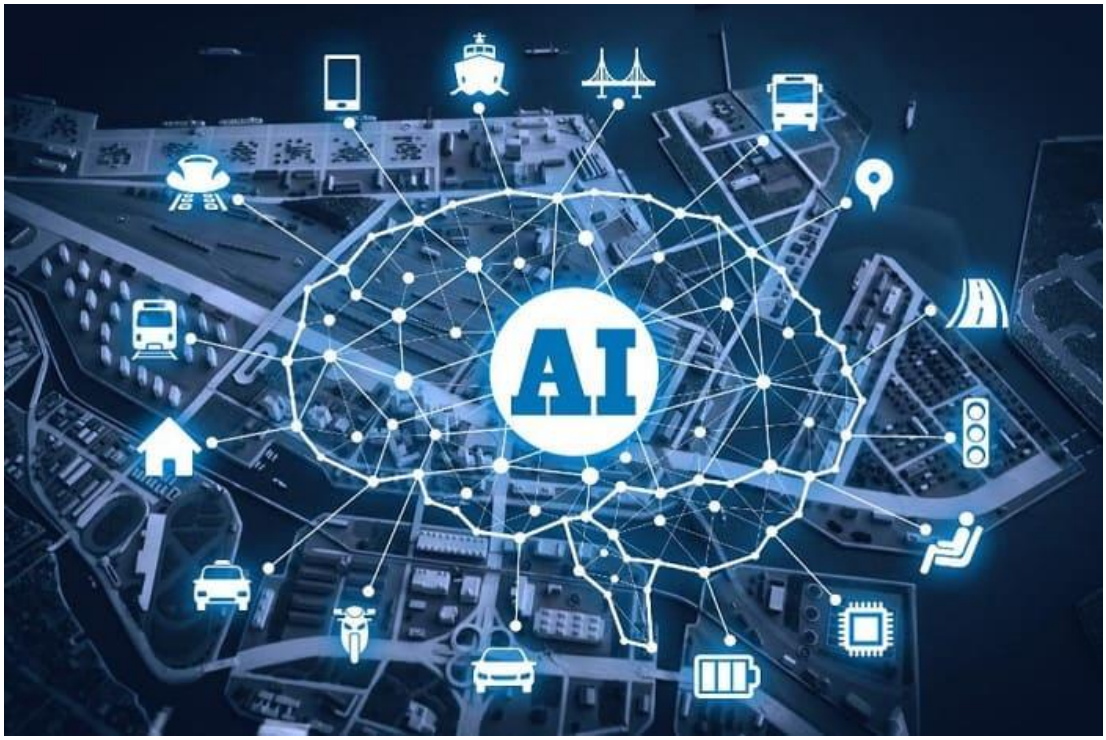
TS. NGUYỄN MINH TUẤN

BÀI GIẢNG
TRÍ TUỆ NHÂN TẠO VÀ ỨNG DỤNG



HƯNG YÊN 2023

CHƯƠNG 1

TỔNG QUAN VỀ TRÍ TUỆ NHÂN TẠO

1.1. Khái niệm trí tuệ nhân tạo

Trong khoa học máy tính, trí tuệ nhân tạo (Artificial Intelligence AI) là trí thông minh được thể hiện bằng máy móc, trái ngược với trí thông minh tự nhiên của con người. Thông thường, thuật ngữ "trí tuệ nhân tạo" thường được sử dụng để mô tả các máy móc (hoặc máy tính) có khả năng bắt chước các chức năng "nhận thức" mà con người thường phải liên kết với tâm trí, như "học tập" và "giải quyết vấn đề".

Khi máy móc ngày càng tăng khả năng, các nhiệm vụ được coi là cần "trí thông minh" thường bị loại bỏ khỏi định nghĩa về AI, một hiện tượng được gọi là hiệu ứng AI. Một câu châm ngôn trong Định lý của Tesler nói rằng "AI là bất cứ điều gì chưa được thực hiện." Ví dụ, nhận dạng ký tự quang học thường bị loại trừ khỏi những thứ được coi là AI, đã trở thành một công nghệ thông thường. Khả năng máy hiện đại thường được phân loại như AI bao gồm thành công hiểu lời nói của con người, cạnh tranh ở mức cao nhất trong trò chơi chiến lược (chẳng hạn như cờ vua và Go), xe hoạt động độc lập, định tuyến thông minh trong mạng phân phối nội dung, và mô phỏng quân sự.

