x}^{(0)}$:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\intercal \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\intercal \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})(\boldsymbol{x} - \boldsymbol{x}^{(0)}) \qquad (4.11)$$

Nếu giải phương trình này, ta sẽ thu được điểm tới hạn sau:

$$\boldsymbol{x}^* = \boldsymbol{x}^{(0)} - \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})^{-1} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}) \qquad (4.12)$$

Nếu f là một hàm bậc hai xác định dương, ta sử dụng phương pháp Newton bằng cách áp dụng phương trình (4.12) một lần và có thể nhảy đến ngay điểm cực tiểu của hàm số. Nếu hàm f không phải là bậc hai nhưng trên cục bộ có thể xấp xỉ tốt bởi một hàm bậc hai xác định dương, thì phương pháp Newton được thực hiện bằng cách áp dụng phương trình (4.12) nhiều lần. Lặp lại việc cập nhật hàm xấp xỉ và nhảy đến cực tiểu của hàm xấp xỉ có thể dẫn đến điểm tới hạn nhanh hơn khá nhiều so với Gradient Descent. Đây là một tính chất hữu ích khi ta đã ở gần một cực tiểu cục bộ, nhưng cũng có thể là một tính chất có hại nếu ta ở gần điểm yên ngựa. Trong mục 8.2.3, ta sẽ chỉ ra phương pháp Newton chỉ phù hợp khi điểm tới hạn cận kề là một cực tiểu (tất cả các trị riêng của ma trận Hesse là dương), khi mà gradient descent không bị hút về phía các điểm yên ngựa nếu gradient không chỉ về phía chúng.

Các thuật toán tối ưu chỉ sử dụng gradient, ví dụ như gradient descent, được gọi là các *thuật toán tối ưu bậc nhất* (first-order optimization algorithm). Các thuật toán tối ưu sử dụng cả ma trận Hesse, ví dụ như phương pháp Newton, được gọi là các *thuật toán tối ưu bậc hai* (second-order optimization algorithm) [Nocedal and Wright, 2006].

Những thuật toán tối ưu được sử dụng trong hầu hết ngữ cảnh của cuốn sách này có thể áp dụng cho một phạm vi rộng các hàm số nhưng hầu như không có đảm bảo rằng chúng luôn hoạt động tốt. Những thuật toán DL thường có xu hướng thiếu sự đảm bảo bởi chúng sử dụng những họ hàm số hết sức phức tạp. Trong những lĩnh vực khác, hướng tiếp cập chủ yếu với bài toán tối ưu hóa là thiết kế những thuật toán tối ưu cho hữu hạn họ các phương trình.

Trong phạm vi của DL, đôi khi ta cũng đạt được một số tính đảm bảo bằng cách giới hạn phạm vi các hàm số phải là *liên tục Lipschitz (Lipschitz continuos) hoặc*

có đạo hàm Lipschitz liên tục. Một hàm số Lipschitz liên tục là một hàm f có tốc độ thay đổi bị giới hạn bởi một hằng số Lipschitz $\mathcal{L}$:

$$\forall \boldsymbol{x}, \forall \boldsymbol{y}, |f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq \mathcal{L}||\boldsymbol{x} - \boldsymbol{y}||_2 \qquad (4.13)$$

Tính chất này khá hữu dụng bởi nó cho phép ta định lượng được giả thiết một sự thay đổi nhỏ ở đầu vào do một thuật toán, như gradient descent chẳng hạn, gây ra sẽ dẫn đến một thay đổi nhỏ ở đầu ra. Tính liên tục Lipschitz cũng là một ràng buộc khá yếu, ta có thể làm cho nhiều bài toán tối ưu trong DL có tính chất liên tục Lipschitz với một vài kĩ thuật thay đối nhỏ.

