Chương 6

Mạng sâu lan truyền thẳng

Mạng sâu lan truyền thẳng, thường được gọi là **mạng nơron lan truyền thẳng** hoặc **nhận thức đa lớp** (multilayer perceptrons, MLPs), là mô hình điển hình của học sâu. Mục tiêu của một mạng lan truyền thẳng là xấp xỉ một hàm f^* nào đó. Ví dụ, đối với một bộ phân loại, $y = f^*(x)$ ánh xạ đầu vào $\boldsymbol{x}$ tới một danh mục y. Một mạng lan truyền tiến định nghĩa một ánh xạ $\boldsymbol{y} = f\left(\boldsymbol{x}; \boldsymbol{\theta}\right)$ và học giá trị của các tham số $\boldsymbol{\theta}$ để có được sự xấp xỉ hàm tốt nhất.

Các mô hình này được gọi là **lan truyền thẳng** vì thông tin đi qua hàm đang được đánh giá từ $\boldsymbol{x}$, qua các phép tính trung gian dùng để định nghĩa f, và cuối cùng đến đầu ra $\boldsymbol{y}$. Không có kết nối **phản hồi** trong đó các đầu ra của mô hình được đưa lại vào chính nó. Khi mạng nơron lan truyền thẳng được mở rộng để bao gồm các kết nối phản hồi, chúng được gọi là **mạng nơron hồi quy**, được giới thiệu trong Chương 10.

Các mạng nơron lan truyền thẳng có tầm quan trọng rất lớn đối với những người thực hành học máy. Chúng là nền tảng của nhiều ứng dụng thương mại quan trọng. Chẳng hạn, các mạng tích chập được sử dụng để nhận diện đối tượng từ ảnh là một loại mạng lan truyền thẳng đặc biệt. Mạng lan truyền thẳng là một khái niệm

bước đệm hướng tới mạng hồi quy, loại mạng hỗ trợ nhiều ứng dụng ngôn ngữ tự nhiên.

Mạng nơron lan truyền thẳng được gọi là **mạng** vì chúng thường được biểu diễn bằng cách kết hợp nhiều hàm khác nhau lại với nhau. Mô hình được liên kết với một đồ thị có hướng và không chu trình mô tả cách các hàm được kết hợp. Ví dụ, ta có thể có ba hàm $f^{(1)}$, $f^{(2)}$, và $f^{(3)}$ được kết nối theo chuỗi để tạo thành $f(\boldsymbol{x}) = f^{(3)}\left(f^{(2)}\left(f^{(1)}(\boldsymbol{x})\right)\right)$. Các cấu trúc chuỗi này là loại cấu trúc phổ biến nhất của mạng nơron. Trong trường hợp này, $f^{(1)}$ được gọi là **lớp thứ nhất** của mạng, $f^{(2)}$ là **lớp thứ hai**, v.v. Độ dài tổng thể của chuỗi xác định **độ sâu** của mô hình, từ đó xuất hiện thuật ngữ "học sâu" (deep learning). Lớp cuối cùng của một mạng lan truyền thẳng được gọi là **lớp đầu ra**. Trong quá trình huấn luyện mạng nơron, ta điều chỉnh $f(\boldsymbol{x})$ để khớp với $f^*(\boldsymbol{x})$. Dữ liệu huấn luyện cung cấp cho chúng ta các ví dụ xấp xỉ và có nhiễu của $f^*(\boldsymbol{x})$ được đánh giá tại các điểm huấn luyện khác nhau. Mỗi ví dụ $\boldsymbol{x}$ đi kèm với một nhãn $y \approx f^*(\boldsymbol{x})$. Các ví dụ huấn luyện xác định trực tiếp những gì lớp đầu ra phải làm tại mỗi điểm $\boldsymbol{x}$; nó phải tạo ra một giá trị gần với y. Hành vi của các lớp khác không được dữ liệu huấn luyện xác định trực tiếp. Thuật toán học phải quyết định cách sử dụng các lớp này để tạo ra đầu ra mong muốn, nhưng dữ liệu huấn luyện không chỉ rõ từng lớp riêng lẻ phải làm gì. Thay vào đó, thuật toán học phải quyết định cách sử dụng các lớp này để triển khai một phép xấp xỉ của f^* tốt nhất. Vì dữ liệu huấn luyện không chỉ ra đầu ra mong muốn cho mỗi lớp này, nên các lớp này được gọi là **lớp ẩn**.

Cuối cùng, các mạng này được gọi là *nơron* vì chúng được lấy cảm hứng một cách không chính xác từ khoa học thần kinh. Mỗi lớp ẩn của mạng thường có giá trị dạng véctơ. Số chiều của các lớp ẩn này xác định **độ rộng** của mô hình. Mỗi phần tử của véctơ có thể được hiểu là đảm nhận một vai trò tương tự như một nơron. Thay vì nghĩ về lớp như là một hàm ánh xạ từ véctơ sang véctơ, ta cũng có thể nghĩ về lớp này như bao gồm nhiều **đơn vị** hoạt động song song, mỗi đơn vị biểu diễn một hàm ánh xạ từ véctơ sang đại lượng vô hướng. Mỗi đơn vị giống như một nơron theo nghĩa là nó nhận đầu vào từ nhiều đơn vị khác và tính toán giá trị kích hoạt riêng của mình. Ý tưởng sử dụng nhiều lớp biểu diễn giá trị véctơ được lấy từ khoa học thần kinh. Việc lựa chọn các hàm $f^{(i)}(\boldsymbol{x})$ dùng để tính toán các biểu diễn này cũng được định hướng không chính xác từ các quan sát khoa học thần kinh về các hàm mà các nơron sinh học thực hiện. Tuy nhiên, nghiên cứu mạng nơron hiện đại được dẫn dắt bởi nhiều ngành toán học và kỹ thuật, và mục tiêu của mạng nơron không phải là mô phỏng hoàn hảo bộ não. Tốt nhất là ta nên nghĩ về các mạng lan truyền thẳng như là các thuật toán xấp xỉ hàm được thiết kế để đạt được

sự tổng quát hóa thống kê, thỉnh thoảng rút ra một số hiểu biết từ những gì chúng ta biết về bộ não, hơn là coi chúng như những mô hình chức năng của não.

Một cách để hiểu về các mạng lan truyền thẳng là bắt đầu với các mô hình tuyến tính và xem xét cách khắc phục các hạn chế của chúng. Các mô hình tuyến tính, chẳng hạn như hồi quy logistic và hồi quy tuyến tính, rất hấp dẫn vì chúng có thể được khớp một cách hiệu quả và đáng tin cậy, hoặc là dưới dạng giải đóng hoặc bằng tối ưu lồi. Tuy nhiên, các mô hình tuyến tính có nhược điểm rõ ràng là năng lực của mô hình bị giới hạn ở các hàm tuyến tính, vì vậy mô hình không thể hiểu được sự tương tác giữa bất kỳ hai biến đầu vào nào.

Để mở rộng các mô hình tuyến tính nhằm biểu diễn các hàm phi tuyến của $\boldsymbol{x}$, ta có thể áp dụng mô hình tuyến tính không phải cho bản thân $\boldsymbol{x}$ mà cho một đầu vào đã được biến đổi $\phi(\boldsymbol{x})$, trong đó ϕ là một biến đổi phi tuyến. Một cách tương đương, ta có thể áp dụng thủ thuật kernel được mô tả trong Mục 5.7.2, để có được một thuật toán học phi tuyến dựa trên việc áp dụng ngầm định ánh xạ ϕ. Ta có thể nghĩ về ϕ như cung cấp một tập hợp các đặc trưng mô tả $\boldsymbol{x}$, hoặc như cung cấp một biểu diễn mới cho $\boldsymbol{x}$.

Câu hỏi sau đó là làm thế nào để chọn ánh xạ ϕ.

- Một lựa chọn là sử dụng một ánh xạ ϕ rất tổng quát, chẳng hạn như ϕ vô hạn chiều được ngầm sử dụng bởi các thuật toán kernel dựa trên kernel RBF. Nếu $\phi(\boldsymbol{x})$ có chiều đủ lớn, ta luôn có đủ năng lực để khớp với tập huấn luyện, nhưng việc tổng quát hóa cho tập kiểm tra thường vẫn còn kém. Các ánh xạ đặc trưng rất tổng quát thường chỉ dựa trên nguyên tắc trơn địa phương và không mã hóa đủ thông tin tiên nghiệm để giải quyết các bài toán phức tạp.

- Lựa chọn khác là thiết kế thủ công ϕ. Trước khi học sâu ra đời, đây là cách tiếp cận chủ đạo. Cách tiếp cận này đòi hỏi hàng thập kỷ nỗ lực của con người cho mỗi tác vụ riêng biệt, với các nhà thực hành chuyên sâu vào các lĩnh vực khác nhau như nhận dạng giọng nói hay thị giác máy tính và ít có sự chuyển giao giữa các lĩnh vực.

- Chiến lược của học sâu là học ϕ. Trong cách tiếp cận này, ta có một mô hình $y = f\left(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w}\right) = \phi\left(\boldsymbol{x}; \boldsymbol{\theta}\right)^{\top} \boldsymbol{w}$. Ta hiện có các tham số $\boldsymbol{\theta}$ mà ta sử dụng để học ϕ từ một lớp rộng các hàm, và các tham số $\boldsymbol{w}$ để ánh xạ từ $\phi(\boldsymbol{x})$ đến đầu ra mong muốn. Đây là một ví dụ về mạng sâu truyền thẳng, với ϕ định nghĩa một lớp ẩn. Cách tiếp cận này là cách duy nhất trong ba cách từ bỏ tính lồi của bài toán huấn luyện, nhưng lợi ích của nó lớn hơn thiệt hại. Trong

cách này, ta tham số hóa biểu diễn dưới dạng $\phi\left(\boldsymbol{x};\boldsymbol{\theta}\right)$ và sử dụng thuật toán tối ưu hóa để tìm $\boldsymbol{\theta}$ tương ứng với một biểu diễn tốt. Nếu muốn, cách tiếp cận này có thể đạt được lợi ích của cách tiếp cận đầu tiên bằng cách rất tổng quát — ta thực hiện điều này bằng cách sử dụng một họ hàm rất rộng $\phi\left(\boldsymbol{x};\boldsymbol{\theta}\right)$. Cách tiếp cận này cũng có thể đạt được lợi ích của cách tiếp cận thứ hai. Các nhà thực hành có thể mã hóa kiến thức của họ để hỗ trợ việc tổng quát hóa bằng cách thiết kế các họ hàm $\phi\left(\boldsymbol{x};\boldsymbol{\theta}\right)$ mà họ kỳ vọng sẽ hoạt động tốt. Lợi thế là nhà thiết kế chỉ cần tìm họ hàm tổng quát đúng thay vì tìm chính xác hàm đúng.

Nguyên tắc chung của việc cải thiện mô hình bằng cách học các đặc trưng không chỉ giới hạn ở các mạng lan truyền thẳng được mô tả trong chương này. Đây là một chủ đề xuyên suốt của học sâu, áp dụng cho tất cả các loại mô hình được mô tả trong cuốn sách này. Các mạng lan truyền thẳng là sự ứng dụng của nguyên tắc này trong việc học các ánh xạ xác định từ $\boldsymbol{x}$ đến $\boldsymbol{y}$ mà không có các kết nối phản hồi. Các mô hình khác được giới thiệu sau sẽ áp dụng những nguyên tắc này cho việc học các ánh xạ ngẫu nhiên, học các hàm có phản hồi và học các phân phối xác suất trên một véctơ.

Chúng ta bắt đầu chương này bằng một ví dụ đơn giản về mạng lan truyền thẳng. Tiếp theo, ta sẽ xem xét từng quyết định thiết kế cần thiết để triển khai một mạng lan truyền thẳng. Đầu tiên, huấn luyện một mạng lan truyền thẳng đòi hỏi thực hiện nhiều quyết định thiết kế tương tự như cho một mô hình tuyến tính: chọn bộ tối ưu hóa, hàm chi phí và dạng của các đơn vị đầu ra. Chúng ta sẽ xem xét lại những điều cơ bản của học dựa trên gradient, sau đó tiếp cận các quyết định thiết kế đặc trưng của mạng lan truyền thẳng. Các mạng lan truyền thẳng đã giới thiệu khái niệm về lớp ẩn, và điều này yêu cầu ta phải chọn các **hàm kích hoạt** sẽ được dùng để tính giá trị của lớp ẩn. Ta cũng phải thiết kế kiến trúc của mạng, bao gồm việc xác định số lớp mà mạng cần có, cách các lớp này kết nối với nhau, và số lượng đơn vị trong mỗi lớp. Học trong các mạng nơron sâu yêu cầu tính gradient của các hàm phức tạp. Ta sẽ trình bày thuật toán **lan truyền ngược** và các tổng quát hóa hiện đại của nó, có thể được sử dụng để tính gradient một cách hiệu quả. Cuối cùng, chúng ta kết thúc với một số quan điểm lịch sử.

6.1 Ví dụ: học hàm XOR

Để làm rõ ý tưởng về mạng lan truyền thẳng, ta bắt đầu với một ví dụ về một mạng lan truyền thẳng hoàn chỉnh trên một tác vụ rất đơn giản: học hàm XOR.

Hàm XOR (hay "phép tuyển loại") là một phép toán trên hai giá trị nhị phân x_1 và x_2. Khi chỉ có duy nhất một trong hai giá trị này bằng 1, hàm XOR trả về 1. Nếu không, hàm trả về 0. Hàm XOR cung cấp hàm mục tiêu $y = f^*(\boldsymbol{x})$ mà ta muốn học. Mô hình cung cấp một hàm $y = f\left(\boldsymbol{x}; \boldsymbol{\theta}\right)$ và thuật toán học sẽ điều chỉnh các tham số $\boldsymbol{\theta}$ để làm cho f gần giống nhất có thể với f^*.

Trong ví dụ đơn giản này, ta sẽ không quan tâm đến khả năng tổng quát hóa thống kê. Ta chỉ muốn mạng thực hiện chính xác trên bốn điểm $\mathbb{X} = \{[0,0]^\top, [0,1]^\top, [1,0]^\top, [1,1]^\top\}$. Ta sẽ huấn luyện mạng trên cả bốn điểm này. Thách thức duy nhất là khớp với tập huấn luyện.

Ta có thể xử lý vấn đề này như một bài toán hồi quy và sử dụng hàm mất mát là sai số bình phương trung bình (MSE). Ta chọn hàm mất mát này để đơn giản hóa toán học cho ví dụ này càng nhiều càng tốt. Trong các ứng dụng thực tế, MSE thường không phải là hàm chi phí phù hợp để mô hình hóa dữ liệu nhị phân. Các phương pháp phù hợp hơn sẽ được mô tả trong Mục 6.2.2.2.

Khi đánh giá trên toàn bộ tập huấn luyện, hàm mất mát MSE là

$$J\left(\boldsymbol{\theta}\right) = \frac{1}{4}\sum_{x\in\mathbb{X}}\left[f^*(\boldsymbol{x}) - f\left(\boldsymbol{x},\boldsymbol{\theta}\right)^2\right]^2. \tag{6.1}$$

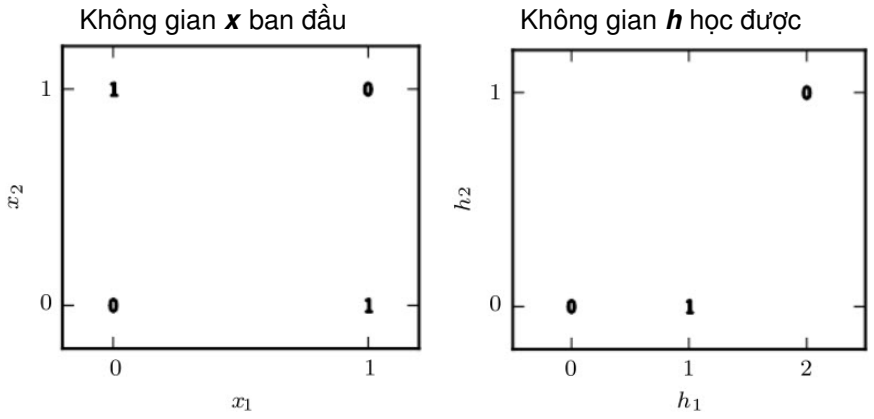
Bây giờ ta cần chọn dạng cho mô hình của mình, $f\left(\boldsymbol{x};\boldsymbol{\theta}\right)$. Giả sử chúng ta chọn mô hình tuyến tính, với $\boldsymbol{\theta}$ gồm $\boldsymbol{w}$ và b. Mô hình được định nghĩa như sau:

$$f(\boldsymbol{x};\boldsymbol{w},b) = \boldsymbol{x}^\top\boldsymbol{w} + b. \tag{6.2}$$

Chúng ta có thể cực tiểu hóa $J\left(\boldsymbol{\theta}\right)$ theo $\boldsymbol{w}$ và b theo dạng đóng bằng cách sử dụng phương trình chuẩn tắc.

Sau khi giải phương trình chuẩn tắc, ta thu được $\boldsymbol{w} = \boldsymbol{0}$ và $b = \frac{1}{2}$. Mô hình tuyến tính chỉ đưa ra giá trị 0.5 tại mọi điểm. Tại sao điều này lại xảy ra? Hình 6.1 cho thấy mô hình tuyến tính không thể biểu diễn hàm XOR. Một cách để giải quyết vấn đề này là sử dụng một mô hình học được một không gian đặc trưng khác, trong đó mô hình tuyến tính có thể biểu diễn được nghiệm.

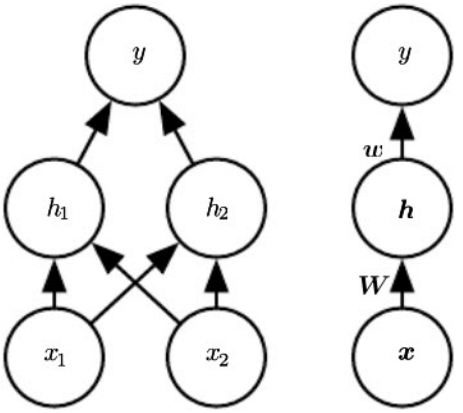
Ở đây, chúng ta sẽ giới thiệu một mạng lan truyền thẳng đơn giản với một lớp ẩn chứa hai đơn vị ẩn. Hình 6.2 minh họa mô hình này. Mạng lan truyền thẳng này có một véctơ các đơn vị ẩn $\boldsymbol{h}$ được tính bởi một hàm $f^{(1)}(\boldsymbol{x};\boldsymbol{W},\boldsymbol{c})$. Giá trị của các đơn



Hình 6.1: Giải bài toán XOR bằng cách học một biểu diễn mới. Các con số in đậm trên đồ thị biểu thị giá trị mà hàm đã học phải trả về tại mỗi điểm. *Hình trái*: Một mô hình tuyến tính áp dụng trực tiếp lên đầu vào ban đầu không thể thực hiện được với hàm XOR. Khi $x_1 = 0$, đầu ra của mô hình phải tăng khi x_2 tăng. Khi $x_1 = 1$, đầu ra của mô hình phải giảm khi x_2 tăng. Tuy nhiên, một mô hình tuyến tính phải áp dụng một hệ số cố định w_2 cho x_2. Vì vậy, mô hình tuyến tính không thể sử dụng giá trị của x_1 để thay đổi hệ số của x_2 và không thể giải quyết bài toán này. *Hình phải*: Trong không gian đã biến đổi được biểu diễn bởi các đặc trưng do mạng nơron trích xuất, mô hình tuyến tính có thể giải quyết được bài toán này. Trong lời giải ví dụ, hai điểm cần đầu ra là 1 đã được gộp lại thành một điểm duy nhất trong không gian đặc trưng. Nói cách khác, các đặc trưng phi tuyến đã ánh xạ cả hai điểm $\boldsymbol{x} = [1, 0]^\top$ và $\boldsymbol{x} = [0, 1]^\top$ vào một điểm duy nhất trong không gian đặc trưng là $\boldsymbol{h} = [1, 0]^\top$. Mô hình tuyến tính giờ đây có thể mô tả hàm là tăng theo h_1 và giảm theo h_2. Trong ví dụ này, mục đích của việc học không gian đặc trưng là để tăng năng lực của mô hình sao cho nó có thể khớp với tập huấn luyện. Trong các ứng dụng thực tế, các biểu diễn học được cũng có thể giúp mô hình tổng quát hóa tốt hơn.

vị ẩn này sau đó được sử dụng làm đầu vào cho lớp thứ hai, là lớp đầu ra của mạng. Lớp đầu ra này chỉ là một mô hình hồi quy tuyến tính, nhưng giờ đây nó được áp dụng lên $\boldsymbol{h}$ thay vì $\boldsymbol{x}$. Mạng giờ chứa hai hàm hợp thành với nhau: $\boldsymbol{h} = f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$ và $y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$, với mô hình hoàn chỉnh là $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}\left(f^{(1)}(x)\right)$.

Vậy, hàm $f^{(1)}$ nên tính toán gì? Các mô hình tuyến tính đã phục vụ chúng ta tốt đến thời điểm này, và có thể sẽ hấp dẫn khi làm cho $f^{(1)}$ cũng là tuyến tính. Tuy nhiên, nếu $f^{(1)}$ là tuyến tính, thì toàn bộ mạng lan truyền thẳng sẽ vẫn là một hàm tuyến tính của đầu vào. Giả sử $f^{(1)}(\boldsymbol{x}) = \boldsymbol{W}^\top \boldsymbol{x}$ và $f^{(2)}(\boldsymbol{h}) = \boldsymbol{h}^\top \boldsymbol{w}$, khi đó

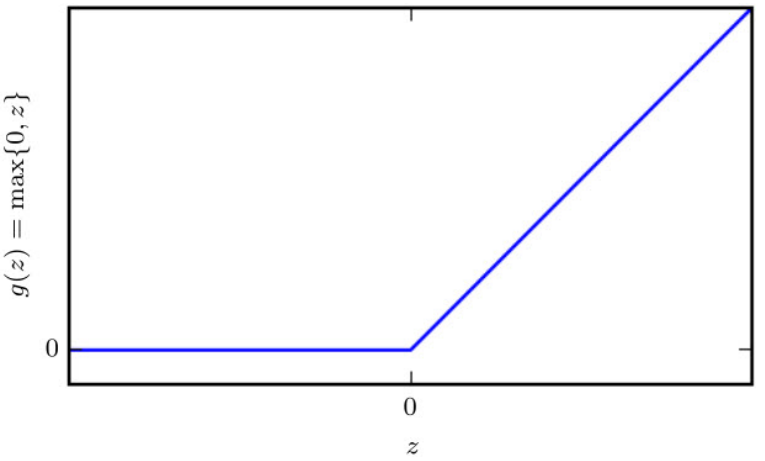


Hình 6.2: Minh họa một ví dụ về mạng lan truyền thẳng, được vẽ theo hai phong cách khác nhau. Đây là mạng lan truyền thẳng ta sử dụng để giải ví dụ XOR. Mạng này có một lớp ẩn chứa hai đơn vị. *Hình trái*: Ở phong cách này, ta vẽ từng đơn vị dưới dạng một nút trong đồ thị. Phong cách này rất rõ ràng và dễ hiểu, nhưng đối với các mạng lớn hơn ví dụ này, nó có thể chiếm quá nhiều không gian. *Hình phải*: Ở phong cách này, ta vẽ một nút trong đồ thị đại diện cho toàn bộ véctơ biểu diễn các kích hoạt của một lớp. Phong cách này gọn gàng hơn nhiều. Đôi khi, ta chú thích các cạnh trong đồ thị bằng tên của các tham số mô tả mối quan hệ giữa hai lớp. Ở đây, ta chỉ ra rằng ma trận $\boldsymbol{W}$ mô tả ánh xạ từ $\boldsymbol{x}$ sang $\boldsymbol{h}$, và véctơ $\boldsymbol{w}$ mô tả ánh xạ từ $\boldsymbol{h}$ sang y. Khi ghi nhãn kiểu hình vẽ này, ta thường bỏ qua các tham số dịch chuyển liên quan đến mỗi lớp.

$f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{W}^\top \boldsymbol{x}$, và ta có thể biểu diễn hàm này bởi $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w}'$, với $\boldsymbol{w}' = \boldsymbol{W}\boldsymbol{w}$.

Rõ ràng, chúng ta cần sử dụng một hàm phi tuyến để mô tả các đặc trưng. Hầu hết các mạng nơron thực hiện điều này bằng cách sử dụng một phép biến đổi affine được điều khiển bởi các tham số học được, sau đó là một hàm phi tuyến cố định gọi là hàm kích hoạt. Ta sẽ áp dụng chiến lược này ở đây, bằng cách định nghĩa $\boldsymbol{h} = g\left(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\right)$, trong đó $\boldsymbol{W}$ cung cấp các trọng số của phép biến đổi tuyến tính và $\boldsymbol{c}$ là các hệ số dịch chuyển. Trước đây, để mô tả một mô hình hồi quy tuyến tính, ta sử dụng một véctơ trọng số và một hệ số dịch chuyển vô hướng để mô tả phép biến đổi affine từ một véctơ đầu vào sang một đầu ra vô hướng. Bây giờ, ta mô tả phép biến đổi affine từ một véctơ $\boldsymbol{x}$ sang một véctơ $\boldsymbol{h}$, vì vậy cần có cả một véctơ các hệ số dịch chuyển. Hàm kích hoạt g thường được chọn là một hàm áp dụng cho từng phần tử, với $h_i = g\left(\boldsymbol{x}^\top \boldsymbol{W}_{:,i} + c_i\right)$. Trong các mạng nơron hiện đại, đề xuất mặc định là sử dụng **đơn vị tuyến tính chỉnh lưu** (rectified linear unit, ReLU) (*What is the Best Multi-Stage Architecture for Object Recognition?*, Jarrett và cộng

sự, 2009, [1]; *Rectified Linear Units Improve Restricted Boltzmann Machines*, Nair và Hinton, 2010, [2]; *Deep Sparse Rectifier Neural Networks*, Glorot và cộng sự, 2011, [3]), được định nghĩa bởi hàm kích hoạt $g(z) = \max\{0, z\}$, như minh họa trong Hình 6.3.



Hình 6.3: Hàm kích hoạt tuyến tính chỉnh lưu ReLU. Đây là hàm kích hoạt mặc định được khuyến nghị sử dụng với hầu hết các mạng nơron lan truyền thẳng. Khi áp dụng hàm này cho đầu ra của một phép biến đổi tuyến tính, ta có được một phép biến đổi phi tuyến. Tuy nhiên, hàm này vẫn gần như tuyến tính vì nó là một hàm tuyến tính từng đoạn với hai đoạn tuyến tính. Vì ReLU gần như tuyến tính, nó giữ lại nhiều đặc tính khiến các mô hình tuyến tính dễ tối ưu bằng các phương pháp dựa trên gradient. Nó cũng giữ lại nhiều đặc tính khiến các mô hình tuyến tính có khả năng tổng quát hóa tốt. Một nguyên tắc phổ biến trong khoa học máy tính là có thể xây dựng các hệ thống phức tạp từ những thành phần đơn giản. Tương tự như bộ nhớ của máy Turing chỉ cần lưu trữ các trạng thái 0 hoặc 1, chúng ta có thể xây dựng một bộ xấp xỉ hàm phổ quát từ các hàm ReLU.