Trí tuệ nhân tạo có thể được phân thành ba loại hệ thống khác nhau: trí tuệ nhân tạo phân tích, lấy cảm hứng từ con người và nhân tạo. AI phân tích chỉ có các đặc điểm phù hợp với trí tuệ nhận thức; tạo ra một đại diện nhận thức về thế giới và sử dụng học tập dựa trên kinh nghiệm trong quá khứ để thông báo các quyết định trong tương lai. AI lấy cảm hứng từ con người có các yếu tố từ trí tuệ nhận thức và cảm xúc; hiểu cảm xúc của con người, ngoài các yếu tố nhận thức và xem xét chúng trong việc ra quyết định. AI nhân cách hóa cho thấy các đặc điểm của tất cả các loại năng lực (nghĩa là trí tuệ nhận thức, cảm xúc và xã hội), có khả năng tự ý thức và tự nhận thức được trong các tương tác.

Trí tuệ nhân tạo được thành lập như một môn học thuật vào năm 1956, và trong những năm sau đó đã trải qua nhiều làn sóng lạc quan, sau đó là sự thất vọng và mất kinh phí (được gọi là " mùa đông AI "), tiếp theo là cách tiếp cận mới, thành công và tài trợ mới. Trong phần lớn lịch sử của mình, nghiên cứu AI đã được chia thành các trường con thường không liên lạc được với nhau. Các trường con này dựa trên các cân nhắc kỹ thuật, chẳng hạn như các mục tiêu cụ thể (ví dụ: " robot học " hoặc "học máy"), việc sử dụng các công cụ cụ thể ("logic" hoặc mạng lưới thần kinh nhân tạo) hoặc sự khác biệt triết học sâu sắc. Các ngành con cũng được dựa trên các yếu tố xã hội (các tổ chức cụ thể hoặc công việc của các nhà nghiên cứu cụ thể).

Lĩnh vực này được thành lập dựa trên tuyên bố rằng trí thông minh của con người "có thể được mô tả chính xác đến mức một cỗ máy có thể được chế tạo để mô phỏng nó". Điều này làm dấy lên những tranh luận triết học về bản chất của tâm trí và đạo đức khi tạo ra những sinh vật nhân tạo có trí thông minh giống con người, đó là những vấn đề đã được thần thoại, viễn tưởng và triết học từ thời cổ đại đề cập tới. Một số người cũng coi AI là mối nguy hiểm cho nhân loại nếu tiến triển của nó không suy giảm. Những người khác tin rằng AI, không giống như các cuộc cách mạng công nghệ trước đây, sẽ tạo ra nguy cơ thất nghiệp hàng loạt.

Trong thế kỷ 21, các kỹ thuật AI đã trải qua sự hồi sinh sau những tiến bộ đồng thời về **sức mạnh máy tính, dữ liệu lớn và hiểu biết lý thuyết**; và kỹ thuật AI đã trở thành một phần thiết yếu của ngành công nghệ, giúp giải quyết nhiều vấn đề thách thức trong học máy, công nghệ phần mềm và nghiên cứu vận hành.

1.2. Lịch sử trí tuệ nhân tạo

Tư tưởng có khả năng sinh vật nhân tạo xuất hiện như các thiết bị kể chuyện thời cổ đại, và đã được phổ biến trong tiểu thuyết, như trong Frankenstein của Mary Shelley hay RUR (máy toàn năng Rossum) của Karel Capek. Những nhân vật này và số phận của họ nêu ra nhiều vấn đề tương tự hiện đang được thảo luận trong đạo đức của trí tuệ nhân tạo.