Có lẽ lĩnh vực thành công nhất của bài toán tối ưu hoá chuyên dụng là *tối ưu lồi* (convex optimization). Các thuật toán tối ưu lồi có nhiều tính chất chứng minh được một cách chặt chẽ do các hàm mục tiêu có nhiều giới hạn mạnh hơn. Những thuật toán này chỉ có thể áp dụng cho những hàm lồi - những hàm số có ma trận Hesse là bán xác định dương tại mọi điểm. Những hàm này dễ tối ưu hơn vì chúng không có điểm yên ngựa, và những cực tiểu cục bộ đều là cực tiểu toàn cục. Tuy nhiên, hầu hết những bài toán trong DL đều khó có thể biểu diễn về dạng bài toán tối ưu lồi. Tối ưu lồi chỉ được sử dụng như một chương trình con của một số thuật toán DL. Các ý tưởng đến từ giải tích của những thuật toán tối ưu lồi có thể hữu ích cho việc chứng minh tính hội tụ của các thuật toán DL, tuy nhiên, nhìn chung, tối ưu lồi không có ý nghĩa quá quan trọng trong phạm vi của DL. Để tìm hiểu thêm về tối ưu lồi, bạn có thể đọc Boyd and Vandenberghe (2001) hoặc Rockafellar(1997).

4.4 Tối ưu có ràng buộc

Đôi khi, mục tiêu của ta không chỉ là tìm giá trị cực đại hoặc cực tiểu của hàm $f(\boldsymbol{x})$ trên toàn bộ tập các giá trị mà $\boldsymbol{x}$ có thể nhận. Mà thay vào đó, ta mong muốn tìm giá trị cực đại hoặc cực tiểu của $f(\boldsymbol{x})$ khi $\boldsymbol{x}$ nằm trong một số tập $\mathbb{S}$ nào đó. Vấn đề này được gọi là bài toán *tối ưu có ràng buộc* (constrained optimization). Các điểm $\boldsymbol{x}$ nằm trong tập $\mathbb{S}$ được gọi là các điểm *khả thi* (feasible) ngữ cảnh của bài toán tối ưu có ràng buộc.

Ta thường muốn tìm ra nghiệm nhỏ, theo một nghĩa nào đó. Cách tiếp cận thông thường trong những trường hợp như vậy là áp thêm ràng buộc về chuẩn (norm), chẳng hạn như $\|\boldsymbol{x}\| < 1$.

Một phương án đơn giản khác theo hướng tối ưu có ràng buộc là sửa đổi Gradient Descent theo hướng sử dụng các ràng buộc. Nếu ta dùng một bước nhảy ϵ nhỏ không đổi, ta có thể tạo ra các bước gradient descent, và chiếu kết quả ngược về tập $\mathbb{S}$. Nếu sử dụng kỹ thuật dò đường, ta chỉ có thể tìm kiếm trong phạm vi các bước nhảy ϵ tạo ra được các điểm $\boldsymbol{x}$ khả thi mới, hoặc ta có thể chiếu mỗi điểm trên đường đó đi ngược về vùng ràng buộc. Nếu được, ta có thể cải thiện phương pháp này bằng cách chiếu gradient vào không gian tiếp tuyến với vùng khả thi trước khi thực hiện bước trượt hoặc dò đường [Rosen, 1960].

Một cách tiếp cận phức tạp hơn là thiết kế một bài toán tối ưu không ràng buộc khác, sao cho đáp án của nó có thể được chuyển đổi thành đáp án của bài toán tối ưu có ràng buộc ban đầu. Chẳng hạn, nếu ta cực tiểu hóa $f(\boldsymbol{x})$ với $\boldsymbol{x} \in \mathbb{R}^2$ và $\boldsymbol{x}$ thỏa mãn ràng buộc rằng nó có một chuẩn L^2 xác định, thì thay vào đó, ta sẽ cực tiểu hóa $g(\theta) = f([\cos\theta, \sin\theta]^\intercal)$ theo θ, sau đó trả về giá trị $[\cos\theta, \sin\theta]$ như là đáp áp của bài toán gốc. Cách tiếp cận này đòi hỏi tính sáng tạo; việc chuyển đổi qua lại giữa các bài toán tối ưu phải được thiết kế riêng cho mỗi trường hợp.