Bây giờ ta có thể xác định hoàn chỉnh mạng nơron bởi hàm:

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{\boldsymbol{0}, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b. \tag{6.3}$$

Tiếp theo ta có thể xác định một nghiệm của bài toán XOR. Xét

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \tag{6.4}$$

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \tag{6.5}$$

$$\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad \text{và} \tag{6.6}$$

$$b = 0. \tag{6.7}$$

Bây giờ, ta sẽ đi qua từng bước mà mô hình xử lý một nhóm các đầu vào. Với $\boldsymbol{X}$ là ma trận thiết kế chứa tất cả bốn điểm trong không gian đầu vào nhị phân, mỗi hàng là một ví dụ:

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{6.8}$$

Bước đầu tiên trong mạng nơron là nhân ma trận đầu vào $\boldsymbol{X}$ với ma trận trọng số W của lớp đầu tiên:

$$\boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}. \tag{6.9}$$

Tiếp theo, ta cộng với véctơ dịch chuyển $\boldsymbol{c}$, được

$$\boldsymbol{XW} + \boldsymbol{c} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}. \tag{6.10}$$

Trong không gian này, tất cả các điểm đầu vào đều nằm dọc theo một đường thẳng có độ dốc bằng 1. Khi di chuyển dọc theo đường thẳng này, đầu ra cần phải bắt đầu từ 0, sau đó tăng lên 1 và rồi giảm trở lại về 0. Một mô hình tuyến tính không thể thực hiện một chức năng như vậy vì các mô hình tuyến tính chỉ có thể mô tả các mối quan hệ đơn điệu giữa đầu vào và đầu ra, không có khả năng thể hiện sự thay đổi hướng trong quan hệ đầu vào – đầu ra như mô hình XOR yêu cầu. Để hoàn thành việc tính toán giá trị của $\boldsymbol{h}$ cho từng điểm đầu vào, ta áp dụng hàm kích hoạt tuyến tính chỉnh lưu (ReLU) cho từng phần tử, biến tất cả giá trị âm thành 0:

$$\boldsymbol{h} = \max\{\boldsymbol{0}, \boldsymbol{XW} + \boldsymbol{c}\} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}. \tag{6.11}$$

Sau khi áp dụng phép biến đổi ReLU, các điểm dữ liệu không còn nằm trên một đường thẳng nữa. Điều này có nghĩa là chúng đã được "phân tách" trong một không gian mới, nơi mối quan hệ giữa các điểm đầu vào trở nên phức tạp hơn. Như được minh họa trong Hình 6.1, sự thay đổi này cho phép một mô hình tuyến tính có thể giải quyết bài toán XOR.

Cuối cùng, chúng ta nhân kết quả $\boldsymbol{h}$ với véctơ trọng số $\boldsymbol{w}$ của lớp đầu ra:

$$\boldsymbol{hw} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}. \tag{6.12}$$

Kết quả này chính là đầu ra mà chúng ta mong muốn cho mỗi điểm đầu vào trong bài toán XOR. Mạng nơron đã đạt được đáp án chính xác cho từng ví dụ trong nhóm đầu vào.

Trong ví dụ này, chúng ta chỉ đơn giản là xác định lời giải rồi chứng minh rằng nó đạt được sai số bằng không. Trong một tình huống thực tế, có thể có hàng tỷ tham số mô hình và hàng tỷ ví dụ huấn luyện, vì vậy không thể chỉ đơn thuần đoán lời giải như cách chúng ta đã làm ở đây. Thay vào đó, một thuật toán tối ưu hóa dựa trên gradient có thể tìm ra các tham số cho ra sai số rất nhỏ. Lời giải mà chúng ta đã mô tả cho bài toán XOR nằm tại một cực tiểu toàn cục của hàm mất mát, vì vậy thuật toán hướng giảm có thể hội tụ tới điểm này. Cũng có những lời giải tương đương khác cho bài toán XOR mà thuật toán hướng giảm có thể tìm thấy. Điểm hội tụ của thuật toán hướng giảm phụ thuộc vào các giá trị khởi tạo của tham số. Trên thực tế, thuật toán hướng giảm thường sẽ không tìm thấy các nghiệm đơn giản, dễ hiểu và có giá trị nguyên như nghiệm đã trình bày ở đây.

6.2 Học dựa trên gradient

Thiết kế và huấn luyện một mạng nơron không khác biệt nhiều so với huấn luyện bất kỳ mô hình máy học nào khác bằng phương pháp hướng giảm. Trong Mục 5.10, chúng ta đã mô tả cách xây dựng một thuật toán máy học bằng cách chỉ định một quy trình tối ưu, một hàm chi phí và một họ mô hình.

Sự khác biệt lớn nhất giữa các mô hình tuyến tính mà chúng ta đã thấy cho đến nay và các mạng nơron là tính phi tuyến của mạng nơron làm cho hầu hết các hàm mất mát thú vị trở nên không lồi. Điều này có nghĩa là mạng nơron thường được huấn luyện bằng cách sử dụng các bộ tối ưu lặp dựa trên gradient, chỉ giảm

hàm chi phí xuống giá trị rất thấp, thay vì sử dụng các trình giải phương trình tuyến tính để huấn luyện các mô hình hồi quy tuyến tính hoặc các thuật toán tối ưu lồi với bảo đảm hội tụ toàn cục dùng để huấn luyện hồi quy logistic hoặc SVM. Tối ưu lồi hội tụ từ bất kỳ khởi tạo tham số nào (trên lý thuyết — trong thực tế, nó rất mạnh mẽ nhưng có thể gặp phải các vấn đề về tính toán số). Phương pháp hướng giảm ngẫu nhiên áp dụng cho các hàm mất mát không lồi không có bảo đảm hội tụ như vậy và nhạy cảm với giá trị khởi tạo của các tham số. Đối với các mạng nơron lan truyền thẳng, việc khởi tạo tất cả các trọng số thành các giá trị ngẫu nhiên nhỏ là rất quan trọng. Các tham số dịch chuyển có thể được khởi tạo bằng 0 hoặc bằng các giá trị dương nhỏ. Các thuật toán tối ưu lặp dựa trên gradient được sử dụng để huấn luyện mạng lan truyền thẳng và hầu hết các mô hình sâu khác sẽ được mô tả chi tiết trong Chương 8, với việc khởi tạo tham số được thảo luận đặc biệt trong Mục 8.4. Tạm thời, chỉ cần hiểu rằng thuật toán huấn luyện hầu như luôn dựa trên việc sử dụng gradient để giảm hàm chi phí theo cách này hay cách khác. Các thuật toán cụ thể là những cải tiến và tinh chỉnh trên các ý tưởng của phương pháp hướng giảm, đã được giới thiệu trong Mục 4.3, và cụ thể hơn, phần lớn là các cải tiến của thuật toán hướng giảm ngẫu nhiên, đã được giới thiệu trong Mục 5.9.

Chúng ta tất nhiên cũng có thể huấn luyện các mô hình như hồi quy tuyến tính và thuật toán véctơ hỗ trợ (SVM) bằng cách sử dụng phương pháp hướng giảm, và thực tế điều này là phổ biến khi tập huấn luyện rất lớn. Từ quan điểm này, việc huấn luyện một mạng nơron không khác mấy so với việc huấn luyện các mô hình khác. Việc tính toán gradient phức tạp hơn một chút đối với mạng nơron, nhưng vẫn có thể thực hiện hiệu quả và chính xác. Mục 6.5 sẽ mô tả cách thu được gradient bằng thuật toán lan truyền ngược và các tổng quát hóa hiện đại của thuật toán này.

Giống như các mô hình học máy khác, để áp dụng học dựa trên gradient, chúng ta phải chọn một hàm chi phí và cách biểu diễn đầu ra của mô hình. Bây giờ, chúng ta sẽ xem xét lại những cân nhắc về thiết kế này với trọng tâm đặc biệt vào bối cảnh của mạng nơron.

6.2.1 Hàm chi phí

Một khía cạnh quan trọng trong thiết kế mạng nơron sâu là việc chọn hàm chi phí. May mắn thay, các hàm chi phí cho mạng nơron hầu như giống với các mô hình tham số khác, như các mô hình tuyến tính.

Trong hầu hết các trường hợp, mô hình có tham số xác định một phân phối $p\left(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}\right)$ và chúng ta đơn giản áp dụng nguyên lý hợp lý cực đại. Điều này có

nghĩa là ta sử dụng entropy chéo giữa dữ liệu huấn luyện và các dự đoán của mô hình làm hàm chi phí.

Đôi khi, chúng ta sử dụng một cách tiếp cận đơn giản hơn, thay vì dự đoán một phân phối xác suất hoàn chỉnh trên $\boldsymbol{y}$, chúng ta chỉ dự đoán một số thống kê của $\boldsymbol{y}$ khi biết $\boldsymbol{x}$. Các hàm mất mát chuyên biệt cho phép chúng ta huấn luyện một mô hình dự đoán các ước lượng này.

Hàm chi phí tổng thể được sử dụng để huấn luyện một mạng nơron thường kết hợp một trong những hàm chi phí chính được mô tả ở đây với một hạng tử điều chuẩn. Chúng ta đã thấy một số ví dụ đơn giản về điều chuẩn áp dụng cho các mô hình tuyến tính trong Mục 5.2.2. Phương pháp suy giảm trọng số sử dụng cho các mô hình tuyến tính cũng có thể áp dụng trực tiếp cho mạng nơron sâu và là một trong những chiến lược điều chuẩn phổ biến nhất. Các chiến lược chuẩn hóa nâng cao hơn cho mạng nơron sẽ được mô tả trong Chương 7.

6.2.1.1 Học phân phối có điều kiện bằng phương pháp hợp lý cực đại

Hầu hết các mạng nơron hiện đại được huấn luyện bằng cách sử dụng phương pháp hợp lý cực đại. Điều này có nghĩa là hàm chi phí đơn giản là đối của logarit hàm hợp lý, hoặc tương đương là entropy chéo giữa dữ liệu huấn luyện và phân phối của mô hình. Hàm chi phí này được cho bởi

$$J\left(\boldsymbol{\theta}\right) = -E_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}\left(\boldsymbol{y} \mid \boldsymbol{x}\right). \tag{6.13}$$

Dạng cụ thể của hàm chi phí thay đổi tùy theo từng mô hình, dựa trên dạng cụ thể của $\log p_{\text{model}}$. Việc mở rộng phương trình trên thường tạo ra một số hạng không phụ thuộc vào các tham số của mô hình và có thể bị bỏ qua. Ví dụ, như chúng ta đã thấy trong Mục 5.5.1, nếu $p_{\text{model}}\left(\boldsymbol{y} \mid \boldsymbol{x}\right) = N\left(\boldsymbol{y}; f\left(\boldsymbol{x};\boldsymbol{\theta}\right), \boldsymbol{I}\right)$, thì ta lại thu được hàm chi phí sai số bình phương trung bình,

$$J\left(\boldsymbol{\theta}\right) = \frac{1}{2}E_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{\text{data}}}\left\|\boldsymbol{y} - f\left(\boldsymbol{x},\boldsymbol{\theta}\right)\right\|^2 + \text{const}, \tag{6.14}$$

với một hệ số tỉ lệ là $\frac{1}{2}$ và một số hạng không phụ thuộc vào $\boldsymbol{\theta}$. Hằng số bị bỏ qua là dựa trên phương sai của phân phối Gauss, mà trong trường hợp này ta không chọn để tham số hóa. Trước đó, ta đã thấy đối với một mô hình tuyến tính, có sự tương đương giữa ước lượng hợp lý cực đại với một phân phối đầu ra và việc cực tiểu hóa sai số bình phương trung bình, nhưng thực tế, sự tương đương này đúng với bất kỳ hàm $f\left(\boldsymbol{x};\boldsymbol{\theta}\right)$ nào được dùng để dự đoán giá trị trung bình của phân phối Gauss.

Một lợi thế của cách tiếp cận này, khi dẫn ra hàm chi phí từ phương pháp hợp lý cực đại, là nó giúp giảm bớt việc phải thiết kế các hàm chi phí cho từng mô hình. Việc xác định một mô hình $p\left(\boldsymbol{y} \mid \boldsymbol{x}\right)$ sẽ tự động quyết định hàm chi phí $\log p\left(\boldsymbol{y} \mid \boldsymbol{x}\right)$.

Một chủ đề lặp lại trong thiết kế mạng nơron là gradient của hàm chi phí phải đủ lớn và có thể dự đoán được để đóng vai trò là hướng dẫn tốt cho thuật toán học. Các hàm bão hòa (trở nên rất phẳng) làm suy yếu mục tiêu này vì chúng khiến gradient trở nên rất nhỏ. Trong nhiều trường hợp, điều này xảy ra do các hàm kích hoạt được sử dụng để tạo đầu ra của các đơn vị ẩn hoặc các đơn vị đầu ra bị bão hòa. Hàm đối của logarit hàm hợp lý giúp tránh vấn đề này cho nhiều mô hình. Nhiều đơn vị đầu ra bao gồm một hàm mũ có thể bão hòa khi đối số của nó rất âm. Hàm log trong hàm chi phí đối của logarit hàm hợp lý sẽ triệt tiêu hàm mũ của một số đơn vị đầu ra. Chúng ta sẽ thảo luận về mối quan hệ giữa hàm chi phí và sự lựa chọn của đơn vị đầu ra trong Mục 6.2.2.

Một đặc điểm khác thường của hàm chi phí entropy chéo, được sử dụng để thực hiện ước lượng hợp lý cực đại, là nó thường không có giá trị nhỏ nhất khi được áp dụng cho các mô hình thường dùng trong thực tế. Đối với các biến đầu ra rời rạc, hầu hết các mô hình được tham số hóa theo cách không thể biểu diễn xác suất bằng không hoặc một, nhưng có thể đến rất gần các giá trị đó. Hồi quy logistic là một ví dụ của loại mô hình này. Đối với các biến đầu ra có giá trị thực, nếu mô hình có thể điều khiển mật độ của phân phối đầu ra (chẳng hạn, bằng cách học tham số phương sai của phân phối đầu ra Gauss), thì có thể gán mật độ cực cao cho các đầu ra chính xác trong tập huấn luyện, dẫn đến entropy chéo tiến tới âm vô cùng. Các kỹ thuật điều chuẩn được mô tả trong Chương 7 cung cấp một số cách khác nhau để sửa đổi vấn đề học, nhằm ngăn cản mô hình đạt được lợi ích không giới hạn theo cách này.

6.2.1.2 Học thống kê có điều kiện

Thay vì học toàn bộ phân phối xác suất $p\left(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}\right)$, chúng ta thường chỉ muốn học một thống kê điều kiện cụ thể của $\boldsymbol{y}$ khi biết $\boldsymbol{x}$.

Chẳng hạn, ta có thể có một bộ dự đoán $f\left(\boldsymbol{x}; \boldsymbol{\theta}\right)$ mà ta mong muốn dự đoán giá trị trung bình của $\boldsymbol{y}$.

Nếu sử dụng một mạng nơron đủ mạnh, ta có thể coi mạng nơron này như có khả năng biểu diễn bất kỳ hàm f nào từ một lớp rộng các hàm, với lớp này chỉ bị giới hạn bởi các đặc tính như tính liên tục và bị chặn, thay vì có một dạng có tham

số cụ thể. Từ quan điểm này, ta có thể coi hàm chi phí như một **phiếm hàm** thay vì chỉ là một hàm thông thường. Một phiếm hàm là một ánh xạ từ các hàm tới các số thực. Do đó, ta có thể xem việc học như là chọn một hàm thay vì chỉ là chọn một tập hợp các tham số. Ta có thể thiết kế phiếm hàm chi phí sao cho điểm cực tiểu của nó xảy ra tại một hàm cụ thể mà ta mong muốn. Chẳng hạn, ta có thể thiết kế phiếm hàm chi phí sao cho điểm cực tiểu của nó nằm trên hàm ánh xạ từ $\boldsymbol{x}$ đến giá trị kỳ vọng của $\boldsymbol{y}$ khi biết $\boldsymbol{x}$. Giải một bài toán tối ưu theo hàm đòi hỏi một công cụ toán học gọi là **giải tích biến phân**, được mô tả trong Mục 13.1.1. Để hiểu nội dung của chương này, không nhất thiết phải nắm rõ giải tích biến phân. Hiện tại, chỉ cần hiểu rằng giải tích biến phân có thể được sử dụng để dẫn ra hai kết quả sau đây.

Kết quả đầu tiên được dẫn ra bằng giải tích biến phân là khi giải bài toán tối ưu

$$f^* = \arg\min_f E_{\boldsymbol{x},\boldsymbol{y}\sim p_{\text{data}}} \|\boldsymbol{y} - f(\boldsymbol{x})\|^2 \tag{6.15}$$

sẽ cho

$$f^*(\boldsymbol{x}) = E_{\boldsymbol{y}\sim p_{\text{data}}(\boldsymbol{y}|\boldsymbol{x})}[\boldsymbol{y}], \tag{6.16}$$

miễn là hàm này nằm trong lớp hàm mà ta đang tối ưu. Nói cách khác, nếu ta có thể huấn luyện trên số lượng mẫu vô hạn từ phân phối sinh dữ liệu thực, việc cực tiểu hàm chi phí sai số bình phương trung bình sẽ cho ra một hàm dự đoán giá trị trung bình của $\boldsymbol{y}$ cho mỗi giá trị của $\boldsymbol{x}$.

Các hàm chi phí khác nhau sẽ cho ra các thống kê khác nhau. Một kết quả thứ hai dẫn ra từ giải tích biến phân là

$$f^* = \arg\min_f E_{\boldsymbol{x},\boldsymbol{y}\sim p_{\text{data}}} \|\boldsymbol{y} - f(\boldsymbol{x})\|_1 \tag{6.17}$$

sẽ cho ra một hàm dự đoán giá trị trung vị của $\boldsymbol{y}$ cho mỗi $\boldsymbol{x}$, miễn là hàm đó có thể được biểu diễn bởi họ các hàm mà ta đang tối ưu. Hàm chi phí này thường được gọi là sai số **tuyệt đối trung bình** (mean absolute error, MAE).

Đáng tiếc, hàm chi phí sai số bình phương trung bình và sai số tuyệt đối trung bình thường dẫn đến kết quả không tốt khi sử dụng với tối ưu hóa dựa trên gradient. Một số đơn vị đầu ra có thể bão hòa và tạo ra gradient rất nhỏ khi kết hợp với các hàm chi phí này. Đây là một lý do khiến hàm chi phí entropy chéo phổ biến hơn so với sai số bình phương trung bình hoặc sai số tuyệt đối trung bình, ngay cả khi không cần phải ước lượng toàn bộ phân phối $p(\boldsymbol{y} \mid \boldsymbol{x})$.

6.2.2 Các đơn vị đầu ra

Việc chọn hàm chi phí có mối liên hệ chặt chẽ với việc chọn đơn vị đầu ra. Hầu hết thời gian, chúng ta đơn giản sử dụng hàm entropy chéo giữa phân phối dữ liệu và phân phối của mô hình. Cách chọn biểu diễn đầu ra sẽ quyết định dạng của hàm entropy chéo.

Bất kỳ loại đơn vị nào của mạng nơron có thể được sử dụng làm đầu ra đều có thể được dùng làm đơn vị ẩn. Ở đây, chúng ta tập trung vào việc sử dụng các đơn vị này như là đầu ra của mô hình, nhưng về nguyên tắc, chúng cũng có thể được sử dụng ở bên trong mạng. Chúng ta sẽ xem lại các đơn vị này với chi tiết bổ sung về cách sử dụng chúng làm đơn vị ẩn trong Mục 6.3.

Xuyên suốt phần này, giả sử mạng lan truyền thẳng cung cấp một tập hợp các đặc trưng ẩn được định nghĩa bởi $\boldsymbol{h} = f\left(\boldsymbol{x}; \boldsymbol{\theta}\right)$. Vai trò của lớp đầu ra là cung cấp một biến đổi bổ sung từ các đặc trưng này để hoàn thành tác vụ mà mạng cần thực hiện.

6.2.2.1 Đơn vị tuyến tính cho phân phối đầu ra Gauss

Một loại đơn vị đầu ra đơn giản là đơn vị dựa trên một phép biến đổi affine không có tính phi tuyến. Những đơn vị này thường được gọi là đơn vị tuyến tính.

Với đặc trưng $\boldsymbol{h}$, một lớp các đơn vị đầu ra tuyến tính tạo ra một véctơ $\widehat{\boldsymbol{y}} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}$.

Các lớp đầu ra tuyến tính thường được sử dụng để tạo ra trung bình của một phân phối Gauss có điều kiện:

$$p\left(\boldsymbol{y} \mid \boldsymbol{x}\right) = N\left(\boldsymbol{y}; \widehat{\boldsymbol{y}}, \boldsymbol{I}\right). \tag{6.18}$$

Cực đại hóa logarit của hàm hợp lý khi đó tương đương với cực tiểu hóa sai số bình phương trung bình.

Phương pháp hợp lý cực đại cũng cho phép dễ dàng học ma trận hiệp phương sai của phân phối Gauss, hoặc khiến ma trận hiệp phương sai trở thành một hàm của đầu vào. Tuy nhiên, ma trận hiệp phương sai cần phải là một ma trận xác định dương cho mọi đầu vào. Rất khó để đảm bảo các ràng buộc như vậy chỉ với lớp đầu ra tuyến tính, nên thường sử dụng các đơn vị đầu ra khác để tham số hóa hiệp phương sai. Các cách tiếp cận để mô hình hóa hiệp phương sai sẽ được mô tả ngay trong Mục 6.2.2.4.

Do các đơn vị tuyến tính không bị bão hòa, chúng ít gây khó khăn cho các thuật toán tối ưu hóa dựa trên gradient và có thể được sử dụng với nhiều loại thuật toán

tối ưu hóa khác nhau.

6.2.2.2 Đơn vị sigmoid cho phân phối đầu ra Bernoulli

Nhiều tác vụ yêu cầu dự đoán giá trị của một biến nhị phân y. Các bài toán phân loại với hai lớp có thể được xây dựng dưới dạng này.

Phương pháp hợp lý cực đại sẽ định nghĩa một phân phối Bernoulli trên y có điều kiện theo $\boldsymbol{x}$.

Phân phối Bernoulli chỉ được định nghĩa bởi một số duy nhất. Mạng nơron chỉ cần dự đoán $P\left(y = 1 \mid \boldsymbol{x}\right)$. Để số này trở thành một xác suất hợp lệ, nó phải nằm trong khoảng [0, 1].

Để thỏa mãn ràng buộc này, cần thiết kế cẩn thận. Giả sử ta dùng một đơn vị tuyến tính và ngưỡng giá trị của nó để đạt được một xác suất hợp lệ:

$$P\left(y = 1 \mid \boldsymbol{x}\right) = \max\left\{0, \min\left\{1, \boldsymbol{w}^\top \boldsymbol{h} + b\right\}\right\}. \tag{6.19}$$

Đúng là điều này sẽ định nghĩa một phân phối có điều kiện hợp lệ, nhưng chúng ta không thể huấn luyện nó một cách hiệu quả bằng phương pháp hướng giảm. Bất kỳ lúc nào $\boldsymbol{w}^\top \boldsymbol{h} + b$ lệch khỏi khoảng đơn vị, gradient của đầu ra mô hình theo các tham số sẽ là $\mathbf{0}$. Gradient bằng $\mathbf{0}$ thường gây ra vấn đề vì thuật toán học sẽ mất phương hướng để cải thiện các tham số tương ứng.

Thay vào đó, một cách tiếp cận tốt hơn là sử dụng phương pháp đảm bảo luôn có gradient mạnh khi mô hình đưa ra kết quả sai. Cách tiếp cận này dựa trên việc sử dụng các đơn vị đầu ra sigmoid kết hợp với phương pháp hợp lý.

Một đơn vị đầu ra sigmoid được định nghĩa bởi

$$\widehat{y} = \sigma\left(w^\top h + b\right) \tag{6.20}$$

với σ là hàm sigmoid logistic, được mô tả trong Mục 3.10.

Ta có thể coi đơn vị đầu ra sigmoid gồm hai phần. Đầu tiên, nó sử dụng một lớp tuyến tính để tính $z = \boldsymbol{w}^\top \boldsymbol{h} + b$. Sau đó, nó dùng hàm kích hoạt sigmoid để chuyển đổi z thành một xác suất.

Tạm thời bỏ qua sự phụ thuộc vào x, chúng ta sẽ thảo luận cách xác định một phân phối xác suất trên y sử dụng giá trị z. Hàm sigmoid có thể được lý giải bằng cách xây dựng một phân phối xác suất chưa chuẩn hóa $\widetilde{P}(y)$, chưa có tổng bằng 1. Ta có thể chia cho một hằng số phù hợp để thu được một phân phối xác suất hợp lệ. Nếu bắt đầu với giả thiết rằng logarit của các xác suất chưa chuẩn hóa là tuyến tính theo y và z, ta có thể lũy thừa để thu được các xác suất chưa chuẩn hóa. Sau

đó, chuẩn hóa để thấy rằng điều này dẫn đến một phân phối Bernoulli được điều khiển bởi một phép biến đổi sigmoid của z:

$$\log \widetilde{P}(y) = yz \tag{6.21}$$

$$\widetilde{P}(y) = e^{yz} \tag{6.22}$$

$$P(y) = \frac{e^{yz}}{\sum_{y'=0}^{1} e^{y'z}} \tag{6.23}$$

$$P(y) = \sigma((2y-1)z). \tag{6.24}$$

Các phân phối xác suất dựa trên hàm mũ và chuẩn hóa là phổ biến trong tài liệu về mô hình thống kê. Biến z xác định phân phối như vậy trên các biến nhị phân được gọi là **logit**.

Cách tiếp cận này trong việc dự đoán xác suất trong không gian logarit là tự nhiên khi sử dụng học cực đại hàm hợp lý. Vì hàm mất mát được sử dụng với phương pháp cực đại hàm hợp lý là $-\log P\left(y \mid \boldsymbol{x}\right)$, lấy logarit trong hàm mất mát sẽ làm mất đi phần hàm mũ của sigmoid. Nếu không có hiệu ứng này, việc bão hòa của sigmoid có thể ngăn cản quá trình học dựa trên gradient đạt tiến triển tốt. Hàm mất mát cho học cực đại hàm hợp lý của một phân phối Bernoulli được tham số hóa bởi sigmoid là

$$J\left(\boldsymbol{\theta}\right) = -\log P\left(y \mid \boldsymbol{x}\right) \tag{6.25}$$

$$= -\log \sigma((2y-1)z) \tag{6.26}$$

$$= \zeta((1-2y)z). \tag{6.27}$$

Phép suy diễn này sử dụng một số tính chất từ Mục 3.10. Bằng cách viết lại hàm mất mát dưới dạng hàm softplus, ta có thể thấy rằng nó chỉ bão hòa khi $(1-2y)z$ là rất âm. Sự bão hòa xảy ra chỉ khi mô hình đã có câu trả lời đúng — khi $y = 1$ và z là rất dương, hoặc $y = 0$ và z là rất âm. Khi z có dấu sai, đối số của hàm softplus, $(1-2y)z$, có thể được đơn giản hóa thành $|z|$. Khi $|z|$ trở nên lớn trong khi z có dấu sai, hàm softplus sẽ tiệm cận dần về đơn giản trả về đối số $|z|$ của nó, tức giá trị đối của z. Đạo hàm theo z tiệm cận về $\text{sign}(z)$, vì vậy, trong trường hợp z sai nghiêm trọng, hàm softplus không làm nhỏ đi gradient. Tính chất này rất hữu ích vì nó cho phép học dựa trên gradient có thể nhanh chóng điều chỉnh z khi có lỗi.

Khi sử dụng các hàm mất mát khác, chẳng hạn như sai số bình phương trung bình, hàm mất mát có thể bão hòa bất cứ khi nào $\sigma(z)$ bão hòa. Hàm kích hoạt

sigmoid bão hòa về 0 khi z trở nên rất âm và bão hòa về 1 khi z trở nên rất dương. Gradient có thể thu nhỏ đến mức quá nhỏ để có ích cho việc học bất cứ khi nào điều này xảy ra, dù mô hình có đưa ra câu trả lời đúng hay sai. Vì lý do này, phương pháp cực đại hợp lý gần như luôn là cách tiếp cận ưu tiên khi huấn luyện các đơn vị đầu ra sigmoid.

Về mặt phân tích, logarit của sigmoid luôn được xác định và hữu hạn, bởi vì sigmoid trả về các giá trị giới hạn trong khoảng mở $(0, 1)$, thay vì sử dụng toàn bộ khoảng đóng của xác suất hợp lệ $[0, 1]$. Trong các triển khai phần mềm, để tránh các vấn đề tính toán số, tốt nhất nên viết hàm đối của logarit hàm hợp lý như một hàm của z, thay vì là một hàm của $\widehat{y} = \sigma(z)$. Nếu hàm sigmoid bị hạ số xuống 0, thì việc lấy logarit của $\widehat{y}$ sẽ dẫn đến âm vô cùng.

6.2.2.3 Đơn vị softmax cho phân phối Bernoulli bội

Bất kỳ khi nào chúng ta muốn biểu diễn một phân phối xác suất trên một biến rời rạc với n giá trị có thể xảy ra, ta có thể sử dụng hàm softmax. Điều này có thể được xem như một sự tổng quát hóa của hàm sigmoid, vốn được dùng để biểu diễn phân phối xác suất trên một biến nhị phân.

Các hàm softmax thường được dùng nhất để làm đầu ra của một bộ phân loại, nhằm biểu diễn phân phối xác suất trên n lớp khác nhau. Hiếm hơn, hàm softmax cũng có thể được sử dụng bên trong mô hình, nếu chúng ta muốn mô hình chọn một trong n lựa chọn cho một biến nội bộ nào đó.

Trong trường hợp biến nhị phân, chúng ta muốn tạo ra một số duy nhất

$$\widehat{y} = P\left(y = 1 \mid \boldsymbol{x}\right). \tag{6.28}$$

Vì giá trị này cần phải nằm giữa 0 và 1, và vì ta muốn logarit của số này có hành vi tốt đối với tối ưu dựa trên gradient của logarit hàm hợp lý, ta đã chọn thay vào đó là dự đoán một giá trị $z = \log \widetilde{P}\left(y = 1 \mid \boldsymbol{x}\right)$. Việc lấy mũ và chuẩn hóa sẽ cho chúng ta một phân phối Bernoulli được điều khiển bởi hàm sigmoid.

Để tổng quát hóa cho trường hợp một biến rời rạc có n giá trị, bây giờ ta cần tạo ra một véctơ $\widehat{\boldsymbol{y}}$, với $\widehat{y}_i = P\left(y = i \mid \boldsymbol{x}\right)$. Ta yêu cầu không chỉ từng phần tử $\widehat{y}_i$ nằm trong khoảng từ 0 đến 1 mà toàn bộ véctơ phải có tổng bằng 1 để nó biểu diễn một phân phối xác suất hợp lệ. Cách tiếp cận đã sử dụng cho phân phối Bernoulli có thể được tổng quát hóa cho phân phối Bernoulli bội. Đầu tiên, một lớp tuyến tính dự đoán các logarit xác suất chưa chuẩn hóa:

$$\boldsymbol{z} = \boldsymbol{W}^{\top}\boldsymbol{h} + \boldsymbol{b}, \tag{6.29}$$

trong đó $z_i = \log \widetilde{P}\left(y = i \mid \boldsymbol{x}\right)$. Sau đó, hàm softmax sẽ lấy mũ và chuẩn hóa $\boldsymbol{z}$ để có được $\widehat{\boldsymbol{y}}$ như mong muốn. Chính xác, hàm softmax được định nghĩa như sau:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}. \tag{6.30}$$

Giống như với hàm sigmoid logistic, việc sử dụng hàm mũ hoạt động rất tốt khi huấn luyện hàm softmax để tạo ra giá trị mục tiêu y bằng cách sử dụng phương pháp cực đại logarit hàm hợp lý. Trong trường hợp này, ta muốn cực đại hóa $\log P(y = i; \boldsymbol{z}) = \log \text{softmax}(\boldsymbol{z})_i$. Định nghĩa softmax bằng cách sử dụng hàm mũ là hợp lý vì phép lấy log trong logarit hàm hợp lý có thể loại bỏ hàm mũ của softmax:

$$\log \text{softmax}(\boldsymbol{z})_i = z_i - \log \sum_j e^{z_j}. \tag{6.31}$$

Hạng tử đầu tiên trong phương trình (6.31) cho thấy đầu vào z_i luôn đóng góp trực tiếp vào hàm chi phí. Vì thành phần này không bão hòa, ta biết rằng quá trình học vẫn có thể tiếp tục, ngay cả khi đóng góp của z_i vào hạng tử thứ hai của phương trình (6.31) trở nên rất nhỏ. Khi cực đại hóa logarit hàm hợp lý, hạng tử đầu tiên khuyến khích z_i tăng lên, trong khi hạng tử thứ hai khuyến khích tất cả các giá trị trong $\boldsymbol{z}$ giảm xuống. Để hiểu trực giác của hạng tử thứ hai, $\log \sum_j e^{z_j}$, hãy quan sát rằng nó có thể được xấp xỉ bởi $\max_j z_j$. Xấp xỉ này dựa trên ý tưởng rằng e^{z_k} trở nên không đáng kể đối với bất kỳ z_k nào nhỏ hơn rõ rệt so với $\max_j z_j$. Trực giác mà ta có thể thu được từ xấp xỉ này là hàm chi phí đối của logarit hàm hợp lý luôn phạt mạnh nhất đối với dự đoán sai có giá trị kích hoạt lớn nhất. Nếu câu trả lời đúng đã có giá trị đầu vào lớn nhất cho softmax, thì hai thành phần $-z_i$ và $\log \sum_j e^{z_j} \approx \max_j z_j = z_i$ sẽ gần như triệt tiêu lẫn nhau. Khi đó, ví dụ này sẽ chỉ đóng góp rất ít vào tổng chi phí huấn luyện, chi phí sẽ chủ yếu bị chi phối bởi các ví dụ khác chưa được phân loại chính xác.

Cho đến thời điểm này, chúng ta chỉ thảo luận về một ví dụ duy nhất. Nhìn chung, khi không có điều chuẩn, phương pháp hợp lý cực đại sẽ thúc đẩy mô hình học các tham số sao cho softmax dự đoán được tỉ lệ số lần xuất hiện của từng kết quả quan sát được trong tập huấn luyện:

$$\text{softmax}\left(\boldsymbol{z}\left(\boldsymbol{x}; \boldsymbol{\theta}\right)\right)_i \approx \frac{\sum_{j=1}^{m} \mathbf{1}_{y^{(j)}=i, \boldsymbol{x}^{(j)}=\boldsymbol{x}}}{\sum_{j=1}^{m} \mathbf{1}_{\boldsymbol{x}^{(j)}=\boldsymbol{x}}}. \tag{6.32}$$

Vì phương pháp hợp lý cực đại là một bộ ước lượng vững, điều này được đảm bảo xảy ra miễn là không gian mô hình có năng lực biểu diễn được phân phối của tập huấn luyện. Trong thực tế, năng lực mô hình hạn chế và quá trình tối ưu hóa không hoàn hảo sẽ khiến mô hình chỉ có thể xấp xỉ các tỉ lệ này.

Nhiều hàm mục tiêu khác ngoài logarit hàm hợp lý không hoạt động tốt với hàm softmax. Đặc biệt, các hàm mục tiêu không sử dụng log để triệt tiêu hàm mũ trong softmax sẽ thất bại khi đối số của hàm mũ trở nên rất âm, làm cho gradient bằng 0. Cụ thể, hàm sai số bình phương là một hàm mất mát kém hiệu quả đối với các đơn vị softmax và có thể không huấn luyện được mô hình để thay đổi đầu ra, ngay cả khi mô hình đưa ra các dự đoán sai với độ tin cậy rất cao (*Alpha-nets: a recurrent "neural" network architecture with a hidden Markov model interpretation*, Bridle, 1990, [4]). Để hiểu tại sao các hàm mất mát này có thể thất bại, ta cần xem xét chính hàm softmax.

Giống như sigmoid, hàm kích hoạt softmax có thể bão hòa. Hàm sigmoid có một đầu ra duy nhất và bão hòa khi đầu vào của nó cực kỳ âm hoặc cực kỳ dương. Trong trường hợp của softmax, có nhiều giá trị đầu ra. Các giá trị đầu ra này có thể bão hòa khi chênh lệch giữa các giá trị đầu vào trở nên cực kỳ lớn. Khi softmax bão hòa, nhiều hàm chi phí dựa trên softmax cũng bão hòa, trừ khi chúng có khả năng nghịch đảo được hàm kích hoạt bão hòa này.

Để thấy rằng hàm softmax phản ứng với sự chênh lệch giữa các đầu vào của nó, ta quan sát rằng đầu ra của softmax không thay đổi khi cộng một số vô hướng giống nhau vào tất cả các đầu vào:

$$\text{softmax}(z) = \text{softmax}(z + c). \tag{6.33}$$

Dựa vào tính chất này, ta có thể xây dựng một biến thể ổn định về tính toán số của softmax:

$$\text{softmax}(\boldsymbol{z}) = \text{softmax}\left(\boldsymbol{z} - \max_i z_i\right). \tag{6.34}$$

Phiên bản cải tiến này cho phép chúng ta tính toán softmax với sai số tính toán nhỏ, ngay cả khi z chứa các giá trị rất lớn hoặc rất âm. Khi quan sát phiên bản ổn định tính toán số, ta nhận thấy rằng hàm softmax bị chi phối bởi độ lệch của các đối số của nó so với $\max_i z_i$.

Một đầu ra $\text{softmax}(\boldsymbol{z})_i$ bão hòa đến 1 khi đầu vào tương ứng là lớn nhất ($z_i = \max_i z_i$) và z_i lớn hơn nhiều so với tất cả các đầu vào khác. Ngược lại, đầu ra $\text{softmax}(\boldsymbol{z})_i$ cũng có thể bão hòa đến 0 khi z_i không phải là giá trị lớn nhất và giá trị lớn nhất vượt trội hơn rất nhiều. Điều này là một sự tổng quát của cách mà các

đơn vị sigmoid bão hòa và có thể gây ra các khó khăn tương tự trong việc học nếu hàm mất mát không được thiết kế để bù đắp cho hiện tượng này.

Đối số $\boldsymbol{z}$ của hàm softmax có thể được tạo ra theo hai cách khác nhau. *Cách thông thường*: Một lớp trước của mạng nơron tính toán từng phần tử của $\boldsymbol{z}$, như đã mô tả với lớp tuyến tính $\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}$. Cách này rất phổ biến nhưng thực tế lại dẫn đến việc tham số hóa thừa cho phân phối. Ràng buộc rằng n đầu ra phải có tổng bằng 1 nghĩa là chỉ cần $n-1$ tham số là đủ; xác suất của giá trị thứ n có thể được tính bằng cách lấy 1 trừ đi tổng của $n-1$ xác suất đầu tiên. Do đó, ta có thể áp đặt ràng buộc một phần tử của $\boldsymbol{z}$ cố định, ví dụ, đặt $z_n = 0$. Điều này tương tự như cách mà đơn vị sigmoid được định nghĩa: $P\left(y = 1 \mid \boldsymbol{x}\right) = \sigma\left(z\right)$ tương đương với $P\left(y = 1 \mid \boldsymbol{x}\right) = \text{softmax}\left(\boldsymbol{z}\right)_1$ với $\boldsymbol{z}$ có hai chiều và $z_1 = 0$. *Cách tham số hóa có hạn chế*: Chỉ sử dụng $n-1$ tham số để mô tả phân phối, trong khi cách thông thường sử dụng n tham số. Mặc dù cả hai cách đều có thể mô tả cùng một tập hợp các phân phối xác suất, chúng có động lực học học khác nhau. Tuy nhiên, trong thực tế, sự khác biệt giữa hai cách này hiếm khi đáng kể, và cách tham số hóa thừa triển khai đơn giản hơn.

Từ góc nhìn thần kinh học, hàm softmax có thể được xem như một cơ chế tạo ra sự cạnh tranh giữa các đơn vị tham gia vào nó. Vì đầu ra của softmax luôn có tổng bằng 1, nên việc tăng giá trị của một đơn vị sẽ kéo theo việc giảm giá trị của các đơn vị khác. Điều này tương tự với ức chế bên, được cho là tồn tại giữa các nơron gần nhau trong vỏ não. Khi sự khác biệt giữa giá trị lớn nhất z_i và các giá trị khác rất lớn, softmax tiến gần đến cơ chế **người thắng được tất cả**, nơi một đầu ra gần như bằng 1 còn các đầu ra khác gần như bằng 0.

Tên gọi "softmax" có thể gây nhầm lẫn vì hàm này liên quan mật thiết hơn đến hàm arg max thay vì hàm max. Thuật ngữ "soft" xuất phát từ việc softmax là một hàm liên tục và khả vi, trong khi hàm arg max, với kết quả được biểu diễn dưới dạng véctơ one–hot, không liên tục và không khả vi. Do đó, softmax cung cấp một phiên bản "mềm hóa" của arg max. Ngoài ra, phiên bản "mềm" tương ứng của hàm max là $\text{softmax}\left(\boldsymbol{z}\right)^\top \boldsymbol{z}$. Có lẽ sẽ hợp lý hơn nếu gọi hàm softmax là "softargmax", nhưng tên gọi hiện tại đã trở thành một quy ước được sử dụng rộng rãi.

6.2.2.4 Các loại đầu ra khác

Các đơn vị đầu ra tuyến tính, sigmoid và softmax được mô tả ở trên là những loại phổ biến nhất. Mạng nơron có thể được tổng quát hóa để sử dụng hầu như bất kỳ loại lớp đầu ra nào mà chúng ta mong muốn. Nguyên lý hợp lý cực đại cung cấp

một hướng dẫn để thiết kế hàm mất mát phù hợp cho hầu hết các loại lớp đầu ra.

Nói chung, nếu ta định nghĩa một phân phối có điều kiện $p\left(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}\right)$, nguyên lý hợp lý cực đại gợi ý rằng ta nên sử dụng $-\log p\left(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}\right)$ làm hàm mất mát.

Nhìn chung, chúng ta có thể coi mạng nơron như việc biểu diễn một hàm $f\left(\boldsymbol{x}; \boldsymbol{\theta}\right)$. Các đầu ra của hàm này không phải là dự đoán trực tiếp của giá trị $\boldsymbol{y}$. Thay vào đó, $f\left(\boldsymbol{x}; \boldsymbol{\theta}\right) = \boldsymbol{\omega}$ cung cấp các tham số cho một phân phối trên $\boldsymbol{y}$. Khi đó, hàm mất mát của chúng ta có thể được diễn giải là $-\log p\left(\boldsymbol{y}; \boldsymbol{\omega}\left(\boldsymbol{x}\right)\right)$.

Chẳng hạn, chúng ta có thể muốn học phương sai của một phân phối Gauss có điều kiện cho $\boldsymbol{y}$, với điều kiện x. Trong trường hợp đơn giản, khi phương sai σ^2 là hằng số, có thể tìm được biểu thức dưới dạng tường minh vì ước lượng hợp lý cực đại của phương sai đơn giản là giá trị trung bình thực nghiệm của bình phương sai lệch giữa các quan sát $\boldsymbol{y}$ và giá trị kỳ vọng của chúng. Một cách tiếp cận tính toán tốn kém hơn nhưng không yêu cầu viết mã xử lý trường hợp đặc biệt là chỉ cần bao gồm phương sai như một trong các đặc tính của phân phối $p\left(\boldsymbol{y} \mid \boldsymbol{x}\right)$, được điều khiển bởi $\boldsymbol{\omega} = f\left(\boldsymbol{x}; \boldsymbol{\theta}\right)$. Hàm đối của logarit hàm hợp lý $-\log p\left(\boldsymbol{y}; \boldsymbol{\omega}\left(\boldsymbol{x}\right)\right)$ khi đó sẽ cung cấp một hàm chi phí với các hạng tử cần thiết để quy trình tối ưu có thể dần học được phương sai. Trong trường hợp đơn giản mà độ lệch chuẩn không phụ thuộc vào đầu vào, ta có thể tạo một tham số mới trong mạng, tham số này được gán trực tiếp vào $\boldsymbol{\omega}$. Tham số mới này có thể là chính σ, hoặc là một tham số v đại diện cho σ^2, hoặc là một tham số β đại diện cho $\frac{1}{\sigma^2}$, tùy thuộc vào cách ta lựa chọn biểu diễn tham số cho phân phối. Ta cũng có thể muốn mô hình dự đoán mức độ phương sai khác nhau của $\boldsymbol{y}$ cho các giá trị $\boldsymbol{x}$ khác nhau. Đây được gọi là mô hình **dị phương sai**. Trong trường hợp dị phương sai, chúng ta chỉ cần làm cho phương sai trở thành một trong các giá trị được xuất ra bởi $f\left(\boldsymbol{x}; \boldsymbol{\theta}\right)$. Một cách phổ biến để làm điều này là xây dựng phân phối Gauss sử dụng độ chính xác, thay vì phương sai, như được mô tả trong phương trình (3.25).

Trong trường hợp đa biến, cách thông dụng nhất là sử dụng ma trận độ chính xác chéo:

$$\text{diag}\left(\boldsymbol{\beta}\right). \tag{6.35}$$

Biểu diễn này hoạt động tốt với phương pháp hướng giảm vì công thức của logarit hàm hợp lý cho phân phối Gauss được tham số hóa bởi $\boldsymbol{\beta}$ chỉ bao gồm phép nhân với β_i và phép cộng với $\log \boldsymbol{\beta}_i$. Gradient của các phép toán nhân, cộng và logarit đều ổn định. Ngược lại, nếu chúng ta tham số hóa đầu ra theo phương sai, cần sử dụng phép chia. Hàm chia trở nên cực kỳ dốc khi tiến gần đến 0. Dù gradient lớn có thể hỗ trợ quá trình học, nhưng gradient cực kỳ lớn thường dẫn đến

bất ổn. Nếu tham số hóa đầu ra theo độ lệch chuẩn, logarit hàm hợp lý vẫn liên quan đến phép chia, và thêm vào đó là phép bình phương. Gradient qua phép bình phương có thể tiến đến 0 gần giá trị 0, khiến việc học các tham số bị bình phương trở nên khó khăn. Dù sử dụng độ lệch chuẩn, phương sai hay độ chính xác, ta phải đảm bảo rằng ma trận hiệp phương sai của phân phối Gauss là xác định dương. Vì các giá trị riêng của ma trận độ chính xác là nghịch đảo của các giá trị riêng của ma trận hiệp phương sai, điều này tương đương với việc đảm bảo rằng ma trận độ chính xác là xác định dương. Nếu ta sử dụng ma trận đường chéo hoặc một hằng số nhân với ma trận đường chéo, điều kiện duy nhất cần áp đặt lên đầu ra của mô hình là dương. Giả sử $\boldsymbol{a}$ là kích hoạt thô của mô hình dùng để xác định độ chính xác dạng đường chéo, ta có thể sử dụng hàm softplus để tạo ra một véctơ độ chính xác giá trị dương: $\boldsymbol{\beta} = \zeta(\boldsymbol{a})$. Chiến lược này cũng áp dụng tương tự nếu sử dụng phương sai hoặc độ lệch chuẩn thay vì độ chính xác, hoặc nếu sử dụng một hằng số nhân với ma trận đơn vị thay vì ma trận đường chéo.

Việc học một ma trận hiệp phương sai hoặc ma trận độ chính xác có cấu trúc phức tạp hơn ma trận đường chéo là rất hiếm. Nếu hiệp phương sai là dạng đầy đủ và có điều kiện, thì cần lựa chọn một cách tham số hóa đảm bảo tính xác định dương của ma trận hiệp phương sai được dự đoán. Điều này có thể được thực hiện bằng cách biểu diễn $\boldsymbol{\Sigma}(\boldsymbol{x}) = \boldsymbol{B}(\boldsymbol{x})\boldsymbol{B}^{\top}(\boldsymbol{x})$, trong đó B là một ma trận vuông không có ràng buộc. Nếu ma trận có hạng đầy đủ, việc tính hàm hợp lý sẽ trở nên tốn kém, vì với một ma trận cỡ $d \times d$, cần $O\left(d^3\right)$ phép tính để tính định thức và ma trận nghịch đảo của $\boldsymbol{\Sigma}(\boldsymbol{x})$ (hoặc tương đương, phổ biến hơn là thực hiện phân tích giá trị riêng $\boldsymbol{B}(\boldsymbol{x})$).

Trong nhiều trường hợp, ta muốn thực hiện hồi quy đa mô hình, tức là dự đoán các giá trị thực từ một phân phối có điều kiện $p\left(\boldsymbol{y} \mid \boldsymbol{x}\right)$, trong đó phân phối này có thể có nhiều đỉnh khác nhau trong không gian $\boldsymbol{y}$ với cùng một giá trị $\boldsymbol{x}$. Trong trường hợp này, phân phối Gauss hỗn hợp là một cách biểu diễn tự nhiên cho đầu ra. (*Adaptive Mixtures of Local Experts*, Jacobs và cộng sự, 1991, [5]; *Mixture Nensity Networks*, Bishop, 1994, [6]). Các mạng nơron có đầu ra có phân phối Gauss hỗn hợp thường được gọi là *mạng mật độ hỗn hợp*. Một đầu ra có phân phối Gauss hỗn hợp với n thành phần được định nghĩa bởi phân phối xác suất có điều kiện:

$$p\left(\boldsymbol{y} \mid \boldsymbol{x}\right) = \sum_{i=1}^{n} p\left(c = i \mid \boldsymbol{x}\right) N\left(\boldsymbol{y}; \boldsymbol{\mu}^{(i)}(\boldsymbol{x}), \boldsymbol{\Sigma}^{(i)}(\boldsymbol{x})\right). \tag{6.36}$$

Mạng nơ-ron phải có ba đầu ra: một véctơ định nghĩa $p\left(c = i \mid \boldsymbol{x}\right)$, một ma trận cung cấp $\boldsymbol{\mu}^{(i)}(x)$ với mọi i, và một tenxơ cung cấp $\boldsymbol{\Sigma}^{(i)}(\boldsymbol{x})$ với mọi i. Các đầu

ra này phải thỏa mãn các ràng buộc sau:

1. Thành phần hỗn hợp $p\left(c = i \mid x\right)$: Đây là một phân phối Bernoulli bội trên n thành phần khác nhau liên kết với biến tiềm ẩn* c, và thường được tính bằng cách áp dụng hàm softmax lên một véctơ n chiều, để đảm bảo các đầu ra này luôn dương và tổng bằng 1.

2. Các giá trị trung bình $\boldsymbol{\mu}^{(i)}(\boldsymbol{x})$: Đây là các trung bình hoặc tâm liên kết với thành phần phân phối Gauss thứ i, không bị ràng buộc (thường không áp dụng hàm phi tuyến nào cho các đơn vị đầu ra này). Nếu $\boldsymbol{y}$ là một véctơ d chiều, mạng phải đưa ra một ma trận $n \times d$ chứa tất cả n véctơ d chiều này. Việc học các trung bình này theo phương pháp hợp lý cực đại phức tạp hơn một chút so với việc học trung bình của một phân phối chỉ có một chế độ đầu ra. Ta chỉ muốn cập nhật trung bình của phân phối thành phần đã thực sự sinh ra quan sát. Trong thực tế, ta không biết phân phối thành phần nào đã sinh ra mỗi quan sát. Biểu thức của đối của logarit hàm hợp lý tự nhiên gán trọng số cho sự đóng góp của mỗi ví dụ vào hàm mất mát của từng phân phối thành phần dựa trên xác suất rằng phân phối thành phần đó đã sinh ra ví dụ.

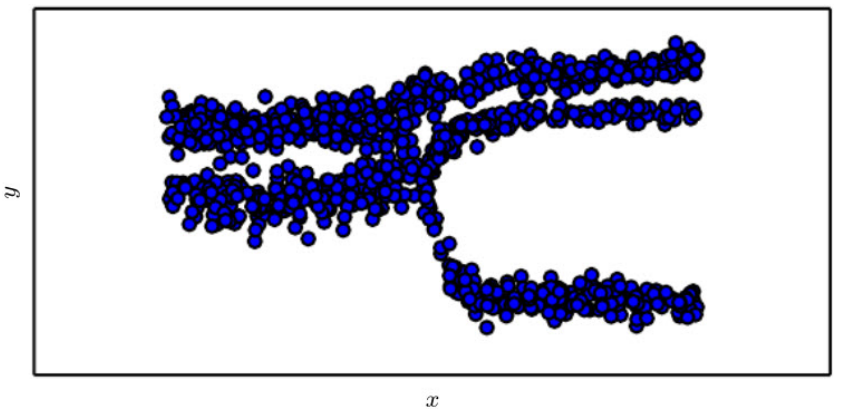
3. Các ma trận hiệp phương sai $\boldsymbol{\Sigma}^{(i)}(\boldsymbol{x})$: Đây là các ma trận hiệp phương sai của mỗi phân phối thành phần i. Khi học một phân phối thành phần Gauss đơn lẻ, ta thường sử dụng ma trận đường chéo để tránh tính các định thức. Tương tự, khi học các trung bình của phân phối hỗn hợp, phương pháp hợp lý cực đại phức tạp hơn do cần gán đóng góp riêng của mỗi điểm vào từng phân phối thành phần trong hỗn hợp. Phương pháp hướng giảm tự động tuân theo quy trình đúng nếu được cung cấp một cách đặc tả chính xác về hàm đối của logarit hàm hợp lý theo mô hình phân phối hỗn hợp.