Phương pháp *Karush-Kuhn-Tucker* (KKT) cung cấp một phương pháp giải quyết rất tổng quát cho bài toán tối ưu có ràng buộc. Trong phương pháp này, chúng tôi giới thiệu một hàm mới, gọi là *Lagrange tổng quát* (generalized Lagrangian) hay *hàm Lagrange tổng quát* (generalized Lagrange function)[1].

> [1] Phương pháp KKT tổng quát hóa phương pháp *nhân tử Lagrange* (Lagrange multiplier), chỉ cho phép đẳng thức ràng buộc mà không cho phép các bất đẳng thức ràng buộc.

Để định nghĩa hàm Lagrange, đầu tiên, ta cần mô tả $\mathbb{S}$ ở dạng các phương trình và bất phương trình. Ta cần một mô tả của $\mathbb{S}$ ở dạng gồm m hàm $g^{(i)}$ và n hàm $h^{(i)}$, sao cho $\mathbb{S} = \{\boldsymbol{x} | \forall i, g^{(i)}(\boldsymbol{x}) = 0 \text{ và } \forall j, h^{(j)}(\boldsymbol{x}) \leq 0\}$. Các phương trình liên quan tới $g^{(i)}$ được gọi là các *đẳng thức ràng buộc*, và các bất phương trình liên quan tới $h^{(i)}$ được gọi là các *bất đẳng thức ràng buộc*.

Ta đưa vào các biến mới λ_i và α_i cho mỗi ràng buộc; chúng được gọi là các *nhân tử KKT* (KKT multiplier). Hàm Lagrange tổng quát được định nghĩa như sau:

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x}) \qquad (4.14)$$

Tới đây, ta có thể giải bài toán cực tiểu hoá có ràng buộc bằng cách giải bài toán tối ưu không ràng buộc - bài toán Lagrange tổng quát. Miễn là tối thiểu có một

điểm khả thi tồn tại và $f(\boldsymbol{x})$ không có khả năng tiến tới giá trị ∞, thì

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) \qquad (4.15)$$

có cùng giá trị hàm mục tiêu tối ưu và cùng tập các điểm tối ưu $\boldsymbol{x}$ với

$$\min_{\boldsymbol{x} \in \mathbb{S}} f(\boldsymbol{x}) \qquad (4.16)$$

Tính chất này là bởi bất cứ khi nào các ràng buộc được thỏa mãn, thì

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x}) \qquad (4.17)$$

Ngược lại, bất cứ khi nào ràng buộc bị vi phạm,

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = \infty \qquad (4.18)$$

Những tính chất này đảm bảo rằng không có điểm bất khả thi nào có thể trở thành điểm tối ưu, và rằng giá trị tối ưu của hàm trong tập điểm khả thi là không đổi.

Để giải quyết bài toán cực đại có ràng buộc, ta có thể xây dựng hàm Lagrange tổng quát của $-f(\boldsymbol{x})$, từ đó ta có bài toán tối ưu sau:

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} -f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x}) \qquad (4.19)$$

Ta cũng có thể chuyển bài toán về dạng có bước ngoài cùng là tìm cực đại:

$$\max_{\boldsymbol{x}} \min_{\boldsymbol{\lambda}} \min_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) - \sum_j \alpha_j h^{(j)}(\boldsymbol{x}) \qquad (4.20)$$

Chú ý rằng dấu của số hạng trong đẳng thức ràng buộc không quan trọng. Dấu cộng hay trừ tùy thuộc vào cách ta đưa ra bài toán, bởi dấu của mỗi λ_i có thể được tự do lựa chọn trong quá trình tối ưu.