Người ta đã báo cáo rằng việc tối ưu dựa trên gradient của các phân phối Gauss hỗn hợp có điều kiện (cho đầu ra của mạng nơron) có thể không đáng tin cậy, một phần do các phép chia (bởi phương sai) có thể không ổn định về mặt tính toán số (khi một phương sai nào đó trở nên rất nhỏ đối với một ví dụ cụ thể, dẫn đến gradient rất lớn). Một giải pháp là **cắt gradient** (xem Mục 10.11.1), hoặc là

*Ta coi c là biến tiềm ẩn vì nó không được quan sát trong dữ liệu: với đầu vào $\boldsymbol{x}$ và đầu ra mục tiêu $\boldsymbol{y}$, không thể biết chắc chắn thành phần phân phối Gauss nào chịu trách nhiệm sinh ra $\boldsymbol{y}$. Tuy nhiên, ta có thể hình dung rằng $\boldsymbol{y}$ được tạo ra bằng cách chọn một trong các phân phối thành phần đó, và coi lựa chọn không được quan sát này là một biến ngẫu nhiên.

co giãn gradient theo cách thực nghiệm (*A Deep and Tractable Density Estimator*, Murray và Larochelle, 2014, [7]).

Đầu ra hỗn hợp Gauss đặc biệt hiệu quả trong các mô hình sinh cho giọng nói (Schuster, 1999) hoặc chuyển động của các vật thể vật lý (Graves, 2013). Chiến lược sử dụng mật độ hỗn hợp cung cấp một cách để mạng nơron biểu diễn nhiều chế độ đầu ra và kiểm soát phương sai của đầu ra, điều này rất quan trọng để đạt được chất lượng cao trong các miền có giá trị thực. Một ví dụ về mạng mật độ hỗn hợp được minh họa trong Hình 6.4.



Hình 6.4: Các mẫu được lấy từ một mạng nơron với lớp đầu ra là phân phối hỗn hợp. Đầu vào x được lấy mẫu từ một phân phối đều, và đầu ra y được lấy mẫu từ $p_{\text{model}}\left(y \mid x\right)$. Mạng nơron có khả năng học các ánh xạ phi tuyến từ đầu vào đến các tham số của phân phối đầu ra. Các tham số này bao gồm các xác suất điều khiển thành phần nào trong ba thành phần hỗn hợp sẽ sinh ra đầu ra, và các tham số cho từng thành phần hỗn hợp. Mỗi thành phần hỗn hợp là một phân phối Gauss với trung bình và phương sai được dự đoán. Tất cả các khía cạnh này của phân phối đầu ra đều có thể thay đổi theo đầu vào x, và thay đổi một cách phi tuyến.

Nói chung, ta có thể muốn tiếp tục mô hình hóa các véctơ $\boldsymbol{y}$ lớn hơn, chứa nhiều biến hơn, và áp đặt cấu trúc ngày càng phong phú lên các biến đầu ra này. Ví dụ, ta có thể muốn mạng nơron tạo ra một chuỗi ký tự hình thành nên một câu. Trong các trường hợp này, chúng ta có thể tiếp tục áp dụng nguyên lý hợp lý cực đại cho mô hình $p\left(\boldsymbol{y}; \boldsymbol{\omega}\left(\boldsymbol{x}\right)\right)$, nhưng mô hình được sử dụng để mô tả $\boldsymbol{y}$ trở nên phức tạp đến mức vượt ra ngoài phạm vi của chương này. Chương 10 trình bày cách sử dụng mạng nơron hồi quy (RNN) để định nghĩa các mô hình như vậy trên các chuỗi, và Phần III mô tả các kỹ thuật tiên tiến để mô hình hóa các phân phối xác suất bất kỳ.

6.3 Các đơn vị ẩn

Cho đến nay, chúng ta đã tập trung thảo luận về các lựa chọn thiết kế cho mạng nơron, vốn phổ biến đối với hầu hết các mô hình học máy có tham số được huấn luyện bằng tối ưu dựa trên gradient. Bây giờ, chúng ta chuyển sang một vấn đề đặc thù của mạng nơron lan truyền thẳng: cách chọn loại đơn vị ẩn để sử dụng trong các lớp ẩn của mô hình.

Thiết kế các đơn vị ẩn là một lĩnh vực nghiên cứu rất sôi động và hiện tại vẫn chưa có nhiều nguyên tắc lý thuyết chắc chắn để hướng dẫn.

Đơn vị tuyến tính chỉnh lưu là một lựa chọn mặc định rất tốt cho các đơn vị ẩn. Cũng có nhiều loại đơn vị ẩn khác. Việc xác định khi nào nên sử dụng loại nào có thể khá khó khăn (mặc dù đơn vị tuyến tính chỉnh lưu thường là một lựa chọn chấp nhận được). Dưới đây, chúng tôi mô tả một số trực giác cơ bản thúc đẩy việc sử dụng từng loại đơn vị ẩn. Những trực giác này có thể giúp bạn quyết định khi nào nên thử sử dụng từng loại đơn vị. Tuy nhiên, thường rất khó để dự đoán trước loại đơn vị nào sẽ hoạt động tốt nhất. Quy trình thiết kế bao gồm thử và đo sai số, suy đoán rằng một loại đơn vị ẩn có thể phù hợp, sau đó huấn luyện một mạng với loại đơn vị ẩn đó và đánh giá hiệu suất của nó trên tập kiểm định.

Một số đơn vị ẩn được liệt kê trong danh sách thực tế không khả vi tại mọi điểm đầu vào. Ví dụ, hàm tuyến tính chỉnh lưu $g(z) = \max\{0, z\}$ không khả vi tại $z = 0$. Điều này dường như làm cho g không phù hợp để sử dụng với các thuật toán học dựa trên gradient. Tuy nhiên, trong thực tế, phương pháp hướng giảm vẫn hoạt động đủ tốt để các mô hình này được áp dụng cho các nhiệm vụ học máy. Điều này một phần là do các thuật toán huấn luyện mạng nơron thường không đạt được cực tiểu địa phương của hàm mất mát, mà thay vào đó chỉ đơn giản là giảm giá trị của nó một cách đáng kể, như minh họa trong Hình 4.3. Những ý tưởng này sẽ được trình bày chi tiết hơn ở Chương 8. Bởi vì chúng ta không kỳ vọng việc huấn luyện thực sự đạt đến một điểm mà gradient bằng $\mathbf{0}$, nên việc các cực tiểu của hàm mất mát tương ứng với các điểm có gradient không xác định là chấp nhận được. Các đơn vị ẩn không khả vi thường chỉ không khả vi tại một số lượng nhỏ các điểm. Nói chung, một hàm $g(z)$ có đạo hàm trái được xác định bởi độ dốc của hàm ngay phía bên trái của z và đạo hàm phải được xác định bởi độ dốc của hàm ngay phía bên phải của z. Một hàm khả vi tại z chỉ khi cả đạo hàm trái và đạo hàm phải đều xác định và bằng nhau. Các hàm được sử dụng trong mạng nơron thường có cả đạo hàm trái và đạo hàm phải xác định. Trong trường hợp $g(z) = \max\{0, z\}$, đạo hàm trái tại $z = 0$ là 0 và đạo hàm phải là 1. Các phần mềm triển khai huấn luyện mạng

nơron thường trả về một trong hai đạo hàm một phía này thay vì báo lỗi hoặc thông báo rằng đạo hàm không xác định. Điều này có thể được biện minh một cách kinh nghiệm bằng cách quan sát rằng tối ưu dựa trên gradient trên máy tính kỹ thuật số luôn chịu ảnh hưởng của sai số tính toán. Khi một hàm được yêu cầu tính $g(0)$, rất hiếm khi giá trị thực sự của đối số là 0. Thay vào đó, nó có khả năng là một giá trị nhỏ ε nào đó được làm tròn về 0. Trong một số ngữ cảnh, có những biện minh lý thuyết thỏa đáng hơn, nhưng thường không áp dụng cho việc huấn luyện mạng nơron. Điểm quan trọng là trong thực tế, người ta có thể bỏ qua một cách an toàn vấn đề không khả vi của các hàm kích hoạt đơn vị ẩn được mô tả dưới đây.

Nếu không có chỉ định khác, hầu hết các đơn vị ẩn có thể được mô tả như sau: chúng nhận một véctơ đầu vào $\boldsymbol{x}$, tính một phép biến đổi affine $\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}$, sau đó áp dụng một hàm phi tuyến lên từng phần tử $g(z)$. Điểm khác biệt chính giữa các đơn vị ẩn thường nằm ở lựa chọn dạng của hàm kích hoạt $g(z)$.

6.3.1 Đơn vị tuyến tính chỉnh lưu và các mở rộng

Đơn vị tuyến tính chỉnh lưu (ReLU) sử dụng hàm kích hoạt $g(z) = \max\{0, z\}$.

ReLU dễ tối ưu hóa bởi vì chúng rất giống với các đơn vị tuyến tính. Sự khác biệt duy nhất giữa một đơn vị tuyến tính và một đơn vị tuyến tính chỉnh lưu là ReLU trả về giá trị bằng 0 trên một nửa miền của nó. Điều này giúp đạo hàm qua ReLU luôn lớn bất cứ khi nào đơn vị này đang hoạt động. Không chỉ gradient lớn, mà chúng còn ổn định. Đạo hàm cấp hai của phép chỉnh lưu bằng 0 ở hầu khắp nơi, và đạo hàm cấp một của nó bằng 1 ở mọi nơi mà đơn vị đang hoạt động. Điều này có nghĩa là hướng gradient hữu ích hơn rất nhiều cho việc học so với các hàm kích hoạt khác có hiệu ứng cấp hai.

ReLU thường được sử dụng trên một phép biến đổi affine:

$$h = g\left(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}\right). \tag{6.37}$$

Khi khởi tạo các tham số của phép biến đổi affine, một thực hành tốt là đặt tất cả các phần tử của $\boldsymbol{b}$ là một giá trị dương nhỏ, chẳng hạn như 0.1. Điều này làm tăng khả năng các ReLU ban đầu sẽ hoạt động với hầu hết các đầu vào trong tập huấn luyện, cho phép các đạo hàm truyền qua.

Có nhiều mở rộng của ReLU đã được phát triển. Hầu hết các mở rộng này có hiệu suất tương đương với ReLU và đôi khi hoạt động tốt hơn.

Một nhược điểm của ReLU là chúng không thể học bằng các phương pháp dựa trên gradient đối với các ví dụ mà kích hoạt của chúng bằng 0. Nhiều mở rộng của

ReLU đã được đề xuất để đảm bảo rằng chúng nhận được gradient ở mọi nơi.

Ba mở rộng của đơn vị tuyến tính chỉnh lưu dựa trên việc sử dụng một độ dốc khác 0, α_i, khi $z_i < 0$: $h_i = g(\boldsymbol{z}, \boldsymbol{\alpha})_i = \max(0, z_i) + \alpha_i \min(0, z_i)$. **Chỉnh lưu giá trị tuyệt đối** cố định $\alpha_i = -1$ để thu được $g(z) = |z|$. Cách tiếp cận này được sử dụng trong nhận dạng đối tượng từ hình ảnh (*What is the Best Multi-Stage Architecture for Object Recognition?*, Jarrett và cộng sự, 2009, [1]), nơi việc tìm kiếm các đặc trưng bất biến với sự đảo ngược cực tính của ánh sáng đầu vào là hợp lý. Các mở rộng khác của ReLU có thể áp dụng rộng rãi hơn. **ReLU rò rỉ** (*Rectifier Nonlinearities Improve Neural Network Acoustic Models*, Maas và cộng sự, 2013, [8]), α_i được cố định ở một giá trị nhỏ như 0.01, trong khi **ReLU có tham số** (PReLU) coi α_i là một tham số có thể học được, cho phép mô hình tự động điều chỉnh độ dốc khi $z_i < 0$. (*Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, He và cộng sự, 2015, [9]).

Đơn vị maxout (*Maxout Networks*, Goodfellow và cộng sự, 2013, [10]) mở rộng ReLU thêm một bước nữa. Thay vì áp dụng một hàm từng phần tử $g(z)$, đơn vị maxout chia $\boldsymbol{z}$ thành các nhóm k giá trị. Mỗi đơn vị maxout sau đó trả về giá trị lớn nhất trong một nhóm:

$$g(\boldsymbol{z})_i = \max_{j \in G^{(i)}} z_j \tag{6.38}$$

với $G^{(i)}$ là tập hợp các chỉ số vào của nhóm i, cụ thể là $\{(i-1)k+1, \ldots, ik\}$. Cách tiếp cận này cung cấp một phương pháp để học một hàm phân đoạn tuyến tính, có khả năng phản hồi với nhiều hướng khác nhau trong không gian đầu vào $\boldsymbol{x}$.

Một đơn vị maxout có thể học một hàm lồi, tuyến tính từng đoạn với tối đa k đoạn. Do đó, các đơn vị maxout có thể được xem như *học chính hàm kích hoạt*, thay vì chỉ học mối quan hệ giữa các đơn vị. Với k đủ lớn, một đơn vị maxout có thể học để xấp xỉ bất kỳ hàm lồi nào với độ chính xác tùy ý. Đặc biệt, một lớp maxout với hai đoạn có thể học để thực hiện cùng một hàm của đầu vào $\boldsymbol{x}$ như một lớp truyền thống sử dụng hàm kích hoạt tuyến tính chỉnh lưu, hàm chỉnh lưu giá trị tuyệt đối, hoặc ReLU rò rỉ hay có tham số, hoặc cũng có thể học để thực hiện một hàm hoàn toàn khác. Tất nhiên, lớp maxout sẽ được tham số hóa khác với bất kỳ loại lớp nào khác, vì vậy động lực học học của nó cũng sẽ khác, ngay cả trong các trường hợp maxout học để thực hiện cùng một hàm của $\boldsymbol{x}$ như một trong các loại lớp khác.

Mỗi đơn vị maxout hiện được tham số hóa bằng k véctơ trọng số thay vì chỉ một, vì vậy các đơn vị maxout thường cần nhiều điều chuẩn hơn so với các đơn vị chỉnh lưu tuyến tính. Chúng có thể hoạt động tốt mà không cần điều chuẩn nếu

tập huấn luyện lớn và số đoạn trên mỗi đơn vị được giữ ở mức thấp (*Deep maxout neural networks for speech recognition*, Cai và cộng sự, 2013, [11]).

Các đơn vị maxout còn có một số lợi ích khác. Trong một số trường hợp, có thể đạt được một số lợi thế về thống kê và tính toán bằng cách yêu cầu ít tham số hơn. Cụ thể, nếu các đặc trưng được nắm bắt bởi n bộ lọc tuyến tính khác nhau có thể được tóm tắt mà không mất thông tin bằng cách lấy giá trị lớn nhất trong mỗi nhóm k đặc trưng, thì lớp tiếp theo có thể sử dụng ít trọng số hơn gấp k lần.

Vì mỗi đơn vị được điều khiển bởi nhiều bộ lọc, các đơn vị maxout có một số tính dư thừa giúp chúng chống lại hiện tượng gọi là **quên thảm họa**, trong đó mạng nơron quên cách thực hiện các tác vụ mà chúng đã được huấn luyện trước đây (*An Empirical Investigation of Catastrophic Forgeting in Gradient-Based Neural Networks*, Goodfellow và cộng sự, 2014, [12]).

Các đơn vị chỉnh lưu tuyến tính và tất cả các dạng mở rộng của chúng đều dựa trên nguyên tắc rằng các mô hình dễ tối ưu hơn nếu hành vi của chúng gần tuyến tính hơn. Nguyên tắc tổng quát này về việc sử dụng hành vi tuyến tính để đạt được tối ưu dễ dàng hơn cũng áp dụng trong các bối cảnh khác ngoài mạng tuyến tính sâu. Các mạng hồi quy có thể học từ các chuỗi và tạo ra một chuỗi trạng thái và đầu ra. Khi huấn luyện chúng, cần truyền thông tin qua nhiều bước thời gian, điều này trở nên dễ dàng hơn khi có một số phép tính tuyến tính (với một số đạo hàm theo hướng có độ lớn gần bằng 1). Một trong những kiến trúc mạng hồi quy hoạt động tốt nhất, LSTM, truyền thông tin qua thời gian thông qua phép cộng — một dạng đặc biệt đơn giản của kích hoạt tuyến tính như vậy. Điều này được thảo luận chi tiết hơn trong Mục 10.10.

6.3.2 Hàm logistic sigmoid và tangent hyperbolic

Trước khi các hàm kích hoạt tuyến tính chỉnh lưu được giới thiệu, hầu hết các mạng nơron sử dụng hàm kích hoạt logistic sigmoid

$$g(z) = \sigma(z) \tag{6.39}$$

hoặc hàm kích hoạt tangent hyperbolic:

$$g(z) = \tanh(z). \tag{6.40}$$

Hai hàm kích hoạt này có mối liên hệ chặt chẽ với nhau vì $\tanh(z) = 2\sigma(2z) - 1$.

Trước đây, chúng ta đã thấy các đơn vị sigmoid được sử dụng làm đầu ra để dự đoán xác suất rằng một biến nhị phân có giá trị là 1. Không giống như các đơn

vị tuyến tính từng đoạn, các đơn vị sigmoid có tính bão hòa trên hầu hết miền của chúng — chúng bão hòa ở giá trị cao khi z rất dương, bão hòa ở giá trị thấp khi z rất âm, và chỉ nhạy cảm mạnh với đầu vào khi z gần 0. Việc bão hòa rộng rãi của các đơn vị sigmoid có thể khiến việc học dựa trên gradient trở nên rất khó khăn. Vì lý do này, việc sử dụng chúng làm đơn vị ẩn trong các mạng truyền thẳng hiện nay không được khuyến khích. Tuy nhiên, việc sử dụng sigmoid làm đơn vị đầu ra vẫn phù hợp với học dựa trên gradient nếu một hàm chi phí thích hợp có thể triệt tiêu tính bão hòa của sigmoid trong lớp đầu ra.

Khi một hàm kích hoạt sigmoid bắt buộc phải được sử dụng, hàm tangent hyperbolic thường cho hiệu suất tốt hơn so với logistic sigmoid. Hàm này gần giống với hàm đồng nhất hơn vì $\tanh(0) = 0$ trong khi $\sigma(0) = \frac{1}{2}$. Do tanh giống với hàm đồng nhất trong lân cận điểm 0, việc huấn luyện một mạng nơron sâu $\widehat{y} = \boldsymbol{w}^\top \tanh\left(\boldsymbol{U}^\top \tanh\left(\boldsymbol{V}^\top \boldsymbol{x}\right)\right)$ giống với việc huấn luyện một mô hình tuyến tính $\widehat{y} = \boldsymbol{w}^\top \boldsymbol{U}^\top \boldsymbol{V}^\top \boldsymbol{x}$ miễn là các kích hoạt trong mạng được giữ ở mức nhỏ. Điều này giúp việc huấn luyện mạng tanh trở nên dễ dàng hơn.

Các hàm kích hoạt sigmoid phổ biến hơn trong các bối cảnh ngoài mạng lan truyền thẳng. Các mạng hồi quy, nhiều mô hình xác suất, và một số bộ mã hóa tự động có các yêu cầu bổ sung không cho phép sử dụng các hàm kích hoạt tuyến tính từng đoạn và khiến các đơn vị sigmoid trở nên hấp dẫn hơn mặc dù có những nhược điểm về tính bão hòa.

6.3.3 Các loại đơn vị ẩn khác

Nhiều loại đơn vị ẩn khác có thể được sử dụng nhưng ít được sử dụng thường xuyên hơn.

Nhìn chung, một loạt các hàm khả vi khác nhau hoạt động tốt. Nhiều hàm kích hoạt chưa được công bố cho kết quả tương đương với các hàm phổ biến. Ví dụ cụ thể, người ta đã thử nghiệm một mạng lan truyền thẳng sử dụng hàm $\boldsymbol{h} = \cos(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ trên tập dữ liệu MNIST và đạt tỉ lệ lỗi dưới 1%, cạnh tranh với kết quả đạt được khi sử dụng các hàm kích hoạt thông thường. Trong quá trình nghiên cứu và phát triển các kỹ thuật mới, việc thử nghiệm nhiều hàm kích hoạt khác nhau là phổ biến, và nhiều biến thể của các thực hành tiêu chuẩn thường cho hiệu suất tương đương. Điều này dẫn đến việc các loại đơn vị ẩn mới chỉ được công bố nếu chúng được chứng minh rõ ràng là mang lại cải thiện đáng kể. Các loại đơn vị ẩn mới có hiệu suất tương đương với các loại đã biết quá phổ biến đến mức trở nên không đáng chú ý.

Việc liệt kê tất cả các loại đơn vị ẩn đã xuất hiện trong tài liệu nghiên cứu là không thực tế. Dưới đây, chúng tôi điểm qua một vài loại đặc biệt hữu ích và khác biệt.

Một khả năng là không sử dụng hàm kích hoạt $g(z)$ nào cả, tương đương với việc sử dụng hàm đồng nhất làm hàm kích hoạt. Như đã thấy trước đây, một đơn vị tuyến tính có thể hữu ích khi được sử dụng làm đầu ra của mạng nơron. Nó cũng có thể được sử dụng làm đơn vị ẩn. Nếu tất cả các tầng của mạng nơron chỉ bao gồm các phép biến đổi tuyến tính, thì mạng tổng thể sẽ là một mạng tuyến tính. Tuy nhiên, việc để một số lớp của mạng chỉ tuyến tính là hoàn toàn chấp nhận được. Hãy xem xét một lớp mạng nơron với n đầu vào và p đầu ra, $\boldsymbol{h} = g\left(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}\right)$. Ta có thể thay thế nó bằng hai lớp, trong đó một lớp sử dụng ma trận trọng số U và lớp còn lại sử dụng ma trận trọng số V. Nếu lớp đầu tiên không có hàm kích hoạt, ta về cơ bản đã phân tích ma trận trọng số của lớp ban đầu dựa trên W. Phương pháp phân tích này tính toán $\boldsymbol{h} = g\left(\boldsymbol{V}^\top \boldsymbol{U}^\top \boldsymbol{x} + \boldsymbol{b}\right)$. Nếu $\boldsymbol{U}$ sinh ra q đầu ra, thì $\boldsymbol{U}$ và $\boldsymbol{V}$ kết hợp chỉ chứa $(n+p)q$ tham số, trong khi $\boldsymbol{W}$ chứa np tham số. Với q nhỏ, điều này có thể giảm đáng kể số lượng tham số. Tuy nhiên, nó phải trả giá là ràng buộc phép biến đổi tuyến tính trở thành ma trận có hạng thấp, nhưng các quan hệ có hạng thấp này thường là đủ. Do đó, các đơn vị ẩn tuyến tính mang lại một cách hiệu quả để giảm số lượng tham số trong mạng.

Đơn vị softmax thường được sử dụng làm đầu ra (như mô tả trong Mục 6.2.2.3), nhưng đôi khi cũng có thể được dùng làm đơn vị ẩn. Đơn vị softmax biểu diễn một cách tự nhiên cho một phân phối xác suất trên một biến rời rạc với k giá trị có thể có, vì vậy chúng có thể được sử dụng như một loại "công tắc". Loại đơn vị ẩn này thường chỉ được sử dụng trong các kiến trúc nâng cao hơn, nơi học được cách thao tác bộ nhớ, như mô tả ở Mục 10.12.

Một số loại đơn vị ẩn phổ biến khác:

- Đơn vị **hàm cơ sở xuyên tâm** (radial basis function, RBF): $h_i = \exp\left(-\frac{1}{\sigma_i^2}\|\boldsymbol{W}_{:,i} - \boldsymbol{x}\|^2\right)$. Hàm này trở nên hoạt động mạnh khi $\boldsymbol{x}$ tiến gần tới một mẫu $\boldsymbol{W}_{:,i}$. Do hàm bão hòa về 0 với hầu hết các giá trị $\boldsymbol{x}$, việc tối ưu hóa loại đơn vị này có thể rất khó khăn.

- **Softplus**: $g(a) = \zeta(a) = \log(1 + e^a)$. Đây là phiên bản làm trơn của hàm kích hoạt ReLU, được giới thiệu bởi *Incorporating Second-Order Functional Knowledge for Better Option Pricing* (Dugas và cộng sự, 2000, [13]) cho xấp xỉ hàm số và bởi *Rectified Linear Units Improve Restricted Boltzmann*

Machines (Nair và Hinton, 2010, [2]) cho các phân phối có điều kiện của các mô hình xác suất không định hướng. *Deep Sparse Rectifier Neural Networks* (Glorot và cộng sự, 2011, [3]) so sánh softplus và ReLU và thấy rằng ReLU cho kết quả tốt hơn. Việc sử dụng softplus thường không được khuyến khích. Loại đơn vị này cho thấy hiệu suất của các loại đơn vị ẩn có thể rất không trực quan — người ta có thể kỳ vọng softplus có lợi thế hơn ReLU do khả vi ở mọi nơi hoặc do ít bão hòa hơn, nhưng thực nghiệm không ủng hộ điều này.

- Tanh **Hard**: Hàm này có hình dạng tương tự như tanh và ReLU nhưng, không giống như ReLU, nó bị chặn bởi một khoảng, $g(a) = \max(-1, \min(1, a))$. Hàm này được giới thiệu bởi *Large Scale Machine Learning* (Collobert, 2004, [14]).

Thiết kế đơn vị ẩn vẫn là một lĩnh vực nghiên cứu tích cực, và nhiều loại đơn vị ẩn hữu ích vẫn chưa được phát hiện.

6.4 Thiết kế kiến trúc

Một trong những yếu tố quan trọng trong thiết kế mạng nơron là xác định kiến trúc. **Kiến trúc** ở đây đề cập đến cấu trúc tổng thể của mạng: số lượng đơn vị mà mạng cần có và cách mà các đơn vị này được kết nối với nhau.

Hầu hết các mạng nơron được tổ chức thành các nhóm các đơn vị được gọi là các lớp. Phần lớn các kiến trúc mạng sắp xếp các lớp này theo cấu trúc chuỗi, trong đó mỗi lớp là một hàm của lớp trước đó. Trong cấu trúc này, lớp đầu tiên được biểu diễn bởi:

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right), \tag{6.41}$$

lớp thứ hai được biểu diễn bởi:

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{x} + \boldsymbol{b}^{(2)}\right), \tag{6.42}$$

và tương tự cho các lớp tiếp theo.

Trong các kiến trúc dựa trên chuỗi, các yếu tố chính cần cân nhắc là chọn độ sâu của mạng (số lớp ẩn trong mạng) và độ rộng của mỗi lớp (số đơn vị trong mỗi lớp). Mạng có một lớp ẩn duy nhất là đủ để khớp với tập huấn luyện trong nhiều trường hợp. Tuy nhiên, mạng sâu hơn thường cần ít đơn vị hơn trong mỗi lớp, dẫn đến tổng số tham số ít hơn và thường tổng quát hóa tốt hơn trên tập kiểm tra. Tuy nhiên, mạng sâu hơn cũng thường khó tối ưu hóa hơn. Kiến trúc mạng lý tưởng cho

một bài toán cụ thể thường phải được tìm kiếm thông qua thực nghiệm, dựa vào việc giám sát sai số trên tập kiểm định.

6.4.1 Tính chất xấp xỉ phổ quát và độ sâu

Một mô hình tuyến tính, ánh xạ từ đặc trưng đầu vào đến đầu ra thông qua phép nhân ma trận, theo định nghĩa chỉ có thể biểu diễn các hàm tuyến tính. Mô hình này có lợi thế là dễ huấn luyện vì nhiều hàm mất mát dẫn đến các bài toán tối ưu lồi khi áp dụng cho mô hình tuyến tính. Tuy nhiên, trong thực tế, ta thường muốn học các hàm phi tuyến.

Thoạt nhìn, chúng ta có thể cho rằng việc học một hàm phi tuyến đòi hỏi phải thiết kế một họ mô hình chuyên biệt phù hợp với loại phi tuyến mà ta muốn học. May mắn thay, các mạng lan truyền thẳng với các lớp ẩn cung cấp một khung lý thuyết xấp xỉ phổ quát. Cụ thể, **định lý xấp xỉ phổ quát** (*Multilayer feedforward networks are universal approximators*, Hornik và cộng sự, 1989, [15]; *Approximation by Superpositions of a Sigmoidal Function*, Cybenko, 1989, [16]) phát biểu rằng một mạng lan truyền thẳng với lớp đầu ra tuyến tính và ít nhất một lớp ẩn sử dụng bất kỳ hàm kích hoạt dạng "nén" nào (chẳng hạn như hàm kích hoạt sigmoid logistic) có thể xấp xỉ bất kỳ hàm đo được Borel nào từ một không gian hữu hạn chiều này đến một không gian hữu hạn chiều khác với bất kỳ sai số khác không nào, miễn là mạng được cung cấp đủ số lượng đơn vị ẩn. Các đạo hàm của mạng lan truyền thẳng cũng có thể xấp xỉ các đạo hàm của hàm mục tiêu một cách chính xác tùy ý (*Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks*, Hornik và cộng sự, 1990, [17]). Khái niệm về đo được Borel nằm ngoài phạm vi của cuốn sách này; đối với mục đích của chúng ta, chỉ cần hiểu rằng bất kỳ hàm liên tục nào trên một tập con đóng và bị chặn của $\mathbb{R}^n$ đều đo được Borel và do đó có thể được xấp xỉ bởi một mạng nơron. Một mạng nơron cũng có thể xấp xỉ bất kỳ hàm nào ánh xạ từ một không gian rời rạc hữu hạn chiều này đến một không gian rời rạc hữu hạn chiều khác. Mặc dù các định lý ban đầu được phát biểu dựa trên các đơn vị có hàm kích hoạt bão hòa cả ở đầu vào rất âm và rất dương, các định lý xấp xỉ phổ quát cũng đã được chứng minh cho một lớp hàm kích hoạt rộng hơn, bao gồm cả hàm kích hoạt tuyến tính chỉnh lưu hiện nay được sử dụng phổ biến (*Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function*, Leshno và cộng sự, 1993, [18]).

Định lý xấp xỉ phổ quát có nghĩa là bất kể hàm nào chúng ta đang cố gắng học,

chúng ta đều biết rằng một mạng lan truyền thẳng đa lớp cỡ lớn sẽ có khả năng *biểu diễn* hàm đó. Tuy nhiên, điều này không đảm bảo rằng thuật toán huấn luyện sẽ có thể *học* được hàm đó. Ngay cả khi mạng lan truyền thẳng đa lớp có khả năng biểu diễn hàm mục tiêu, quá trình học vẫn có thể thất bại vì hai lý do khác nhau. Thứ nhất, thuật toán tối ưu được sử dụng để huấn luyện có thể không tìm được giá trị của các tham số tương ứng với hàm mong muốn. Thứ hai, thuật toán huấn luyện có thể chọn sai hàm do hiện tượng quá khớp. Nhớ lại từ Mục 5.2.1 rằng định lý "không có bữa ăn miễn phí" chỉ ra rằng không tồn tại thuật toán học máy nào vượt trội trong mọi trường hợp. Các mạng lan truyền thẳng cung cấp một hệ thống phổ quát để biểu diễn các hàm, theo nghĩa là, với một hàm bất kỳ, sẽ tồn tại một mạng lan truyền thẳng có khả năng xấp xỉ hàm đó. Tuy nhiên, không có quy trình nào phổ quát để phân tích tập huấn luyện gồm các ví dụ cụ thể và chọn ra một hàm có khả năng tổng quát hóa cho các điểm không nằm trong tập huấn luyện.

Định lý xấp xỉ phổ quát khẳng định rằng tồn tại một mạng đủ lớn để đạt được bất kỳ độ chính xác nào mà chúng ta mong muốn, nhưng định lý không nói rõ mạng đó phải lớn đến mức nào. *Universal Approximation Bounds for Superpositions of a Sigmoidal Function* (Barron, 1993, [19]) đưa ra một số giới hạn về kích thước của một mạng một lớp cần thiết để xấp xỉ một lớp hàm rộng. Tuy nhiên, trong trường hợp xấu nhất, có thể cần đến số lượng đơn vị ẩn theo cấp số mũ (có khả năng với mỗi cấu hình đầu vào cần phân biệt tương ứng với một đơn vị ẩn). Điều này dễ thấy nhất trong trường hợp nhị phân: số lượng hàm nhị phân khả dĩ trên các véctơ $\boldsymbol{v} \in \{0,1\}^n$ là 2^{2^n} và việc chọn một hàm như vậy cần 2^n bit, điều này nói chung đòi hỏi $O(2^n)$ bậc tự do.

Tóm lại, một mạng truyền thẳng với một lớp ẩn là đủ để biểu diễn bất kỳ hàm nào, nhưng lớp này có thể quá lớn đến mức không khả thi và có thể thất bại trong việc học và tổng quát hóa đúng. Trong nhiều trường hợp, việc sử dụng các mô hình sâu hơn có thể giảm số lượng đơn vị cần thiết để biểu diễn hàm mong muốn và giảm sai số tổng quát hóa.

Tồn tại các họ hàm có thể được xấp xỉ hiệu quả bởi một kiến trúc với độ sâu lớn hơn một giá trị d nào đó, nhưng lại đòi hỏi một mô hình lớn hơn nhiều nếu độ sâu bị giới hạn nhỏ hơn hoặc bằng d. Trong nhiều trường hợp, số lượng đơn vị ẩn mà mô hình nông cần có là theo cấp số mũ theo n. Các kết quả này ban đầu được chứng minh cho các mô hình không giống các mạng nơron liên tục, khả vi được sử dụng trong học máy, nhưng sau đó đã được mở rộng cho các mô hình này. Những kết quả đầu tiên liên quan đến các mạch logic (*Almost Optimal Lower Bounds for Small Depth Circuits*, Håstad, 1986, [20]). Sau đó, các nghiên cứu mở rộng kết

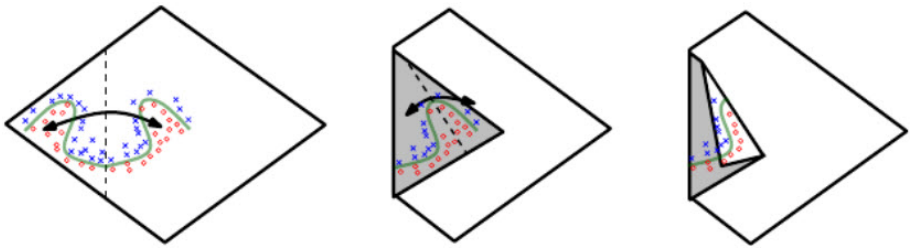
quả này cho các đơn vị ngưỡng tuyến tính với trọng số không âm (*On the Power of Small-Depth Threshold Circuits*, Håstad và Goldmann, 1991, [21]; *Threshold Circuits of Bounded Depth*, Hajnal và cộng sự, 1993, [22]), và tiếp theo là cho các mạng với các hàm kích hoạt có giá trị liên tục (*Bounds for the Computational Power and Learning Complexity of Analog Neural Nets*, Maass, 1993, [23]; *A Comparison of the Computational Power of Sigmoid and Boolean Threshold Circuits*, Maass và cộng sự, 1994, [24]). Nhiều mạng nơron hiện đại sử dụng các đơn vị tuyến tính chỉnh lưu. *Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function* (Leshno và cộng sự, 1993, [18]) đã chứng minh rằng các mạng nông với một họ rộng các hàm kích hoạt phi đa thức, bao gồm ReLU, có tính chất xấp xỉ phổ quát, nhưng các kết quả này không giải quyết câu hỏi về độ sâu hay hiệu quả — chúng chỉ khẳng định rằng một mạng ReLU đủ rộng có thể biểu diễn bất kỳ hàm nào. *On the Number of Linear Regions of Deep Neural Networks* (Montufar và cộng sự, 2014, [25]) cho thấy rằng các hàm có thể biểu diễn bằng một mạng ReLU sâu có thể yêu cầu số lượng đơn vị ẩn theo cấp số mũ nếu sử dụng mạng nông (chỉ một lớp ẩn). Cụ thể hơn, họ chứng minh rằng các mạng dạng từng đoạn tuyến tính (có thể thu được từ các hàm phi tuyến ReLU hoặc maxout) có thể biểu diễn các hàm với số lượng miền là theo cấp số mũ theo độ sâu của mạng. Hình 6.5 minh họa cách một mạng với hàm chỉnh lưu giá trị tuyệt đối tạo ra các hình ảnh đối xứng của hàm được tính toán trên một số đơn vị ẩn, đối xứng qua đầu vào của đơn vị ẩn đó. Mỗi đơn vị ẩn xác định nơi để gấp không gian đầu vào nhằm tạo ra các phản hồi đối xứng (ở cả hai phía của hàm phi tuyến giá trị tuyệt đối). Bằng cách kết hợp các thao tác gấp này, ta thu được số lượng miền tuyến tính từng đoạn rất lớn theo cấp số mũ, có thể mô tả mọi loại mẫu (ví dụ: mẫu lặp lại) khác nhau.

Cụ thể hơn, định lý chính trong *On the Number of Linear Regions of Deep Neural Networks* (Montufar và cộng sự, 2014, [25]) phát biểu rằng số lượng miền tuyến tính được tạo ra bởi một mạng ReLU sâu với d đầu vào, độ sâu ℓ, và n đơn vị trong mỗi lớp ẩn được xác định bởi:

$$O\left(\binom{n}{d}^{d(\ell-1)} n^d\right), \tag{6.43}$$

tức là, tăng theo cấp số mũ theo độ sâu ℓ. Trong trường hợp mạng maxout với k bộ lọc trên mỗi đơn vị, số lượng miền tuyến tính được xác định bởi:

$$O\left(k^{(\ell-1)+d}\right). \tag{6.44}$$

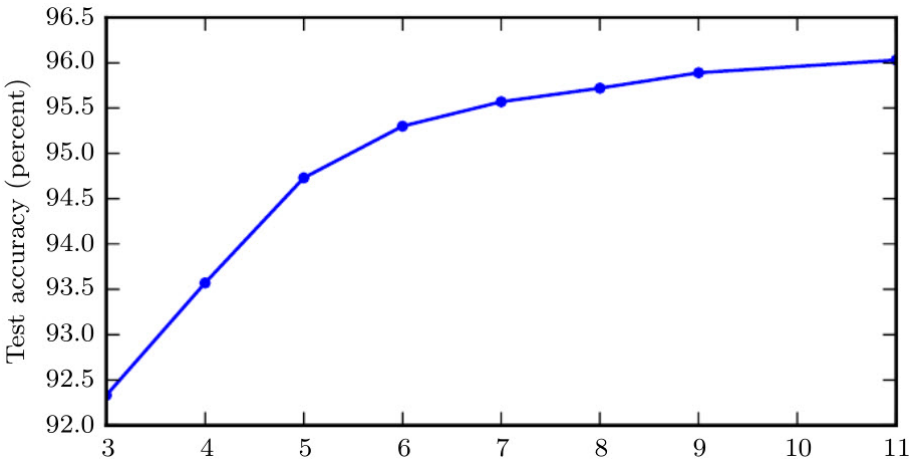


Hình 6.5: Một giải thích trực quan, mang tính hình học về lợi thế theo cấp số mũ của các mạng ReLU sâu đã được *On the Number of Linear Regions of Deep Neural Networks* (Montufar và cộng sự, 2014, [25]) trình bày chính thức. *Hình trái*: Một đơn vị chỉnh lưu giá trị tuyệt đối có đầu ra giống nhau cho mọi cặp điểm đối xứng qua trục đối xứng trong không gian đầu vào của nó. Trục đối xứng này được xác định bởi siêu phẳng được định nghĩa bởi các trọng số và độ chệch của đơn vị. Một hàm được tính toán trên đơn vị này (mặt quyết định màu xanh lá) sẽ là hình ảnh đối xứng của một mẫu đơn giản hơn qua trục đối xứng đó. *Hình giữa*: Hàm này có thể được hình dung như kết quả của việc "gấp" không gian quanh trục đối xứng. *Hình phải*: Một mẫu lặp lại khác có thể được gấp thêm lên mẫu đầu tiên (bởi một đơn vị hạ nguồn khác) để tạo ra một đối xứng mới (lúc này được lặp lại bốn lần, với hai lớp ẩn).

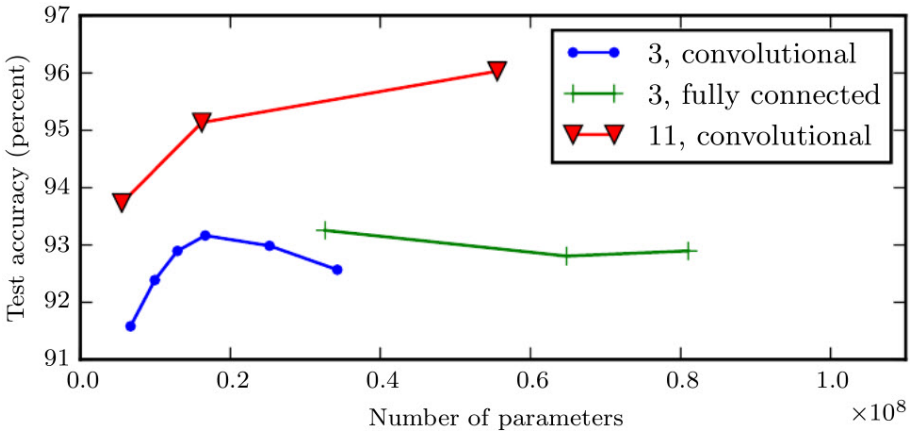
Tuy nhiên, không có gì đảm bảo rằng các loại hàm mà chúng ta muốn học trong ứng dụng học máy (đặc biệt là trong AI) sẽ chia sẻ tính chất như vậy.

Chúng ta cũng có thể muốn chọn mô hình sâu vì các lý do thống kê. Khi chọn một thuật toán học máy cụ thể, chúng ta ngầm định một số niềm tin tiên nghiệm về loại hàm mà thuật toán đó nên học. Việc chọn một mô hình sâu thể hiện một niềm tin rất tổng quát rằng hàm mục tiêu mà chúng ta muốn học bao gồm sự kết hợp của một số hàm đơn giản hơn. Từ góc nhìn học biểu diễn, điều này có thể được diễn giải rằng vấn đề học là phát hiện một tập hợp các nhân tố tiềm ẩn của sự biến thiên, và các nhân tố này có thể được mô tả thông qua các nhân tố tiềm ẩn đơn giản hơn. Một cách diễn giải khác, việc sử dụng kiến trúc sâu thể hiện niềm tin rằng hàm mà chúng ta muốn học là một chương trình máy tính bao gồm nhiều bước, trong đó mỗi bước sử dụng đầu ra của bước trước đó. Các đầu ra trung gian này không nhất thiết phải là các nhân tố của sự biến thiên, mà có thể tương tự như các bộ đếm hoặc con trỏ mà mạng sử dụng để tổ chức quá trình xử lý nội bộ của mình. Trên thực tế, độ sâu lớn hơn dường như cải thiện khả năng tổng quát hóa trong nhiều loại tác vụ khác nhau (*Greedy Layer-Wise Training of Deep Networks*, Bengio và cộng sự, 2006, [26]; *The Difficulty of Training Deep Architectures and*

the Effect of Unsupervised Pre-Training, Erhan và cộng sự, 2009, [27]; *Learning Deep Architectures for AI*, Bengio, 2009, [28]; *Unsupervised and Transfer Learning Challenge: a Deep Learning Approach*, Mesnil và cộng sự, 2012, [29]; *Multi-column deep neural network for traffic sign classification*, Ciresan và cộng sự, 2012, [30]; *ImageNet Classification with Deep Convolutional Neural Networks*, Krizhevsky và cộng sự, 2012, [31]; *Pedestrian Detection with Unsupervised Multi-stage Feature Learning*, Sermanet và cộng sự, 2013, [32]; *Learning Hierarchical Features for Scene Labeling*, Farabet và cộng sự, 2013, [33]; *Indoor Semantic Segmentation using depth information*, Couprie và cộng sự, 2013, [34]; *Combining Modality Specific Deep Neural Networks for Emotion Recognition in Video*, Kahou và cộng sự, 2013, [35]; *Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks*, Goodfellow và cộng sự, 2014, [36]; *Going Deeper with Convolutions*, Szegedy và cộng sự, 2014, [37]). Hình 6.6 và Hình 6.7 minh họa một số kết quả thực nghiệm này. Những kết quả này gợi ý rằng việc sử dụng các kiến trúc sâu thực sự thể hiện một tiên nghiệm hữu ích đối với không gian các hàm mà mô hình học được.



Hình 6.6: Các kết quả thực nghiệm cho thấy rằng các mạng sâu hơn có khả năng tổng quát hóa tốt hơn khi được sử dụng để chuyển đổi các số có nhiều chữ số từ ảnh chụp các địa chỉ. Dữ liệu được lấy từ nghiên cứu *Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks* (Goodfellow và cộng sự, 2014, [36]). Độ chính xác trên tập kiểm tra tăng đều đặn khi độ sâu của mạng tăng lên. Xem Hình 6.7 để biết thí nghiệm kiểm soát, trong đó cho thấy rằng việc tăng kích thước mô hình theo cách khác không mang lại hiệu quả tương tự.



Hình 6.7: Các mô hình sâu hơn có xu hướng hoạt động tốt hơn, và điều này không chỉ đơn thuần vì kích thước của mô hình lớn hơn. Thí nghiệm từ *Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks* (Goodfellow và cộng sự, 2014, [36]) cho thấy rằng việc tăng số lượng tham số trong các lớp mạng tích chập mà không tăng độ sâu của mạng không mang lại hiệu quả cải thiện đáng kể đối với hiệu suất trên tập kiểm tra. Chú giải trong hình minh họa độ sâu của mạng được sử dụng để tạo từng đường cong và liệu đường cong đó biểu diễn sự thay đổi kích thước của các lớp tích chập hay các lớp kết nối đầy đủ. Ta quan sát thấy rằng trong ngữ cảnh này, các mô hình nông bị quá khớp khi số tham số xấp xỉ 20 triệu, trong khi các mô hình sâu có thể tận dụng được hơn 60 triệu tham số. Điều này gợi ý rằng việc sử dụng mô hình sâu thể hiện một ưu tiên hữu ích đối với không gian các hàm mà mô hình có thể học. Cụ thể, nó thể hiện một niềm tin rằng hàm mục tiêu nên bao gồm sự kết hợp của nhiều hàm đơn giản hơn. Điều này có thể dẫn đến việc học một biểu diễn bao gồm các biểu diễn đơn giản hơn (ví dụ: các góc được định nghĩa thông qua các cạnh), hoặc học một chương trình gồm các bước phụ thuộc tuần tự (ví dụ: đầu tiên xác định một tập hợp các đối tượng, sau đó phân đoạn chúng, rồi nhận diện chúng).

6.4.2 Các cân nhắc khác về kiến trúc

Cho đến nay, chúng ta đã mô tả mạng nơron như những chuỗi lớp đơn giản, với các yếu tố chính cần xem xét là độ sâu của mạng và độ rộng của mỗi lớp. Tuy nhiên, trong thực tế, các mạng nơron thể hiện sự đa dạng đáng kể hơn.

Nhiều kiến trúc mạng nơron đã được phát triển để phục vụ các nhiệm vụ cụ thể. Các kiến trúc chuyên biệt cho thị giác máy tính, được gọi là mạng tích chập, sẽ được mô tả trong Chương 9. Các mạng lan truyền thẳng cũng có thể được mở rộng

thành các mạng nơron hồi quy để xử lý chuỗi dữ liệu, được mô tả trong Chương 10, và chúng có những cân nhắc riêng về mặt kiến trúc.

Nhìn chung, các lớp không nhất thiết phải được kết nối thành một chuỗi, mặc dù đây là cách thực hành phổ biến nhất. Nhiều kiến trúc xây dựng một chuỗi chính nhưng sau đó thêm các tính năng kiến trúc bổ sung, chẳng hạn như các kết nối bỏ qua từ lớp i đến lớp $i + 2$ hoặc cao hơn. Các kết nối bỏ qua này giúp gradient dễ dàng lan truyền từ các lớp đầu ra đến các lớp gần đầu vào hơn.

Một yếu tố quan trọng khác trong thiết kế kiến trúc là cách kết nối chính xác một cặp lớp với nhau. Trong lớp mạng nơron mặc định được mô tả bởi một phép biến đổi tuyến tính thông qua ma trận $\boldsymbol{W}$, mỗi đơn vị đầu vào được kết nối với mọi đơn vị đầu ra. Tuy nhiên, nhiều mạng chuyên biệt trong các chương sau có ít kết nối hơn, để mỗi đơn vị trong lớp đầu vào chỉ được kết nối với một tập con nhỏ các đơn vị trong lớp đầu ra. Các chiến lược giảm số lượng kết nối này giúp giảm số lượng tham số và lượng tính toán cần thiết để đánh giá mạng, nhưng thường phụ thuộc chặt chẽ vào bài toán cụ thể. Ví dụ, mạng tích chập, được mô tả trong Chương 9, sử dụng các mẫu kết nối thưa đặc biệt rất hiệu quả cho các bài toán thị giác máy tính. Trong chương này, rất khó để đưa ra lời khuyên cụ thể hơn về kiến trúc của một mạng nơron tổng quát. Các chương sau sẽ phát triển các chiến lược kiến trúc cụ thể đã được chứng minh là hiệu quả với các lĩnh vực ứng dụng khác nhau.

6.5 Lan truyền ngược và các thuật toán vi phân khác

Khi chúng ta sử dụng một mạng nơron lan truyền thẳng để nhận một đầu vào $\boldsymbol{x}$ và tạo ra một đầu ra $\widehat{\boldsymbol{y}}$, thông tin sẽ lan truyền theo hướng từ đầu vào đến đầu ra. Đầu vào $\boldsymbol{x}$ cung cấp thông tin ban đầu, sau đó lan truyền qua các đơn vị ẩn ở mỗi lớp và cuối cùng tạo ra $\widehat{\boldsymbol{y}}$. Quá trình này được gọi là **lan truyền tiến**. Trong quá trình huấn luyện, lan truyền tiến có thể tiếp tục cho đến khi tạo ra một giá trị vô hướng của hàm mất mát $J\left(\boldsymbol{\theta}\right)$. Thuật toán **lan truyền ngược** (*Learning Representations by Back-propagating Errors*, Rumelhart và cộng sự, 1986, [38]), cho phép thông tin từ hàm mất mát lan ngược lại qua mạng để tính gradient.

Việc biểu diễn gradient bằng biểu thức giải tích là một nhiệm vụ đơn giản, nhưng việc đánh giá biểu thức này bằng tính toán số có thể tốn nhiều chi phí tính toán. Thuật toán lan truyền ngược thực hiện việc này bằng một quy trình đơn giản và ít tốn kém.

Thuật ngữ “lan truyền ngược” thường bị hiểu nhầm là toàn bộ thuật toán học

dành cho mạng nơron nhiều lớp. Thực tế, lan truyền ngược chỉ là phương pháp tính gradient, trong khi một thuật toán khác, chẳng hạn như hướng giảm ngẫu nhiên, được sử dụng để thực hiện quá trình học dựa trên gradient đó. Hơn nữa, lan truyền ngược thường bị hiểu nhầm là chỉ áp dụng cho mạng nơron nhiều lớp, nhưng về nguyên tắc, nó có thể tính đạo hàm của bất kỳ hàm nào (với một số hàm, kết quả đúng là báo cáo rằng đạo hàm không xác định). Cụ thể, chúng ta sẽ mô tả cách tính gradient $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{y})$ cho một hàm bất kỳ f, trong đó $\boldsymbol{x}$ là tập hợp các biến cần tính đạo hàm, và $\boldsymbol{y}$ là tập hợp các biến đầu vào của hàm nhưng không cần tính đạo hàm. Trong các thuật toán học, gradient mà chúng ta thường cần tính nhất là gradient của hàm mất mát theo các tham số, $\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}\right)$. Nhiều bài toán học máy khác yêu cầu tính các đạo hàm khác, hoặc như một phần của quá trình học, hoặc để phân tích mô hình đã học. Thuật toán lan truyền ngược cũng có thể áp dụng cho các tác vụ này, và không bị giới hạn trong việc tính gradient của hàm mất mát theo tham số. Ý tưởng tính đạo hàm bằng cách lan truyền thông tin qua mạng là rất tổng quát và có thể được sử dụng để tính các giá trị như ma trận Jacobi của một hàm f với nhiều đầu ra. Tuy nhiên, ở đây chúng ta sẽ giới hạn mô tả trong trường hợp phổ biến nhất, khi f có một đầu ra duy nhất.

6.5.1 Đồ thị tính toán

Đến nay, chúng ta đã thảo luận về mạng nơron bằng ngôn ngữ đồ thị tương đối không chính thức. Để mô tả thuật toán lan truyền ngược một cách chính xác hơn, việc sử dụng một ngôn ngữ **đồ thị tính toán** chặt chẽ hơn sẽ rất hữu ích.