Các bất đẳng thức ràng buộc đặc biệt thú vị. Giả sử rằng một ràng buộc $h^{(i)}(\boldsymbol{x})$ là *có hiệu lực* (active) nếu $h^{(i)}(\boldsymbol{x}^*) = 0$. Nếu một ràng buộc là phi hiệu lực thì đáp án tìm được của bài toán khi sử dụng ràng buộc đó sẽ vẫn là một đáp án cục bộ khi ta loại bỏ ràng buộc đó ra khỏi bài toán. Cũng có khi một ràng buộc phi hiệu lực loại trừ những đáp án khác. Chẳng hạn, một bài toán tối ưu lồi với toàn bộ vùng các điểm tối ưu toàn cục (một vùng rộng, phẳng mà hàm có cùng giá trị tối ưu tại mọi điểm) có thể bị các ràng buộc loại ra một phần, hay một bài toán tối ưu không lồi có thể có các điểm dừng cục bộ tốt hơn bị loại bỏ bởi một ràng buộc phi hiệu lực tại thời điểm hội tụ. Tuy vậy, điểm mà ta tìm được sau khi hội tụ vẫn là một điểm dừng, bất kể ta có đưa vào các ràng buộc phi hiệu lực hay

không. Bởi vì nếu một phi hiệu lực $h^{(i)}$ có giá trị âm, thì lời giải cho $\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha})$ sẽ có $\alpha_i = 0$. Do đó, ta có nhận xét rằng đáp án tối ưu sẽ thoả mãn $\boldsymbol{\alpha} \odot \boldsymbol{h}(\boldsymbol{x}) = 0$. Nói cách khác, với mọi i, ta biết rằng ít nhất một trong số các ràng buộc $\alpha_i \geq 0$ hoặc $h^{(i)}(\boldsymbol{x}) \leq 0$ phải có hiệu lực trong đáp án tối ưu. Để dễ mường tượng hơn, ta có thể giả sử rằng, hoặc là đáp án nằm trên biên của bất đẳng thức và ta phải sử dụng nhân tử KKT để tác động tính chất này của đáp án lên biến $\boldsymbol{x}$; hoặc là bất đẳng thức không có ảnh hưởng gì lên đáp án và ta biểu diễn điều này bằng cách thiết lập giá trị 0 cho các nhân tử KKT.

Có một tập các điều kiện đơn giản, mô tả các tính chất mà điểm tối ưu của các bài toán tối ưu có ràng buộc phải thỏa mãn. Các tính chất này được gọi là các điều kiện Karush-Kuhn-Tucker (KKT) [Karush, 1939, Kuhn and Tucker, 1951]. Chúng là các điều kiện cần, nhưng không phải luôn là điều kiện đủ để một điểm là tối ưu. Các điều kiện gồm:

- Gradient của hàm Lagrange tổng quát bằng 0.
- Tất cả ràng buộc trên cả $\boldsymbol{x}$ lẫn các nhân tử KKT phải được thỏa mãn.
- Các bất đẳng thức ràng buộc có tính "lỏng lẻo bù trừ" (complementary slackness): $\boldsymbol{\alpha} \odot \boldsymbol{h}(\boldsymbol{x}) = 0$.

Để hiểu rõ hơn về phương pháp KKT, tìm đọc [Nocedal and Wright, 2006].

4.5 Ví dụ: Bình phương cực tiểu tuyến tính

Giả sử ta muốn tìm giá trị của $\boldsymbol{x}$ sao cho hàm số sau đạt giá trị nhỏ nhất:

$$f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 \qquad (4.21)$$

Các thuật toán đại số tuyến tính chuyên dụng có thể giải bài toán này một cách hiệu quả; tuy nhiên, ta cũng có thể tìm hiểu cách giải bằng phương pháp tối ưu dựa trên gradient để coi đây như là một ví dụ đơn giản minh họa cách các kĩ thuật này hoạt động.

Trước tiên, ta cần tính gradient:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \boldsymbol{A}^\intercal(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{A}^\intercal \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\intercal \boldsymbol{b} \qquad (4.22)$$

Tiếp đó ta tính gradient descent này theo hướng xuống theo từng bước nhỏ. Xem chi tiết trong thuật toán 4.1.

Thuật toán 4.1: Một thuật toán cực tiểu hóa hàm $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}\|_2^2$ theo $\boldsymbol{x}$ bằng phương pháp gradient descent, bắt đầu từ một giá trị $\boldsymbol{x}$ tùy ý.