Có nhiều cách để chính thức hóa các phép tính toán dưới dạng đồ thị.

Ở đây, chúng ta sử dụng mỗi nút trong đồ thị để biểu diễn một biến. Biến này có thể là một vô hướng, một véctơ, ma trận, tenxơ, hoặc thậm chí là một biến thuộc loại khác.

Để chính thức hóa đồ thị của mình, chúng ta cũng cần giới thiệu khái niệm **phép toán**. Một phép toán là một hàm đơn giản nhận một hoặc nhiều biến làm đầu vào. Ngôn ngữ đồ thị của chúng ta đi kèm với một tập hợp các phép toán được phép sử dụng. Các hàm phức tạp hơn so với các phép toán trong tập hợp này có thể được biểu diễn bằng cách kết hợp nhiều phép toán với nhau.

Không làm mất tính tổng quát, chúng ta định nghĩa một phép toán chỉ trả về một biến đầu ra duy nhất. Điều này không làm mất tính tổng quát bởi vì biến đầu ra có thể chứa nhiều phần tử, chẳng hạn như một véctơ. Các triển khai phần mềm của thuật toán lan truyền ngược thường hỗ trợ các phép toán với nhiều đầu ra,

nhưng trong phần mô tả này, chúng ta tránh trường hợp đó để loại bỏ nhiều chi tiết không cần thiết cho việc hiểu khái niệm.

Nếu một biến y được tính bằng cách áp dụng một phép toán lên biến x, thì chúng ta vẽ một cạnh có hướng từ x đến y. Đôi khi, chúng ta chú thích nút đầu ra bằng tên của phép toán đã áp dụng, và những lần khác, chúng ta bỏ qua nhãn này nếu phép toán rõ ràng từ ngữ cảnh.

Các ví dụ về đồ thị tính toán được minh họa trong Hình 6.8.

6.5.2 Quy tắc chuỗi trong giải tích

Quy tắc chuỗi trong giải tích (khác với quy tắc chuỗi trong xác suất), còn gọi là quy tắc đạo hàm của hàm hợp, được sử dụng để tính đạo hàm của các hàm được tạo thành bằng cách kết hợp các hàm khác mà có đạo hàm đã biết. Thuật toán lan truyền ngược là một thuật toán thực thi quy tắc chuỗi với thứ tự các phép toán được tối ưu hóa để đạt hiệu quả cao.

Giả sử x là một số thực, và f và g đều là các hàm ánh xạ từ một số thực thành một số thực. Nếu $y = g(x)$ và $z = f(g(x)) = f(y)$, quy tắc chuỗi phát biểu rằng:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}. \tag{6.45}$$

Ta có thể tổng quát vượt ra ngoài trường hợp vô hướng. Giả sử $\boldsymbol{x} \in \mathbb{R}^m$, $\boldsymbol{y} \in \mathbb{R}^n$, g ánh xạ từ $\mathbb{R}^m$ đến $\mathbb{R}^n$, và f ánh xạ từ $\mathbb{R}^n$ đến $\mathbb{R}$. Nếu $\boldsymbol{y} = g(\boldsymbol{x})$ và $z = f(\boldsymbol{y})$, thì:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}. \tag{6.46}$$

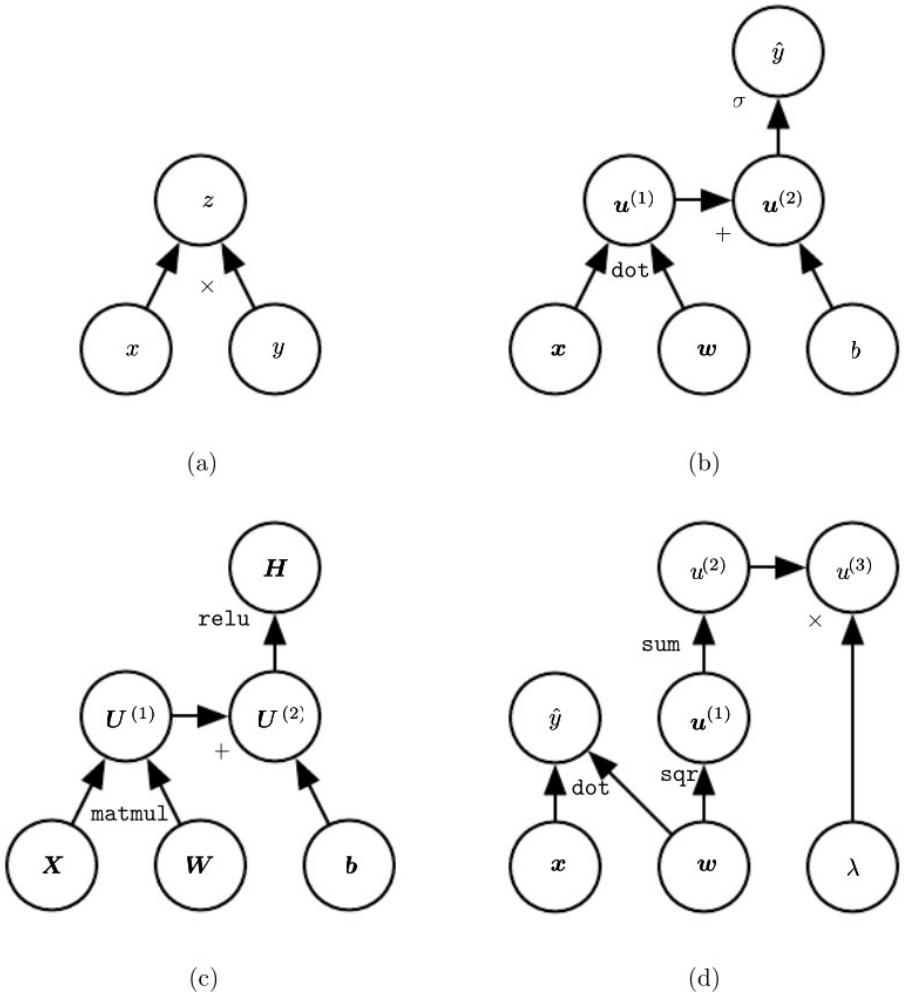
Ta có thể viết tương đương dưới dạng véctơ:

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \Delta_{\boldsymbol{y}} z, \tag{6.47}$$

trong đó $\dfrac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ là ma trận Jacobi cỡ $n \times m$ của hàm g.

Từ công thức, ta thấy gradient của một biến $\boldsymbol{x}$ có thể được tính bằng cách nhân ma trận Jacobi $\dfrac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ với gradient $\nabla_{\boldsymbol{y}} z$. Thuật toán lan truyền ngược được xây dựng bằng cách thực hiện phép nhân Jacobi – gradient này cho mỗi phép toán trong đồ thị tính toán.

Thông thường, ta không chỉ áp dụng thuật toán lan truyền ngược cho các véctơ mà còn cho các tenxơ có số chiều bất kỳ. Về mặt ý tưởng, quá trình này giống



Hình 6.8: Ví dụ về Đồ thị Tính toán. (a) Đồ thị sử dụng phép toán $\times$ để tính $z = xy$. (b) Đồ thị cho dự đoán hồi quy logistic $\hat{y} = \sigma\left(\boldsymbol{x}^\top \boldsymbol{w} + b\right)$. Một số biểu thức trung gian không có tên trong biểu thức đại số nhưng cần được đặt tên trong đồ thị. Chúng ta đơn giản đặt tên cho biến thứ i là $\boldsymbol{u}^{(i)}$. (c) Đồ thị tính toán cho biểu thức $\boldsymbol{H} = \max\{0, \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}\}$, trong đó $\boldsymbol{H}$ là một ma trận thiết kế chứa các giá trị kích hoạt ReLU, được tính từ một ma trận thiết kế đầu vào $\boldsymbol{X}$ chứa một nhóm nhỏ các đầu vào. (d) Các ví dụ (a)–(c) chỉ áp dụng tối đa một phép toán lên mỗi biến, nhưng có thể áp dụng nhiều phép toán lên một biến. Ở đây, đồ thị thể hiện việc áp dụng nhiều phép toán lên trọng số $\boldsymbol{w}$ của mô hình hồi quy tuyến tính. Các trọng số được sử dụng để tạo ra cả dự đoán $\hat{y}$ và mức phạt trọng số $\lambda \sum_i w_i^2$.

hệt với lan truyền ngược trên véctơ. Sự khác biệt duy nhất nằm ở cách sắp xếp các số trong dạng lưới để tạo thành một tenxơ. Có thể hình dung việc làm phẳng

mỗi tenxơ thành một vectơ trước khi thực hiện lan truyền ngược, tính gradient dưới dạng véctơ, rồi chuyển đổi gradient trở lại thành tenxơ. Với cách nhìn sắp xếp lại này, lan truyền ngược vẫn chỉ là phép nhân giữa Jacob và gradient.

Để biểu diễn gradient của một giá trị z theo tenxơ $\mathcal{X}$, ta viết $\nabla_{\mathcal{X}} z$, tương tự như khi $\mathcal{X}$ là véctơ. Chỉ số của $\mathcal{X}$ giờ đây có thể là một tập hợp tọa độ — ví dụ, tenxơ 3 chiều được đánh chỉ số bởi ba tọa độ. Ta có thể trừu tượng hóa các tọa độ này bằng cách sử dụng một biến duy nhất i để đại diện cho toàn bộ tổ hợp các chỉ số. Với mọi tổ hợp chỉ số i, $(\nabla_{\mathcal{X}} z)_i$ biểu diễn $\frac{\partial z}{\partial \mathcal{X}_i}$. Điều này hoàn toàn tương tự như cách mà với véctơ, $(\nabla_{\boldsymbol{x}} z)_i$ biểu diễn $\frac{\partial z}{\partial x_i}$. Sử dụng ký hiệu này, ta có thể viết quy tắc chuỗi khi áp dụng cho tenxơ. Nếu $\mathcal{Y} = g(\mathcal{X})$ và $z = f(\mathcal{Y})$, thì:

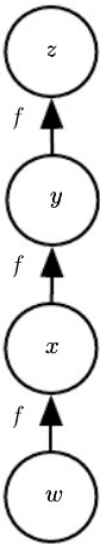
$$\Delta_{\mathcal{X}} z = \sum_j \left(\Delta_{\mathcal{X}} \mathcal{Y}_j\right) \frac{\partial z}{\partial \mathcal{Y}_j}. \tag{6.48}$$

6.5.3 Áp dụng quy tắc chuỗi đệ quy để xây dựng lan truyền ngược

Sử dụng quy tắc chuỗi, việc viết biểu thức đại số cho gradient của một đại lượng vô hướng đối với bất kỳ nút nào trong đồ thị tính toán sinh ra đại lượng đó là đơn giản. Tuy nhiên, việc đánh giá biểu thức này trên máy tính đòi hỏi xem xét thêm một số yếu tố.

Cụ thể, nhiều biểu thức con có thể được lặp lại nhiều lần trong biểu thức tổng thể của gradient. Bất kỳ quy trình nào tính toán gradient sẽ cần quyết định lưu trữ các biểu thức con để sử dụng lại hoặc tính toán lại chúng mỗi lần cần. Ví dụ về cách các biểu thức con lặp lại được minh họa trong Hình 6.9. Trong một số trường hợp, việc tính toán lại các biểu thức giống nhau sẽ chỉ gây lãng phí tài nguyên. Với các đồ thị phức tạp, có thể tồn tại số lượng hàm mũ các phép tính lãng phí, khiến việc triển khai quy tắc chuỗi một cách đơn giản trở nên không khả thi. Tuy nhiên, trong một số trường hợp, việc tính toán lại một biểu thức có thể là lựa chọn hợp lý để giảm mức tiêu thụ bộ nhớ, đổi lại bằng tăng thời gian chạy.

Chúng ta bắt đầu với một phiên bản của thuật toán lan truyền ngược mô tả trực tiếp cách tính gradient thực tế (Thuật toán 6.2 cùng với Thuật toán 6.1 cho phần tính toán tiến tương ứng), theo thứ tự mà các bước thực sự được thực hiện và dựa trên việc áp dụng đệ quy của quy tắc chuỗi. Ta có thể thực hiện trực tiếp các phép tính này hoặc coi phần mô tả thuật toán như là một đặc tả ký hiệu của đồ thị tính toán để thực hiện lan truyền ngược. Tuy nhiên, cách tiếp cận này không làm rõ việc



Hình 6.9: Một đồ thị tính toán dẫn đến các biểu thức con lặp lại khi tính gradient. Giả sử $w \in \mathbb{R}$ là đầu vào của đồ thị. Ta sử dụng cùng một hàm $f : \mathbb{R} \to \mathbb{R}$ làm phép toán được áp dụng tại mỗi bước của chuỗi: $x = f(w)$, $y = f(x)$, $z = f(y)$. Để tính $\frac{\partial z}{\partial w}$, ta áp dụng phương trình (6.45) và thu được:

$$\frac{\partial z}{\partial z} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} = f'(y)\ f'(x)\ f'(w) = f'(f(f(w)))\ f'(f(w))\ f'(w)$$

Đẳng thức thứ hai gợi ý một cách triển khai trong đó giá trị của $f(w)$ chỉ được tính một lần và lưu trữ trong biến x. Đây là cách tiếp cận mà thuật toán lan truyền ngược sử dụng. Một cách tiếp cận khác được gợi ý bởi đẳng thức thứ ba, trong đó biểu thức con $f(w)$ xuất hiện nhiều lần. Với cách tiếp cận này, $f(w)$ được tính lại mỗi khi cần đến. Khi bộ nhớ cần thiết để lưu trữ giá trị của các biểu thức này thấp, cách tiếp cận của đẳng thức thứ hai (thuật toán lan truyền ngược) rõ ràng là ưu tiên hơn nhờ giảm thời gian chạy. Tuy nhiên, đẳng thức thứ ba cũng là một triển khai hợp lệ của quy tắc chuỗi và trở nên hữu ích khi bộ nhớ bị hạn chế.

thao tác và xây dựng đồ thị ký hiệu phục vụ cho việc tính gradient. Cách tiếp cận này sẽ được trình bày chi tiết hơn trong Mục 6.5.6, với Thuật toán 6.5, nơi chúng ta cũng tổng quát hóa cho các nút chứa các tenxơ bất kỳ.

Trước tiên, hãy xem xét một đồ thị tính toán mô tả cách tính một số vô hướng $u^{(n)}$ (ví dụ, hàm mất mát trên một mẫu huấn luyện). Đây là đại lượng mà ta muốn tính gradient theo các nút đầu vào $u^{(1)}$ đến $u^{(n_i)}$. Nói cách khác, chúng ta muốn tính $\frac{\partial u^{(n)}}{\partial u^{(i)}}$ với mọi $i \in \{1, 2, \ldots, n_i\}$. Trong ứng dụng của lan truyền ngược để tính gradient phục vụ cho thuật toán hướng giảm trên các tham số, $u^{(n)}$ sẽ là hàm chi

Thuật toán 6.1: Một quy trình thực hiện các phép tính ánh xạ n_i đầu vào $u^{(1)}$ đến $u^{(n_i)}$ thành một đầu ra $u^{(n)}$. Điều này định nghĩa một đồ thị tính toán, trong đó mỗi nút tính toán giá trị số $u^{(i)}$ bằng cách áp dụng một hàm $f^{(i)}$ lên tập các đối số $A^{(i)}$, bao gồm các giá trị của các nút trước đó $u^{(j)}$, với $j < i$ và $j \in \text{Pa}\left(u^{(i)}\right)$ (Pa là tập các nút cha của $u^{(i)}$). Đầu vào của đồ thị tính toán là véctơ $\boldsymbol{x}$, được gán giá trị cho n_i nút đầu tiên, từ $u^{(1)}$ đến $u^{(n_i)}$. Đầu ra của đồ thị tính toán được lấy từ nút cuối cùng (nút đầu ra) $u^{(n)}$.

for $i \leftarrow 1, \ldots, n_i$ **do**
  $u^{(i)} \leftarrow x_i$
for $i = n_i + 1, \ldots, n$ **do**
  $A^{(i)} \leftarrow \left\{u^{(j)} \mid j \in \text{Pa}\left(u^{(i)}\right)\right\}$
  $u^{(i)} \leftarrow f^{(i)}\left(A^{(i)}\right)$
return $u^{(n)}$

phí liên quan đến một mẫu hoặc một nhóm mẫu nhỏ, trong khi $u^{(1)}$ đến $u^{(n_i)}$ tương ứng với các tham số của mô hình.

Ta giả sử các nút trong đồ thị đã được sắp xếp sao cho có thể tính đầu ra của chúng tuần tự, bắt đầu từ $u^{(n_i+1)}$ đến $u^{(n)}$. Như được định nghĩa trong Thuật toán 6.1, mỗi nút $u^{(i)}$ được liên kết với một phép toán $f^{(i)}$ và được tính bằng cách đánh giá hàm:

$$u^{(i)} = f\left(A^{(i)}\right) \tag{6.49}$$

trong đó $A^{(i)}$ là tập hợp tất cả các nút cha của $u^{(i)}$.

Thuật toán đó xác định phép tính lan truyền, mà chúng ta có thể biểu diễn trong một đồ thị $\mathcal{G}$. Để thực hiện lan truyền ngược, ta có thể xây dựng một đồ thị tính toán phụ thuộc vào G bằng cách thêm vào nó một tập hợp nút bổ sung. Các nút này tạo thành một đồ thị con $\mathcal{B}$, trong đó mỗi nút tương ứng với một nút trong $\mathcal{G}$. Phép tính trong $\mathcal{B}$ được thực hiện theo đúng thứ tự ngược lại so với phép tính trong $\mathcal{G}$, và mỗi nút trong $\mathcal{B}$ tính đạo hàm $\dfrac{\partial u^{(n)}}{\partial u^{(i)}}$ liên quan đến nút tương ứng $u^{(i)}$ trong đồ thị lan truyền tiến. Điều này được thực hiện bằng cách áp dụng quy tắc chuỗi đối với đầu ra vô hướng $u^{(n)}$:

$$\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{i:\, j \in \text{Pa}\left(u^{(i)}\right)} \frac{\partial u^{(n)}}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial u^{(j)}} \tag{6.50}$$

như được chỉ ra trong Thuật toán 6.2. Đồ thị con $\mathcal{B}$ có đúng một cạnh tương ứng với mỗi cạnh từ nút $u^{(j)}$ đến nút $u^{(i)}$ trong $\mathcal{G}$. Cạnh từ $u^{(j)}$ đến $u^{(i)}$ liên quan đến việc

Thuật toán 6.2: Phiên bản đơn giản hóa của thuật toán lan truyền ngược được sử dụng để tính các đạo hàm của $u^{(n)}$ theo các biến trong đồ thị. Ví dụ này nhằm giúp hiểu rõ hơn bằng cách minh họa một trường hợp đơn giản, trong đó tất cả các biến đều là số vô hướng, và chúng ta muốn tính đạo hàm theo $u^{(1)}, \dots, u^{(n_i)}$. Phiên bản đơn giản hóa này tính các đạo hàm của tất cả các nút trong đồ thị. Chi phí tính toán của thuật toán này tỷ lệ với số cạnh trong đồ thị, giả sử rằng đạo hàm riêng tương ứng với mỗi cạnh chỉ yêu cầu thời gian hằng số. Điều này có cùng bậc với số phép tính cần thiết cho lan truyền tiến. Mỗi $\frac{\partial u^{(i)}}{\partial u^{(j)}}$ là một hàm của các nút cha $u^{(j)}$ của $u^{(i)}$, do đó liên kết các nút của đồ thị tiến với các nút được thêm vào cho đồ thị lan truyền ngược.

1 Chạy lan truyền tiến (Thuật toán 6.1 trong ví dụ này) để tính các giá trị kích hoạt của mạng.

2 Khởi tạo grad_table, một cấu trúc dữ liệu để lưu trữ các đạo hàm đã tính được. Mục grad_table $\left[u^{(i)}\right]$ sẽ lưu giá trị tính xong của $\frac{\partial u^{(n)}}{\partial u^{(i)}}$.

3 grad_table $\left[u^{(n)}\right] \leftarrow 1$

4 **for** $j \leftarrow n-1, \dots, 1$ **do**

5 Tính $\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{i:\, j \in \mathrm{Pa}\left(u^{(i)}\right)} \frac{\partial u^{(n)}}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial u^{(j)}}$ bằng cách sử dụng các giá trị đã được lưu: grad_table $\left[u^{(j)}\right] \leftarrow \sum_{i:\, j \in \mathrm{Pa}\left(u^{(i)}\right)}$ grad_table $\left[u^{(i)}\right] \frac{\partial u^{(i)}}{\partial u^{(j)}}$

6 **return** $\left\{\text{grad\_table}\left[u^{(i)}\right] \mid i = 1, \dots, n_i\right\}$

tính $\frac{\partial u^{(i)}}{\partial u^{(j)}}$. Ngoài ra, tại mỗi nút, một tích vô hướng được thực hiện giữa gradient đã được tính liên quan đến các nút $u^{(i)}$ là con của $u^{(j)}$ và véctơ chứa các đạo hàm riêng $\frac{\partial u^{(i)}}{\partial u^{(j)}}$ cho các nút con $u^{(i)}$ đó. Tóm lại, lượng tính toán cần thiết để thực hiện lan truyền ngược tỷ lệ tuyến tính với số cạnh trong $\mathcal{G}$. Mỗi phép tính trên một cạnh bao gồm việc tính một đạo hàm riêng (của một nút theo một trong các nút cha của nó) cùng với một phép nhân và một phép cộng. Ở phần dưới, chúng ta sẽ mở rộng phân tích này cho các nút giá trị tenxơ, vốn là cách gom nhiều giá trị vô hướng trong cùng một nút để cho phép thực hiện các tính toán hiệu quả hơn.

Thuật toán lan truyền ngược được thiết kế để giảm số lượng các biểu thức con chung mà không chú ý đến việc sử dụng bộ nhớ. Cụ thể, nó thực hiện theo cấp

của một phép nhân Jacobi cho mỗi nút trong đồ thị. Điều này có thể thấy rõ từ thực tế rằng lan truyền ngược (Thuật toán 6.2) chỉ duyệt qua mỗi cạnh từ nút $u^{(j)}$ đến $u^{(i)}$ trong đồ thị đúng một lần để tính đạo hàm riêng tương ứng $\frac{\partial u^{(i)}}{\partial u^{(j)}}$. Do đó, thuật toán lan truyền ngược tránh được sự bùng nổ theo cấp hàm mũ của các biểu thức con lặp lại. Tuy nhiên, một số thuật toán khác có thể giảm được nhiều biểu thức con hơn bằng cách thực hiện các phép đơn giản hóa trên đồ thị tính toán, hoặc có thể tiết kiệm bộ nhớ bằng cách tính toán lại thay vì lưu trữ một số biểu thức con. Chúng ta sẽ quay lại thảo luận những ý tưởng này sau khi mô tả chi tiết thuật toán lan truyền ngược.

6.5.4 Phép tính lan truyền ngược trong mạng đa lớp kết nối đầy đủ

Để làm rõ định nghĩa trên về phép tính lan truyền ngược, hãy xem xét đồ thị cụ thể liên quan đến một mạng nơron đa lớp kết nối đầy đủ.

Thuật toán 6.3 đầu tiên trình bày quá trình lan truyền tiến, ánh xạ các tham số đến hàm mất mát có giám sát $L\left(\widehat{\boldsymbol{y}}, \boldsymbol{y}\right)$ liên quan đến một cặp huấn luyện (đầu vào, mục tiêu) cụ thể $(\boldsymbol{x}, \boldsymbol{y})$, trong đó $\widehat{\boldsymbol{y}}$ là đầu ra của mạng nơron khi $\boldsymbol{x}$ được cung cấp làm đầu vào.

Sau đó, Thuật toán 6.4 mô tả các phép tính tương ứng cần thực hiện để áp dụng thuật toán lan truyền ngược trên đồ thị này.

Thuật toán 6.3 và 6.4 được trình bày nhằm minh họa một cách đơn giản và dễ hiểu. Tuy nhiên, chúng được thiết kế chuyên biệt cho một bài toán cụ thể.

Các triển khai phần mềm hiện đại dựa trên dạng tổng quát của lan truyền ngược được mô tả trong Mục 6.5.6 bên dưới, cho phép xử lý bất kỳ đồ thị tính toán nào bằng cách thao tác trực tiếp trên cấu trúc dữ liệu biểu diễn các phép tính với ký hiệu.

6.5.5 Đạo hàm từ ký hiệu sang ký hiệu

Các biểu thức đại số và đồ thị tính toán đều hoạt động trên các **ký hiệu**, hay các biến không có giá trị cụ thể. Những biểu diễn đại số và dựa trên đồ thị này được gọi là biểu diễn **ký hiệu**. Khi chúng ta thực sự sử dụng hoặc huấn luyện một mạng nơron, ta phải gán các giá trị cụ thể cho các ký hiệu này. Chẳng hạn, ta thay thế một đầu vào ký hiệu $\boldsymbol{x}$ của mạng bằng một giá trị số cụ thể, chẳng hạn như $[1.2, 3.765, -1.8]^\top$.

Thuật toán 6.3: Lan truyền tiến qua một mạng nơron sâu thông thường và phép tính hàm chi phí. Hàm mất mát $L\left(\widehat{\boldsymbol{y}}, \boldsymbol{y}\right)$ phụ thuộc vào đầu ra $\widehat{\boldsymbol{y}}$ của mạng và mục tiêu $\boldsymbol{y}$ (xem Mục 6.2.1.1 để biết các ví dụ về hàm mất mát). Để thu được tổng chi phí J, hàm mất mát có thể được cộng thêm một bộ điều chuẩn $\Omega\left(\boldsymbol{\theta}\right)$, trong đó $\boldsymbol{\theta}$ bao gồm tất cả các tham số (trọng số và độ dời). Thuật toán 6.4 chỉ ra cách tính gradient của J theo các tham số $\boldsymbol{W}$ và $\boldsymbol{b}$. Để đơn giản hóa, minh họa này chỉ sử dụng một ví dụ đầu vào $\boldsymbol{x}$. Trong các ứng dụng thực tế, nên sử dụng một nhóm nhỏ các ví dụ. Xem Mục 6.5.7 để biết một minh họa thực tế hơn.