> Chọn bước nhảy (ϵ) và sai số cho phép (δ) có giá trị dương, độ lớn nhỏ.
>
> **while** $\|\boldsymbol{A}^\intercal\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\intercal\boldsymbol{b}\|_2 > \delta$ **do**
>
> $\quad \boldsymbol{x} \leftarrow \boldsymbol{x} - \epsilon(\boldsymbol{A}^\intercal\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\intercal\boldsymbol{b})$
>
> **end while**

Bài toán này cũng có thể giải được bằng phương pháp Newton. Trong trường hợp này, bởi vì hàm số đúng là hàm bậc hai, phương pháp xấp xỉ hàm bậc hai sử dụng phương pháp Newton là chính xác, và thuật toán sẽ hội tụ về điểm cực tiểu toàn cục trong vòng một bước.

Tới đây, giả sử ta muốn cực tiểu một hàm số tương tự, nhưng bổ sung thêm ràng buộc $\boldsymbol{x}^\intercal\boldsymbol{x} \leq 1$. Để giải quyết bài toán này, ta đưa vào hàm Lagrange:

$$L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda(\boldsymbol{x}^\intercal\boldsymbol{x} - 1). \qquad (4.23)$$

Tiếp theo ta giải bài toán

$$\min_x \max_{\lambda, \lambda \geq 0} L(\boldsymbol{x}, \lambda) \qquad (4.24)$$

Nghiệm của bài toán có chuẩn nhỏ nhất cho bài toán bình phương cực tiểu không ràng buộc có thể tìm ra bằng phương pháp giả nghịch đảo Moore-Penrose: $\boldsymbol{x} = \boldsymbol{A}^+\boldsymbol{b}$. Nếu điểm này khả thi, thì nó cũng là đáp án của bài toán có ràng buộc. Ngược lại, ta phải tìm đáp án trong đó ràng buộc về chuẩn là có hiệu lực. Lấy đạo hàm hàm Lagrange theo $\boldsymbol{x}$, ta được phương trình:

$$\boldsymbol{A}^\intercal\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\intercal\boldsymbol{b} + 2\lambda\boldsymbol{x} = 0 \qquad (4.25)$$

Như vậy nghiệm của bài toán sẽ có dạng:

$$\boldsymbol{x} = (\boldsymbol{A}^\intercal\boldsymbol{A} + 2\lambda\boldsymbol{I})^{-1}\boldsymbol{A}^\intercal\boldsymbol{b} \qquad (4.26)$$

Độ lớn của λ phải được chọn sao cho kết quả bài toán tuân theo ràng buộc về chuẩn. Ta có thể tìm giá trị này bằng cách áp dụng phương pháp leo dốc gradient (gradient ascent) với λ. Để làm được điều đó, ta thấy rằng

$$\frac{\partial}{\partial \lambda} L(\boldsymbol{x}, \lambda) = \boldsymbol{x}^\intercal \boldsymbol{x} - 1 \qquad (4.27)$$

Khi chuẩn của $\boldsymbol{x}$ vượt quá 1, đạo hàm trên có giá trị dương, do đó để men theo đạo hàm "leo dốc" và tăng giá trị hàm Lagrange theo biến λ, ta phải tăng λ. Bởi hệ số λ dùng để phạt $\boldsymbol{x}^\intercal \boldsymbol{x}$ tăng lên, nên việc giải phương trình tuyến tính (4.25) tìm $\boldsymbol{x}$ sẽ cho ra đáp án với chuẩn nhỏ hơn. Quá trình giải phương trình tuyến tính và điều chỉnh λ diễn ra liên tục cho đến khi $\boldsymbol{x}$ có chuẩn đúng và đạo hàm trên λ bằng 0.

Đến đây, chúng tôi đã kết thúc sơ bộ phần toán học mà ta cần để phát triển các thuật toán ML. Ta đã sẵn sàng để xây dựng và phân tích một số hệ thống tự học đầy đủ.