Đầu vào:

- Độ sâu ℓ của mạng
- Các ma trận trọng số $\boldsymbol{W}^{(i)}$, $i \in \{1, \ldots, \ell\}$ của mô hình
- Các tham số độ dời $\boldsymbol{b}^{(i)}$, $i \in \{1, \ldots, \ell\}$ của mô hình
- Đầu vào $\boldsymbol{x}$ để xử lý
- Đầu ra mục tiêu $\boldsymbol{y}$

$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \ldots, \ell$ **do**
  $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)}\boldsymbol{h}^{(k-1)}$
  $\boldsymbol{h}^{(k)} = f\left(\boldsymbol{a}^{(k)}\right)$
$\widehat{\boldsymbol{y}} = \boldsymbol{h}^{(\ell)}$
$J = L\left(\widehat{\boldsymbol{y}}, \boldsymbol{y}\right) + \lambda\Omega\left(\boldsymbol{\theta}\right)$

Một số phương pháp lan truyền ngược sử dụng một đồ thị tính toán cùng với một tập giá trị số cho các đầu vào của đồ thị, sau đó trả về một tập giá trị số mô tả gradient tại các giá trị đầu vào đó. Chúng ta gọi phương pháp này là phương pháp đạo hàm "từ ký hiệu sang số". Đây là phương pháp được sử dụng bởi các thư viện như Torch (*Torch7: A Matlab-like Environment for Machine Learning*, Collobert và cộng sự, 2011, [39]) và Caffe (*Caffe: Convolutional Architecture for Fast Feature Embedding*, Jia, 2014, [40]).

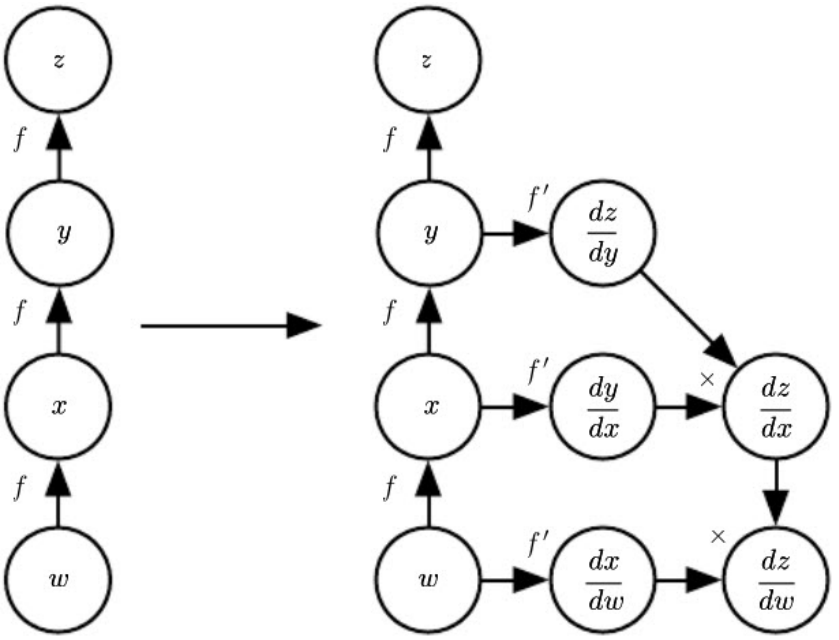
Một phương pháp khác là sử dụng một đồ thị tính toán và thêm các nút mới vào đồ thị để cung cấp mô tả ký hiệu về các đạo hàm mong muốn. Đây là phương pháp được sử dụng bởi Theano (*Theano: A CPU and GPU Math Compiler in Python*,

Thuật toán 6.4: Tính toán **lan truyền ngược** cho mạng nơron sâu trong Thuật toán 6.3, sử dụng đầu vào $\boldsymbol{x}$ và mục tiêu $\boldsymbol{y}$. Phép tính cho ra các gradient trên các kích hoạt $\boldsymbol{a}^{(k)}$ cho từng lớp k, bắt đầu từ lớp đầu ra và quay ngược về lớp ẩn đầu tiên. Từ các gradient này, có thể hiểu chúng như chỉ dẫn về cách đầu ra của từng lớp nên thay đổi để giảm sai số, từ đó thu được gradient theo các tham số của mỗi lớp. Các gradient theo trọng số và độ dời có thể được sử dụng ngay lập tức trong một bước cập nhật hướng ngẫu nhiên (thực hiện cập nhật ngay sau khi tính toán xong gradient), hoặc sử dụng cùng với các phương pháp tối ưu khác dựa trên gradient.

1 Sau khi thực hiện truyền tiến, tính gradient trên lớp đầu ra:
$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L\left(\hat{\boldsymbol{y}}, \boldsymbol{y}\right)$
2 **for** $k = \ell, \ell - 1, \ldots, 1$ **do**
3 Chuyển gradient trên đầu ra của lớp thành gradient trên giá trị trước kích hoạt phi tuyến (nhân từng phần tử với đạo hàm của hàm kích hoạt f nếu f áp dụng từng phần tử):
$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'\left(\boldsymbol{a}^{(k)}\right)$
4 Tính các gradient trên các trọng số và độ dời từ gradient kích hoạt và đầu vào của lớp (bao gồm hạng tử điều chuẩn nếu cần):
$\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega\left(\boldsymbol{\theta}\right)$
$\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega\left(\boldsymbol{\theta}\right)$
5 Lan truyền các gradient ngược về các kích hoạt của lớp ẩn liền kề:
$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top} \boldsymbol{g}$

Bergstra và cộng sự, 2010, [41]; *Theano: new features and speed improvements*, Bastien và cộng sự, 2012, [42]) và TensorFlow (*TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*, Abadi và cộng sự, 2016, [43]). Một ví dụ về cách hoạt động của phương pháp này được minh họa trong Hình 6.10. Lợi ích chính của phương pháp này là các đạo hàm được mô tả bằng cùng ngôn ngữ với biểu thức ban đầu. Vì các đạo hàm chỉ là một đồ thị tính toán khác, nên có thể thực hiện lan truyền ngược một lần nữa để tính các đạo hàm bậc cao hơn. Việc tính toán các đạo hàm bậc cao được mô tả trong Mục 6.5.10.

Chúng ta sẽ sử dụng phương pháp thứ hai và mô tả thuật toán lan truyền ngược theo cách xây dựng một đồ thị tính toán cho các đạo hàm. Bất kỳ tập con nào của đồ thị sau đó có thể được đánh giá bằng các giá trị số cụ thể tại một thời điểm sau



Hình 6.10: Một ví dụ về phương pháp từ ký hiệu sang ký hiệu trong việc tính toán đạo hàm. Trong phương pháp này, thuật toán lan truyền ngược không cần truy cập bất kỳ giá trị số cụ thể nào. Thay vào đó, nó thêm các nút vào đồ thị tính toán để mô tả cách tính các đạo hàm này. Một công cụ đánh giá đồ thị tổng quát sau đó có thể tính các đạo hàm cho bất kỳ giá trị số cụ thể nào. *Hình trái*: Trong ví dụ này, chúng ta bắt đầu với một đồ thị biểu diễn $z = f(f(f(w)))$. *Hình phải*: Ta chạy thuật toán lan truyền ngược, yêu cầu nó xây dựng đồ thị cho biểu thức tương ứng với $\frac{dz}{dw}$. Trong ví dụ này, chúng ta không giải thích cách thuật toán lan truyền ngược hoạt động. Mục đích chỉ là minh họa kết quả mong muốn: một đồ thị tính toán với mô tả ký hiệu của đạo hàm.

đó. Điều này cho phép chúng ta tránh phải chỉ định chính xác thời điểm cần tính toán từng phép toán. Thay vào đó, một công cụ đánh giá đồ thị tổng quát có thể đánh giá từng nút ngay khi các giá trị của các nút cha của nó sẵn sàng.

Mô tả về phương pháp dựa trên ký hiệu sang ký hiệu bao quát cả phương pháp từ ký hiệu sang số. Phương pháp từ ký hiệu sang số có thể được hiểu là thực hiện chính xác các phép tính tương tự như được thực hiện trong đồ thị được xây dựng bởi phương pháp từ ký hiệu sang ký hiệu. Sự khác biệt chính là phương pháp từ ký hiệu sang số không hiển thị đồ thị.

6.5.6 Lan truyền ngược tổng quát

Thuật toán lan truyền ngược rất đơn giản. Để tính gradient của một số vô hướng z theo một trong các nút tổ tiên $\boldsymbol{x}$ của nó trong đồ thị, ta bắt đầu bằng cách quan sát rằng gradient của z theo chính nó được cho bởi $\frac{dz}{dz} = 1$. Ta có thể tính gradient theo mỗi nút cha của z trong đồ thị bằng cách nhân gradient hiện tại với ma trận Jacobi của phép toán tạo ra z. Ta tiếp tục nhân với các ma trận Jacobi khi di chuyển ngược qua đồ thị theo cách này cho đến khi gặp $\boldsymbol{x}$. Đối với bất kỳ nút nào có thể đến được bằng cách đi ngược từ z qua hai hoặc nhiều con đường, ta chỉ cần cộng các gradient đến từ các con đường khác nhau tại nút đó.

Cách mô tả chính xác hơn, mỗi nút trong đồ thị $\mathcal{G}$ tương ứng với một biến. Để đạt được tính tổng quát tối đa, ta mô tả biến này là một tenxơ $\boldsymbol{\mathcal{V}}$. Tenxơ có thể có bất kỳ số chiều nào. Chúng bao gồm các số vô hướng, véctơ và ma trận.

Chúng ta giả định rằng mỗi biến $\boldsymbol{\mathcal{V}}$ được liên kết với các thủ tục con sau đây:

- get_operation ($\boldsymbol{\mathcal{V}}$): Thủ tục này trả về phép toán tính toán $\boldsymbol{\mathcal{V}}$, được biểu diễn bởi các cạnh đi vào $\boldsymbol{\mathcal{V}}$ trong đồ thị tính toán. Ví dụ, có thể có một lớp Python hoặc C++ đại diện cho phép nhân ma trận, và hàm get_operation sẽ trả về phép toán tương ứng. Giả sử chúng ta có một biến được tạo ra bởi phép nhân ma trận $\boldsymbol{C} = \boldsymbol{AB}$. Khi đó, get_operation ($\boldsymbol{\mathcal{V}}$) trả về một con trỏ đến một thực thể của lớp C++ tương ứng.

- get_consumers ($\boldsymbol{\mathcal{V}}, \mathcal{G}$): Thủ tục này trả về danh sách các biến là con của $\boldsymbol{\mathcal{V}}$ trong đồ thị tính toán $\mathcal{G}$.

- get_inputs ($\boldsymbol{\mathcal{V}}, \mathcal{G}$): Thủ tục này trả về danh sách các biến là cha của $\boldsymbol{\mathcal{V}}$ trong đồ thị tính toán $\mathcal{G}$.

Mỗi phép toán "op" cũng được liên kết với một phép toán "bprop". Phép toán "bprop" này có thể tính một tích ma trận Jacobi với véctơ như được mô tả trong phương trình (6.48). Đây là cách mà thuật toán lan truyền ngược đạt được tính tổng quát cao. Mỗi phép toán chịu trách nhiệm biết cách lan truyền ngược qua các cạnh trong đồ thị mà nó tham gia. Ví dụ, ta có thể sử dụng một phép nhân ma trận để tạo một biến $\boldsymbol{C} = \boldsymbol{AB}$. Giả sử gradient của một vô hướng z theo $\boldsymbol{C}$ được cho bởi $\boldsymbol{G}$. Phép nhân ma trận chịu trách nhiệm định nghĩa hai quy tắc lan truyền ngược, một cho mỗi tham số đầu vào. Nếu ta gọi phương thức "bprop" để yêu cầu gradient theo $\boldsymbol{A}$ khi gradient trên đầu ra là G, thì phương thức "bprop" của phép nhân ma trận phải xác định rằng gradient theo $\boldsymbol{A}$ được cho bởi $\boldsymbol{GB}^\top$. Tương tự, nếu ta gọi

phương thức "bprop" để yêu cầu gradient theo $\boldsymbol{B}$, thì phép nhân ma trận chịu trách nhiệm triển khai phương thức "bprop" và xác định rằng gradient mong muốn được cho bởi $\boldsymbol{A}^\top \boldsymbol{G}$. Thuật toán lan truyền ngược tự nó không cần biết bất kỳ quy tắc tính vi phân nào. Nó chỉ cần gọi các quy tắc "bprop" của từng phép toán với các đối số đúng. Một cách chính thức, "op.bprop (inputs, $\boldsymbol{\mathcal{X}}$, $\boldsymbol{\mathcal{G}}$)" phải trả về:

$$\sum_i \left(\nabla_{\boldsymbol{\mathcal{X}}} \text{op.f(inputs)}_i\right) \mathcal{G}_i. \tag{6.51}$$

Đây chính là một cách triển khai quy tắc chuỗi như đã biểu diễn trong phương trình (6.48). Ở đây "inputs" là danh sách các đầu vào được cung cấp cho phép toán, "op.f" là hàm toán học mà phép toán triển khai, $\boldsymbol{\mathcal{X}}$ là đầu vào mà ta muốn tính gradient, và $\boldsymbol{\mathcal{G}}$ là gradient trên đầu ra của phép toán.

Phương thức "op.bprop" luôn phải giả định rằng tất cả các đầu vào của nó là khác biệt nhau, ngay cả khi chúng không phải. Ví dụ, nếu toán tử "mul" được truyền hai bản sao của x để tính x^2, thì phương thức "op.bprop" vẫn phải trả về x làm đạo hàm theo cả hai đầu vào. Thuật toán lan truyền ngược sau đó sẽ cộng hai giá trị này lại để thu được $2x$, là đạo hàm tổng đúng của x.

Các triển khai phần mềm của lan truyền ngược thường cung cấp cả các phép toán và các phương thức "bprop" tương ứng, để người dùng của các thư viện phần mềm học sâu có thể thực hiện lan truyền ngược qua các đồ thị được xây dựng bằng các phép toán phổ biến như nhân ma trận, lũy thừa, logarit, v.v. Các kỹ sư phần mềm xây dựng một triển khai mới của lan truyền ngược, hoặc những người dùng nâng cao cần thêm phép toán mới vào một thư viện hiện có, thường phải tự suy ra phương thức "op.bprop" cho bất kỳ phép toán mới nào theo cách thủ công.

Thuật toán lan truyền ngược được mô tả chính thức trong Thuật toán 6.5.

Trong Mục 6.5.2, chúng ta đã giải thích rằng thuật toán lan truyền ngược được phát triển nhằm tránh việc tính toán lặp lại cùng một biểu thức con trong quy tắc chuỗi nhiều lần. Thuật toán đơn giản có thể có thời gian chạy theo hàm mũ do những biểu thức con bị lặp lại. Giờ đây, khi đã mô tả cụ thể thuật toán lan truyền ngược, ta có thể hiểu được chi phí tính toán của nó. Nếu giả định rằng mỗi phép toán có chi phí xấp xỉ như nhau, ta có thể phân tích chi phí tính toán dựa trên số lượng phép toán được thực hiện. Ở đây, phép toán được xem là đơn vị cơ bản trong đồ thị tính toán, có thể bao gồm rất nhiều phép toán số học (ví dụ, một đồ thị có thể coi phép nhân ma trận là một phép toán duy nhất). Việc tính gradient trong một đồ thị có n nút sẽ không bao giờ thực hiện quá $O\left(n^2\right)$ phép toán hoặc lưu trữ đầu ra của hơn $O\left(n^2\right)$ phép toán. Ở đây, ta đang đếm các phép toán trong đồ thị tính

Thuật toán 6.5: Bộ khung ngoài cùng của thuật toán lan truyền ngược. Phần này thực hiện các công việc thiết lập và dọn dẹp đơn giản. Hầu hết công việc quan trọng diễn ra trong thủ tục con "build_grad" của Thuật toán 6.6.

Đầu vào:

- T: tập các biến mục tiêu mà ta cần tính gradient của chúng.
- $\mathcal{G}$: đồ thị tính toán.
- z: biến cần lấy đạo hàm.

1 Gọi $\mathcal{G}'$ là đồ thị $\mathcal{G}$ được cắt tỉa để chỉ chứa các nút là tổ tiên của z và hậu duệ của các nút trong T.
2 Khởi tạo "grad_table", một cấu trúc dữ liệu liên kết các tenxơ với gradient của chúng:
grad_table $[z] \leftarrow 1$
3 **for** $\boldsymbol{\mathcal{V}} \in T$ **do**
4 build_grad $\left(\boldsymbol{\mathcal{V}}, \mathcal{G}, \mathcal{G}', \text{grad\_table}\right)$
5 Trả về "grad_table" được giới hạn cho tập T

toán, chứ không phải các phép toán được thực hiện bởi phần cứng nền tảng, vì vậy cần nhớ rằng thời gian chạy của mỗi phép toán có thể rất khác nhau. Ví dụ, việc nhân hai ma trận chứa hàng triệu phần tử mỗi ma trận có thể tương ứng với một phép toán duy nhất trong đồ thị. Ta có thể thấy rằng việc tính gradient yêu cầu tối đa $O\left(n^2\right)$ phép toán bởi vì giai đoạn lan truyền tiến sẽ thực hiện tối đa n nút trong đồ thị gốc (tùy thuộc vào các giá trị cần tính, ta có thể không cần thực hiện toàn bộ đồ thị). Thuật toán lan truyền ngược thêm một tích ma trận Jacobi với véctơ, được biểu diễn với $O(1)$ nút, cho mỗi cạnh trong đồ thị gốc. Vì đồ thị tính toán là một đồ thị có hướng không chu trình, nó có tối đa $O\left(n^2\right)$ cạnh. Trong thực tế, tình hình còn khả quan hơn. Hầu hết các hàm mục tiêu của mạng nơron có cấu trúc dạng chuỗi, khiến chi phí lan truyền ngược chỉ là $O(n)$. Điều này tốt hơn rất nhiều so với phương pháp đơn giản, có thể cần thực hiện một số lượng nút theo cấp số mũ. Chi phí tiềm năng theo hàm mũ có thể được quan sát khi triển khai và viết lại quy tắc

Thuật toán 6.6: Thủ tục con build_grad $\left(\boldsymbol{\mathcal{V}}, \mathcal{G}, \mathcal{G}', \text{grad\_table}\right)$ của thuật toán lan truyền ngược, được gọi bởi thuật toán lan truyền ngược xác định trong Thuật toán 6.5.

Đầu vào:

- $\boldsymbol{\mathcal{V}}$: biến mà gradient của nó cần được thêm vào $\mathcal{G}$ và "grad_table".
- $\mathcal{G}$: đồ thị cần được sửa đổi.
- $\mathcal{G}'$: phiên bản giới hạn của $\mathcal{G}$, chỉ chứa các nút tham gia vào gradient.
- "grad_table": cấu trúc dữ liệu ánh xạ các nút đến gradient của chúng.

if $\boldsymbol{\mathcal{V}} \in$ grad_table **then**
  return grad_table $\left[\boldsymbol{\mathcal{V}}\right]$
$i \leftarrow 1$
for $\boldsymbol{\mathcal{C}} \in$ get_consumers $\left(\boldsymbol{\mathcal{V}}, \mathcal{G}'\right)$ **do**
  op $\leftarrow$ get_operation $(\boldsymbol{\mathcal{C}})$
  $\boldsymbol{\mathcal{D}} \leftarrow$ build_grad $\left(\boldsymbol{\mathcal{C}}, \mathcal{G}, \mathcal{G}', \text{grad\_table}\right)$
  $\boldsymbol{\mathcal{G}}^{(i)} \leftarrow$ op.bprop $\left(\text{get\_inputs}\left(\boldsymbol{\mathcal{C}}, \mathcal{G}'\right), \boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{D}}\right)$
  $i \leftarrow i + 1$
$\boldsymbol{\mathcal{G}} \leftarrow \sum_i \boldsymbol{\mathcal{G}}^{(i)}$
grad_table $\left[\boldsymbol{\mathcal{V}}\right] = \boldsymbol{\mathcal{G}}$
Thêm $\boldsymbol{\mathcal{G}}$ và các phép toán tạo ra nó vào $\mathcal{G}$.
return $\boldsymbol{\mathcal{G}}$

chuỗi đệ quy (phương trình (6.50)) dưới dạng không đệ quy:

$$\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{\substack{\text{đường đi } \left(u^{(\pi_1)}, u^{(\pi_2)}, \ldots, u^{(\pi_t)}\right) \\ \text{từ } \pi_1 = j \text{ đến } \pi_t = n}} \prod_{k=2}^{t} \frac{\partial u^{(\pi_k)}}{\partial u^{(\pi_{k-1})}}. \tag{6.52}$$

Vì số đường đi từ nút j đến nút n có thể tăng theo hàm mũ theo độ dài của các đường này, số số hạng trong tổng ở trên (tức là số lượng các đường đi như vậy) có thể tăng theo hàm mũ với độ sâu của đồ thị lan truyền tiến. Chi phí lớn này phát sinh do cùng một phép tính $\frac{\partial u^{(i)}}{\partial u^{(j)}}$ bị thực hiện lại nhiều lần. Để tránh việc tính toán lặp lại như vậy, chúng ta có thể xem lan truyền ngược như một thuật toán điền vào bảng tận dụng việc lưu trữ các kết quả trung gian $\frac{\partial u^{(n)}}{\partial u^{(i)}}$. Mỗi nút trong đồ

thị có một ô tương ứng trong bảng để lưu trữ gradient cho nút đó. Bằng cách điền các mục trong bảng theo thứ tự, thuật toán lan truyền ngược tránh được việc lặp lại nhiều biểu thức con chung. Chiến lược điền bảng này đôi khi được gọi là **quy hoạch động**.

6.5.7 Ví dụ: lan truyền ngược cho huấn luyện mạng đa lớp

Như một ví dụ, chúng ta sẽ minh họa thuật toán lan truyền ngược khi nó được sử dụng để huấn luyện một mạng đa tầng.

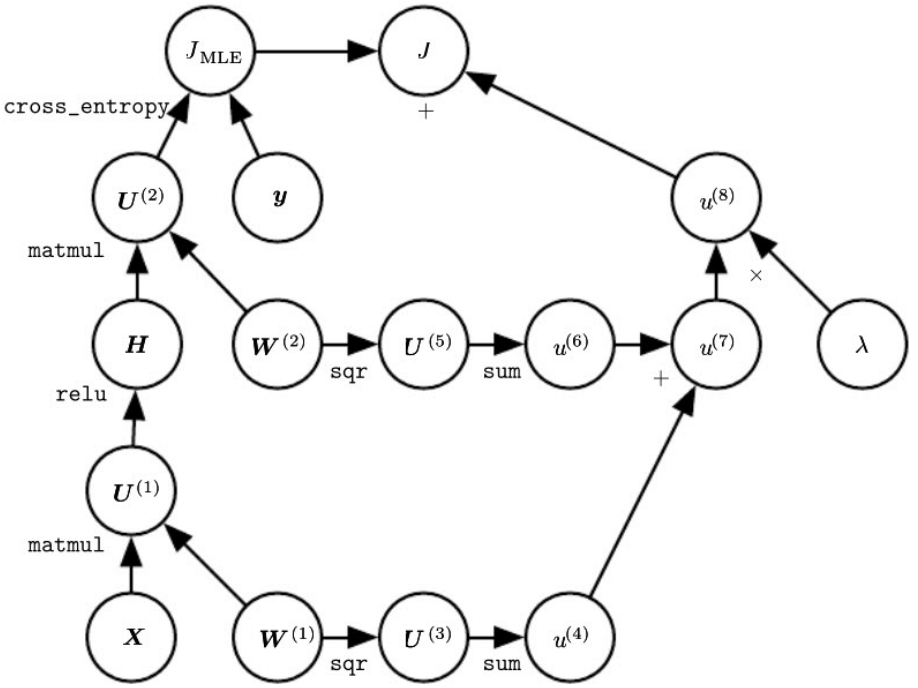
Ở đây, ta phát triển một mạng đa lớp rất đơn giản với một lớp ẩn. Để huấn luyện mô hình này, ta sẽ sử dụng phương pháp hướng giảm ngẫu nhiên theo nhóm mẫu nhỏ. Thuật toán lan truyền ngược được dùng để tính gradient của hàm mất mát trên một nhóm mẫu nhỏ cụ thể. Cụ thể, ta sử dụng một nhóm mẫu nhỏ gồm các ví dụ từ tập huấn luyện, được định dạng như một ma trận thiết kế $\boldsymbol{X}$ và một véctơ các nhãn lớp tương ứng $\boldsymbol{Y}$. Mạng tính toán một lớp các đặc trưng ẩn $\boldsymbol{H} = \max\{0, \boldsymbol{X}\boldsymbol{W}^{(1)}\}$. Để đơn giản hóa, ta không sử dụng độ dời trong mô hình này. Ta giả định rằng ngôn ngữ biểu diễn đồ thị bao gồm một phép toán "relu", có thể tính $\max\{0, \boldsymbol{Z}\}$ trên từng phần tử. Các dự đoán về logarit các xác suất chưa chuẩn hóa trên các lớp được tính bởi $\boldsymbol{H}\boldsymbol{W}^{(2)}$. Ta giả định ngôn ngữ đồ thị bao gồm một phép toán "cross_entropy", dùng để tính entropy chéo giữa các mục tiêu $\boldsymbol{Y}$ và phân phối xác suất được định nghĩa bởi logarit các xác suất chưa chuẩn hóa này. Entropy chéo thu được sẽ định nghĩa hàm mất mát J_{MLE}. Việc cực tiểu hóa entropy chéo này thực hiện phép ước lượng hợp lý cực đại cho bộ phân loại. Tuy nhiên, để làm ví dụ thực tế hơn, ta còn thêm một hạng tử điều chuẩn. Hàm mất mát tổng thể:

$$J = J_{\text{MLE}} + \lambda \left(\sum_{i,j} \left(W_{i,j}^{(1)} \right)^2 + \sum_{i,j} \left(W_{i,j}^{(2)} \right)^2 \right) \tag{6.53}$$

bao gồm entropy chéo và một hạng tử suy giảm trọng số với hệ số λ. Đồ thị tính toán được minh họa trong Hình 6.11.

Đồ thị tính toán cho gradient của ví dụ này đủ lớn để việc vẽ hoặc đọc trở nên tẻ nhạt. Điều này minh họa một trong những lợi ích của thuật toán lan truyền ngược: khả năng tự động tạo ra các gradient mà một kỹ sư phần mềm có thể dễ dàng hiểu nhưng sẽ tốn công sức nếu phải tự dẫn xuất thủ công.

Chúng ta có thể phác thảo sơ bộ cách hoạt động của thuật toán lan truyền ngược bằng cách nhìn vào đồ thị lan truyền xuôi trong Hình 6.11. Để huấn luyện, chúng ta cần tính cả $\nabla_{\boldsymbol{W}^{(1)}} J$ và $\nabla_{\boldsymbol{W}^{(2)}} J$. Có hai đường dẫn khác nhau đi ngược từ



Hình 6.11: Đồ thị tính toán được sử dụng để tính toán hàm mất mát trong việc huấn luyện ví dụ mạng đa lớp chỉ có một lớp, bao gồm tổn thất entropy chéo và suy giảm trọng số.

J đến các trọng số: một đường qua tổn thất entropy chéo và một đường qua tổn thất suy giảm trọng số. Đường dẫn qua tổn thất suy giảm trọng số khá đơn giản; nó luôn đóng góp $2\lambda \boldsymbol{W}^{(i)}$ vào gradient trên $\boldsymbol{W}^{(i)}$.

Đường dẫn còn lại qua tổn thất entropy chéo phức tạp hơn một chút. Gọi $\boldsymbol{G}$ là gradient trên logarit các xác suất chưa chuẩn hóa $\boldsymbol{U}^{(2)}$, được cung cấp bởi phép toán "cross_entropy". Thuật toán lan truyền ngược sau đó cần khám phá hai nhánh khác nhau. Trên nhánh ngắn hơn, thuật toán cộng $\boldsymbol{H}^\top \boldsymbol{G}$ vào gradient trên $\boldsymbol{W}^{(2)}$, sử dụng quy tắc lan truyền ngược cho đối số thứ hai của phép nhân ma trận. Nhánh còn lại tương ứng với chuỗi dài hơn đi sâu hơn vào mạng. Đầu tiên, thuật toán lan truyền ngược tính $\nabla_{\boldsymbol{H}} J = \boldsymbol{G}{\boldsymbol{W}^{(2)}}^\top$, sử dụng quy tắc lan truyền ngược cho đối số thứ nhất của phép nhân ma trận. Tiếp theo, phép toán "relu" sử dụng quy tắc lan truyền ngược của nó để đặt về 0 các thành phần của gradient tương ứng với các phần tử của $U^{(1)}$ nhỏ hơn 0. Kết quả được gọi là $\boldsymbol{G}'$. Bước cuối cùng của thuật toán lan truyền ngược là sử dụng quy tắc lan truyền ngược cho đối số thứ hai của phép nhân ma trận "matmul" để cộng $\boldsymbol{X}^\top \boldsymbol{G}'$ vào gradient trên $\boldsymbol{W}^{(1)}$.

Sau khi các gradient được tính toán, trách nhiệm của thuật toán hướng giảm

hoặc một thuật toán tối ưu nào khác là sử dụng các gradient này để cập nhật các tham số.

Đối với mạng đa lớp này, chi phí tính toán chủ yếu bị chi phối bởi chi phí của phép nhân ma trận. Trong giai đoạn lan truyền tiến, ta nhân với mỗi ma trận trọng số, dẫn đến $O(w)$ phép nhân – cộng, trong đó w là số lượng trọng số. Trong giai đoạn lan truyền ngược, chúng ta nhân với ma trận chuyển vị của mỗi ma trận trọng số, có cùng chi phí tính toán. Chi phí bộ nhớ chính của thuật toán là việc cần lưu trữ đầu vào của hàm phi tuyến tại lớp ẩn. Giá trị này được lưu trữ từ lúc nó được tính toán cho đến khi quá trình lan truyền ngược quay trở lại cùng điểm đó. Do đó, chi phí bộ nhớ là $O(mn_h)$, trong đó m là số lượng mẫu trong nhóm mẫu nhỏ và n_h là số lượng đơn vị ẩn.

6.5.8 Các khía cạnh phức tạp

Mô tả của chúng ta về thuật toán lan truyền ngược ở đây đơn giản hơn so với các triển khai thực tế được sử dụng.

Như đã đề cập ở trên, chúng ta đã giới hạn định nghĩa của một phép toán là một hàm trả về một tenxơ duy nhất. Tuy nhiên, hầu hết các triển khai phần mềm thực tế cần hỗ trợ các phép toán có thể trả về nhiều hơn một tenxơ. Ví dụ, nếu chúng ta muốn tính cả giá trị lớn nhất trong một tenxơ và chỉ số của giá trị đó, cách tốt nhất là tính cả hai trong một lần truy cập bộ nhớ, vì vậy việc triển khai thủ tục này dưới dạng một phép toán duy nhất với hai đầu ra sẽ hiệu quả hơn.

Chúng ta chưa mô tả cách kiểm soát mức tiêu thụ bộ nhớ của lan truyền ngược. Lan truyền ngược thường liên quan đến việc cộng nhiều tenxơ lại với nhau. Trong cách tiếp cận đơn giản, mỗi tensor sẽ được tính riêng lẻ, sau đó tất cả sẽ được cộng trong một bước thứ hai. Cách tiếp cận đơn giản này tạo ra một nút cổ chai về bộ nhớ quá cao, điều này có thể tránh được bằng cách duy trì một bộ đệm duy nhất và cộng từng giá trị vào bộ đệm đó ngay khi nó được tính toán.

Các triển khai thực tế của lan truyền ngược cũng cần xử lý các kiểu dữ liệu khác nhau, chẳng hạn như số thực 32 – bit, số thực 64 – bit và các giá trị nguyên. Chính sách xử lý mỗi kiểu dữ liệu này cần được thiết kế một cách cẩn thận.

Một số phép toán có gradient không xác định, và điều quan trọng là phải theo dõi các trường hợp này để xác định liệu gradient mà người dùng yêu cầu có bị không xác định hay không.

Nhiều chi tiết kỹ thuật khác làm cho việc tính toán đạo hàm trong thực tế trở nên phức tạp hơn. Những chi tiết kỹ thuật này không phải là không thể vượt qua,

và chương này đã mô tả các công cụ trí tuệ quan trọng cần thiết để tính toán đạo hàm, nhưng cần lưu ý rằng còn nhiều vấn đề tinh tế hơn tồn tại.

6.5.9 Phép lấy đạo hàm bên ngoài cộng đồng học sâu

Cộng đồng học sâu có phần tách biệt với cộng đồng khoa học máy tính rộng lớn hơn và đã phát triển các quan điểm văn hóa riêng về cách thực hiện phép lấy đạo hàm. Tổng quát hơn, lĩnh vực **lấy đạo hàm tự động** nghiên cứu cách tính đạo hàm một cách thuật toán. Thuật toán lan truyền ngược được mô tả ở đây chỉ là một cách tiếp cận trong lĩnh vực này. Đây là một trường hợp đặc biệt của một lớp kỹ thuật rộng hơn được gọi là **tích lũy theo chiều ngược**. Các cách tiếp cận khác đánh giá các biểu thức con trong quy tắc chuỗi theo thứ tự khác nhau. Nói chung, việc xác định thứ tự đánh giá sao cho chi phí tính toán là thấp nhất là một bài toán khó. Việc tìm ra trình tự tối ưu của các phép toán để tính gradient là một bài toán "NP-complete" (Naumann, 2008), vì nó có thể yêu cầu đơn giản hóa các biểu thức đại số về dạng ít tốn chi phí nhất.

Ví dụ, giả sử chúng ta có các biến $p_1, p_2, \ldots, p_n$ biểu diễn xác suất và các biến $z_1, z_2, \ldots, z_n$ biểu diễn các giá trị logarit xác suất chưa chuẩn hóa. Giả sử ta định nghĩa:

$$q_i = \frac{e^{z_i}}{\sum_i e^{z_i}}, \tag{6.54}$$

trong đó ta xây dựng hàm softmax từ các phép toán mũ, tổng và chia, và thiết lập hàm mất mát entropy chéo $J = -\sum_i p_i \log q_i$. Một nhà toán học có thể quan sát rằng đạo hàm của J theo z_i có dạng rất đơn giản: $q_i - p_i$. Thuật toán lan truyền ngược không thể tự đơn giản hóa gradient theo cách này, thay vào đó sẽ truyền gradient một cách tường minh qua tất cả các phép toán logarit và mũ trong đồ thị gốc. Một số thư viện phần mềm như Theano (*Theano: A CPU and GPU Math Compiler in Python*, Bergstra và cộng sự, 2010, [41]; *Theano: new features and speed improvements*, Bastien và cộng sự, 2012, [42]) có khả năng thực hiện một số dạng thay thế đại số để cải thiện đồ thị được đề xuất bởi thuật toán lan truyền ngược thuần túy.

Khi đồ thị truyền xuôi $\mathcal{G}$ có một nút đầu ra duy nhất và mỗi đạo hàm riêng $\frac{\partial u^{(i)}}{\partial u^{(j)}}$ có thể được tính toán với một lượng tính toán cố định, lan truyền ngược đảm bảo rằng số lượng tính toán để tính gradient cùng bậc với số lượng tính toán trong truyền xuôi. Điều này có thể được thấy trong Thuật toán 6.2, vì mỗi đạo hàm riêng

địa phương $\frac{\partial u^{(i)}}{\partial u^{(j)}}$ chỉ cần được tính một lần, cùng với một phép nhân và cộng liên quan trong công thức quy tắc chuỗi đệ quy (phương trình (6.50)). Do đó, tổng chi phí tính toán là $O\left(\text{số cạnh}\right)$.

Tuy nhiên, chi phí này có thể giảm hơn nữa bằng cách đơn giản hóa đồ thị tính toán được xây dựng bởi lan truyền ngược, và đây là một bài toán "NP-complete". Các triển khai như Theano và TensorFlow sử dụng các phương pháp thực nghiệm dựa trên việc khớp các mẫu đơn giản hóa đã biết để cố gắng đơn giản hóa đồ thị một cách lặp đi lặp lại. Lan truyền ngược được định nghĩa chỉ để tính gradient của một đầu ra vô hướng, nhưng nó có thể được mở rộng để tính ma trận Jacobi (hoặc của k nút vô hướng khác nhau trong đồ thị, hoặc của một nút giá trị tenxơ chứa k giá trị). Một triển khai đơn giản có thể yêu cầu tính toán nhiều hơn gấp k lần: đối với mỗi nút trong vô hướng trong đồ thị truyền tiến ban đầu, triển khai này tính toán k gradient thay vì chỉ một gradient. Khi số đầu ra của đồ thị lớn hơn số đầu vào, đôi khi nên sử dụng một dạng lấy đạo hàm tự động khác gọi là **tích lũy theo chiều xuôi**. Phương pháp này đã được đề xuất để tính gradient trong mạng hồi quy theo thời gian thực, ví dụ như trong công trình *A Learning Algorithm for Continually Running Fully Recurrent Neural Networks* (Williams và Zipser, 1989, [44]). Phương pháp này cũng tránh được việc phải lưu trữ toàn bộ giá trị và gradient cho đồ thị, đánh đổi hiệu quả tính toán để tiết kiệm bộ nhớ. Quan hệ giữa tích lũy theo chiều xuôi và chiều ngược tương tự như quan hệ giữa nhân ma trận từ trái sang phải so với từ phải sang trái, ví dụ:

$$\boldsymbol{ABCD},$$

trong đó các ma trận có thể được coi là các ma trận Jacobi. Ví dụ, nếu $\boldsymbol{D}$ là một véctơ cột, trong khi $\boldsymbol{A}$ có nhiều hàng, điều này tương ứng với một đồ thị có một đầu ra và nhiều đầu vào. Bắt đầu nhân từ cuối và đi ngược lại chỉ yêu cầu các phép nhân ma trận với véctơ, tương ứng với phương pháp lan truyền ngược. Ngược lại, nếu bắt đầu nhân từ trái sang phải sẽ bao gồm một chuỗi các phép nhân ma trận với ma trận, khiến cho việc tính toán toàn bộ tốn kém hơn nhiều. Tuy nhiên, nếu $\boldsymbol{A}$ có ít hàng hơn số cột của $\boldsymbol{D}$, sẽ rẻ hơn khi thực hiện các phép nhân từ trái sang phải, tương ứng với phương pháp lan truyền tiến.

Trong nhiều cộng đồng ngoài lĩnh vực học máy, việc triển khai phần mềm tính đạo hàm thường dựa trực tiếp trên mã của các ngôn ngữ lập trình truyền thống như Python hoặc C, và tự động tạo ra các chương trình để tính đạo hàm của các hàm được viết bằng những ngôn ngữ này. Trong cộng đồng học sâu, đồ thị tính toán thường được biểu diễn bởi các cấu trúc dữ liệu rõ ràng được tạo ra bởi các thư viện

chuyên biệt. Cách tiếp cận chuyên biệt có nhược điểm là yêu cầu nhà phát triển thư viện phải định nghĩa các phương thức "bprop" cho mọi phép toán và giới hạn người dùng của thư viện chỉ trong các phép toán đã được định nghĩa. Tuy nhiên, nó cũng mang lại lợi ích đáng kể khi cho phép phát triển các quy tắc lan truyền ngược được tùy chỉnh riêng cho từng phép toán. Điều này giúp nhà phát triển cải thiện tốc độ hoặc độ ổn định theo những cách không hiển nhiên mà một quy trình tự động khó có thể lặp lại.

Do đó, lan truyền ngược không phải là cách duy nhất hoặc cách tối ưu để tính gradient, nhưng nó là một phương pháp rất thực tiễn và tiếp tục phục vụ tốt cộng đồng học sâu. Trong tương lai, công nghệ tính đạo hàm cho các mạng học sâu có thể được cải thiện khi các nhà nghiên cứu học sâu ngày càng nhận thức được những tiến bộ trong lĩnh vực rộng lớn hơn của tự động hóa tính đạo hàm.

6.5.10 Đạo hàm cấp cao

Một số khung phần mềm hỗ trợ việc sử dụng các đạo hàm cấp cao. Trong số các khung phần mềm học sâu, ít nhất Theano và TensorFlow đã cung cấp tính năng này. Những thư viện này sử dụng cùng một loại cấu trúc dữ liệu để mô tả biểu thức cho đạo hàm như cách chúng mô tả hàm ban đầu được tính đạo hàm. Điều này có nghĩa là cơ chế đạo hàm ký hiệu có thể được áp dụng cho cả các đạo hàm.

Trong bối cảnh học sâu, việc tính một đạo hàm bậc hai của một hàm vô hướng là khá hiếm. Thay vào đó, ta thường quan tâm đến các tính chất của ma trận Hess. Với một hàm $f : \mathbb{R}^n \to \mathbb{R}$, ma trận Hess có cỡ $n \times n$. Trong các ứng dụng học sâu điển hình, n là số lượng tham số của mô hình, con số này có thể dễ dàng lên đến hàng tỷ. Vì vậy, việc biểu diễn toàn bộ ma trận Hess là không khả thi.

Thay vì tính toán trực tiếp ma trận Hess, cách tiếp cận phổ biến trong học sâu là sử dụng các **phương pháp Krylov**. Đây là một tập hợp các kỹ thuật lặp để thực hiện nhiều thao tác khác nhau, chẳng hạn như xấp xỉ nghịch đảo của ma trận, hay tìm các xấp xỉ của véctơ riêng hoặc giá trị riêng. Điểm đặc biệt của các phương pháp Krylov là chúng chỉ yêu cầu phép nhân ma trận – véctơ.

Để sử dụng các phương pháp Krylov với ma trận Hess, ta chỉ cần có khả năng tính tích của ma trận Hess $\boldsymbol{H}$ với một véctơ bất kỳ $\boldsymbol{v}$. Một kỹ thuật đơn giản (*Automatic Hessians by Reverse Accumulation*, Christianson, 1992, [120]) để thực hiện điều này là tính:

$$\boldsymbol{H}\boldsymbol{v} = \nabla_{\boldsymbol{x}} \left[\left(\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \right)^{\top} \boldsymbol{v} \right]. \tag{6.55}$$

Cả hai tính toán gradient trong biểu thức này đều có thể tính toán tự động

bằng thư viện phần mềm thích hợp. Chú ý rằng biểu gradient bên ngoài dấu ngoặc vuông lấy gradient của một hàm được định nghĩa bởi gradient bên trong.

Nếu $\boldsymbol{v}$ là một véctơ được sinh ra bởi đồ thị tính toán, cần chỉ rõ rằng phần mềm lấy đạo hàm tự động không nên tính đạo hàm qua đồ thị đã tạo ra $\boldsymbol{v}$. Điều này đảm bảo tính đúng đắn và hiệu quả của phép tính.

Trong trường hợp cần thiết phải tính toán toàn bộ ma trận Hess (dù thường không được khuyến khích), ta có thể thực hiện bằng phép lấy tích ma trận Hess với véctơ. Đơn giản ta tính $\boldsymbol{H}\boldsymbol{e}^{(i)}$ với $i = 1, \dots, n$, trong đó $\boldsymbol{e}^{(i)}$ là véctơ one – hot với $e_i^{(i)} = 1$ và tất cả các phần tử khác bằng 0. Phương pháp này cho phép tính từng cột của ma trận Hess, nhưng thường không thực tế trong các mô hình học sâu lớn do chi phí tính toán và bộ nhớ cao.

6.6 Ghi chú lịch sử

Mạng nơ-ron truyền thẳng có thể được xem là các bộ xấp xỉ hàm phi tuyến hiệu quả, được xây dựng dựa trên việc sử dụng phương pháp hướng giảm để cực tiểu hóa sai số trong bài toán xấp xỉ hàm. Từ góc độ này, mạng nơron truyền thẳng hiện đại là kết quả của hàng thế kỷ tiến bộ trong bài toán xấp xỉ hàm tổng quát.

Quy tắc chuỗi, nền tảng của thuật toán lan truyền ngược, được phát minh vào thế kỷ 17 (Leibniz, 1676; L'Hôpital, 1696). Trong lịch sử, phép tính vi phân và đại số đã được sử dụng để giải các bài toán tối ưu dưới dạng đóng, nhưng phương pháp hướng giảm như một kỹ thuật xấp xỉ lặp để giải các bài toán tối ưu chỉ được giới thiệu vào thế kỷ 19 (Cauchy, 1847).

Từ những năm 1940, các kỹ thuật xấp xỉ hàm này đã được sử dụng để thúc đẩy sự phát triển của các mô hình học máy như một loại nhận thức. Tuy nhiên, các mô hình ban đầu chủ yếu dựa trên mô hình tuyến tính. Những nhà phê bình, bao gồm Marvin Minsky, đã chỉ ra một số hạn chế của chúng, chẳng hạn như việc không thể học được hàm XOR, điều này dẫn đến sự đối lập với cách tiếp cận mạng nơron nói chung.

Học các hàm phi tuyến đòi hỏi sự phát triển của mạng nhận thức đa lớp và một phương pháp tính gradient thông qua mô hình này. Các ứng dụng hiệu quả của quy tắc chuỗi dựa trên quy hoạch động bắt đầu xuất hiện vào những năm 1960 và 1970, chủ yếu trong các ứng dụng điều khiển (Kelley, 1960; Bryson và Denham, 1961; Dreyfus, 1962; Bryson và Ho, 1969; Dreyfus, 1973) cũng như trong phân tích độ nhạy (Linnainmaa, 1976). Werbos (1981) đã đề xuất áp dụng các kỹ thuật này

để huấn luyện mạng nơron nhân tạo. Ý tưởng này cuối cùng đã được phát triển và thực hiện trong thực tiễn sau khi được độc lập tái phát hiện theo nhiều cách khác nhau (LeCun, 1985; Parker, 1985; Rumelhart và cộng sự, 1986a). Cuốn sách **Parallel Distributed Processing** đã giới thiệu các kết quả của một số thí nghiệm thành công đầu tiên với thuật toán lan truyền ngược trong một chương (Rumelhart và cộng sự, 1986b), góp phần đáng kể vào việc phổ biến thuật toán lan truyền ngược và khởi xướng một giai đoạn nghiên cứu rất sôi động về mạng nơron đa lớp. Tuy nhiên, các ý tưởng được đưa ra bởi các tác giả của cuốn sách này, đặc biệt là Rumelhart và Hinton, còn vượt xa thuật toán lan truyền ngược. Chúng bao gồm các ý tưởng quan trọng về việc triển khai tính toán của nhiều khía cạnh cốt lõi trong nhận thức và học, được gọi chung là "kết nối luận", nhấn mạnh tầm quan trọng của các kết nối giữa các nơron như là trung tâm của việc học và ghi nhớ. Đặc biệt, những ý tưởng này bao gồm khái niệm về biểu diễn phân tán (Hinton và cộng sự, 1986).

Sau thành công của thuật toán lan truyền ngược, nghiên cứu về mạng nơron đã trở nên phổ biến và đạt đỉnh vào đầu những năm 1990. Sau đó, các kỹ thuật học máy khác dần trở nên thịnh hành hơn cho đến thời kỳ phục hưng học sâu hiện đại bắt đầu vào năm 2006.

Những ý tưởng cốt lõi đằng sau các mạng truyền thẳng hiện đại không thay đổi đáng kể kể từ những năm 1980. Thuật toán lan truyền ngược và các phương pháp tiếp cận dựa trên phương pháp hướng giảm vẫn được sử dụng. Hầu hết sự cải thiện về hiệu suất của mạng nơron từ năm 1986 đến 2015 có thể được giải thích bởi hai yếu tố chính. Thứ nhất, các tập dữ liệu lớn hơn đã làm giảm đáng kể thách thức về khả năng tổng quát hóa thống kê của mạng nơron. Thứ hai, các mạng nơron đã trở nên lớn hơn rất nhiều nhờ vào sự phát triển của máy tính mạnh hơn và cơ sở hạ tầng phần mềm tốt hơn. Tuy nhiên, một số ít các thay đổi thuật toán đã cải thiện đáng kể hiệu suất của mạng nơron.

Một trong những thay đổi thuật toán đó là việc thay thế hàm sai số bình phương trung bình bằng các hàm mất mát thuộc họ hàm entropy chéo. Vào những năm 1980 và 1990, hàm sai số bình phương trung bình rất phổ biến nhưng dần được thay thế bằng hàm mất mát entropy chéo và nguyên tắc nguyên lý hợp lý cực đại khi các ý tưởng này lan tỏa giữa cộng đồng thống kê và cộng đồng học máy. Việc sử dụng hàm mất mát entropy chéo đã cải thiện đáng kể hiệu suất của các mô hình sử dụng hàm sigmoid và softmax ở đầu ra, vốn trước đây gặp phải vấn đề bão hòa và học chậm khi sử dụng hàm mất mát sai số bình phương trung bình.

Thay đổi thuật toán quan trọng khác đã cải thiện đáng kể hiệu suất của mạng

truyền thẳng là việc thay thế các đơn vị ẩn sigmoid bằng các đơn vị ẩn tuyến tính từng phần, như các đơn vị tuyến tính chỉnh lưu. Chỉnh lưu sử dụng hàm $\max\{0, z\}$ đã được giới thiệu trong các mô hình mạng nơron từ rất sớm, ít nhất là từ "Cognitron and Neocognitron" (Fukushima, 1975, 1980). Những mô hình này không sử dụng đơn vị tuyến tính chỉnh lưu, mà áp dụng chỉnh lưu cho các hàm phi tuyến khác. Mặc dù chỉnh lưu phổ biến từ sớm, chúng đã bị thay thế bởi sigmoid trong những năm 1980, có lẽ vì sigmoid hoạt động tốt hơn khi mạng nơron còn rất nhỏ. Vào đầu những năm 2000, các đơn vị tuyến tính chỉnh lưu bị tránh sử dụng do niềm tin sai lầm rằng cần phải tránh các hàm kích hoạt có điểm không khả vi. Điều này bắt đầu thay đổi vào khoảng năm 2009. Jarrett và cộng sự (2009) nhận thấy rằng "việc sử dụng một hàm phi tuyến chỉnh lưu là yếu tố quan trọng nhất trong việc cải thiện hiệu suất của hệ thống nhận diện" so với các yếu tố thiết kế kiến trúc mạng khác.

Đối với các tập dữ liệu nhỏ, Jarrett và cộng sự (2009) nhận thấy rằng việc sử dụng hàm phi tuyến chỉnh lưu thậm chí còn quan trọng hơn so với việc học trọng số của các lớp ẩn. Các trọng số ngẫu nhiên đủ để truyền tải thông tin hữu ích qua một mạng ReLU, giúp lớp phân loại ở đầu ra học cách ánh xạ các véctơ đặc trưng thành các nhãn lớp.

Khi có nhiều dữ liệu hơn, việc học bắt đầu trích xuất đủ thông tin hữu ích để vượt qua hiệu suất của các trọng số được chọn ngẫu nhiên. Glorot và cộng sự (2011a) đã chỉ ra rằng việc huấn luyện trở nên dễ dàng hơn nhiều trong các mạng ReLU sâu so với các mạng sâu sử dụng hàm kích hoạt có độ cong hoặc hiện tượng bão hòa hai phía.

Các đơn vị tuyến tính chỉnh lưu cũng có ý nghĩa lịch sử vì chúng minh họa rằng khoa học thần kinh tiếp tục ảnh hưởng đến sự phát triển của các thuật toán học sâu. Glorot và cộng sự (2011a) đã đưa ra động cơ cho việc sử dụng ReLU từ các cân nhắc sinh học. Hàm phi tuyến bán chỉnh lưu được thiết kế để mô phỏng các đặc tính này của nơron sinh học: 1) Với một số đầu vào, nơron sinh học hoàn toàn không hoạt động. 2) Với một số đầu vào khác, đầu ra của nơron sinh học tỷ lệ thuận với đầu vào của nó. 3) Hầu hết thời gian, các nơron sinh học vận hành trong trạng thái không hoạt động (tức là **các kích hoạt nên thưa thớt**).

Khi sự hồi sinh hiện đại của học sâu bắt đầu vào năm 2006, các mạng truyền thẳng vẫn bị đánh giá thấp. Trong khoảng thời gian từ 2006 đến 2012, nhiều người tin rằng mạng truyền thẳng sẽ không hoạt động tốt trừ khi được hỗ trợ bởi các mô hình khác như mô hình xác suất. Ngày nay, với các nguồn lực và kỹ thuật kỹ thuật phù hợp, mạng truyền thẳng đã chứng minh khả năng hoạt động xuất sắc. Hiện nay, học dựa trên gradient trong mạng truyền thẳng được sử dụng như một công

cụ để phát triển các mô hình xác suất, như bộ mã hóa tự động biến phân và mạng sinh đối kháng (được mô tả trong chương 20). Thay vì bị coi là công nghệ không đáng tin cậy cần được hỗ trợ bởi các kỹ thuật khác, kể từ năm 2012, học dựa trên gradient trong mạng truyền thẳng đã được xem là công nghệ mạnh mẽ có thể áp dụng cho nhiều nhiệm vụ học máy khác. Trước đây, vào năm 2006, cộng đồng học sâu sử dụng học không giám sát để hỗ trợ học có giám sát. Tuy nhiên, trớ trêu thay, hiện nay việc sử dụng học có giám sát để hỗ trợ học không giám sát lại phổ biến hơn.

Mạng truyền thẳng vẫn còn nhiều tiềm năng chưa được khai thác. Trong tương lai, chúng được kỳ vọng sẽ được áp dụng cho nhiều nhiệm vụ hơn nữa, và các tiến bộ trong thuật toán tối ưu hóa cũng như thiết kế mô hình sẽ tiếp tục cải thiện hiệu suất của chúng. Chương này chủ yếu mô tả họ mô hình mạng nơron. Trong các chương tiếp theo, chúng ta sẽ chuyển sang cách sử dụng các mô hình này – cách điều chuẩn và huấn luyện chúng.