Nghiên cứu về lý trí cơ học hoặc "chính thức" bắt đầu với các nhà triết học và toán học thời cổ đại. Nghiên cứu về logic toán học đã dẫn trực tiếp đến lý thuyết tính toán của Alan Turing, người cho rằng một cỗ máy, bằng cách xáo trộn các ký hiệu đơn giản như "0" và "1", có thể mô phỏng bất kỳ hành động suy luận toán học nào có thể hiểu được. Tầm nhìn sâu sắc này, cho thấy máy tính kỹ thuật số có thể mô phỏng bất kỳ quá trình suy luận hình thức nào, đã được gọi là luận án Church-Turing. Cùng với những khám phá đồng thời về sinh học thần kinh, lý thuyết thông tin và điều khiển học, điều này khiến các nhà nghiên cứu cân nhắc khả năng xây dựng bộ não điện tử. Turing đã đề xuất rằng "nếu một con người không thể phân biệt giữa các phản hồi từ một máy và một con người, máy tính có thể được coi là 'thông minh'. Công việc đầu tiên mà bây giờ được công nhận là trí tuệ nhân tạo là thiết kế hình thức "tế bào thần kinh nhân tạo" do McCullouch và Pitts đưa ra năm 1943.

Lĩnh vực nghiên cứu AI được ra đời tại một hội thảo tại Đại học Dartmouth năm 1956. Những người tham dự Allen Newell (CMU), Herbert Simon (CMU), John McCarthy (MIT), Marvin Minsky (MIT) và Arthur Samuel (IBM) đã trở thành những người sáng lập và lãnh đạo nghiên cứu AI. Họ và các sinh viên của họ đã tạo ra các chương trình mà báo chí mô tả là "đáng kinh ngạc": máy tính đang học chiến lược kiểm tra và đến năm 1959 được cho là chơi tốt hơn người bình thường, giải từ các vấn đề về đại số, chứng minh các định lý logic (Lý thuyết logic, lần chạy đầu tiên vào năm 1956) và nói tiếng Anh. Đến giữa thập niên 1960, nghiên cứu ở Mỹ được Bộ Quốc phòng tài trợ rất nhiều và các phòng thí nghiệm đã được thành lập trên khắp thế giới. Những người sáng lập AI rất lạc quan về tương lai: Herbert Simon dự đoán, "máy móc sẽ có khả năng, trong vòng hai mươi năm nữa, làm bất kỳ công việc nào mà một người có thể làm". Marvin Minsky đồng ý, viết, "trong một thế hệ ... Vấn đề tạo ra 'trí tuệ nhân tạo' về cơ bản sẽ được giải quyết ". Tuy nhiên, họ đã không nhận ra độ khó của một số nhiệm vụ còn lại. Tiến độ chậm lại và vào năm 1974, để đáp lại sự chỉ trích của Sir James Lighthill và áp lực liên tục từ Quốc hội Hoa Kỳ để tài trợ cho các dự án năng suất cao hơn, cả chính phủ Hoa Kỳ và Anh đều dừng nghiên cứu khám phá về AI. Vài năm sau đó sẽ được gọi là " mùa đông AI ", giai đoạn mà việc kiếm được tài trợ cho các dự án AI là khó khăn.

Đầu những năm 1980, nghiên cứu AI đã được hồi sinh nhờ thành công thương mại của các hệ chuyên gia, một dạng chương trình AI mô phỏng kiến thức và kỹ năng phân tích của các chuyên gia về con người. Đến năm 1985, thị trường cho AI đã đạt hơn một tỷ đô la. Đồng thời, dự án máy tính thế hệ thứ năm của Nhật Bản đã truyền cảm hứng cho chính phủ Hoa Kỳ và Anh khôi phục tài trợ cho nghiên cứu học thuật. Tuy nhiên, bắt đầu với sự sụp đổ của thị trường Máy Lisp vào năm 1987, AI một lần nữa rơi vào tình trạng khó khăn, và một sự gián đoạn thứ hai, kéo dài hơn đã bắt đầu.

Vào cuối những năm 1990 và đầu thế kỷ 21, AI bắt đầu được sử dụng cho hậu cần, khai thác dữ liệu, chẩn đoán y tế và các lĩnh vực khác. Thành công là nhờ sức mạnh tính toán ngày càng tăng (xem định luật Moore), nhấn mạnh hơn vào việc giải quyết các vấn đề cụ thể, mối quan hệ mới giữa AI và các lĩnh vực khác (như thống kê, kinh tế và toán học) và cam kết của các nhà nghiên cứu về phương pháp toán học và tiêu chuẩn khoa học. Deep Blue trở thành hệ thống chơi cờ trên máy tính đầu tiên đánh bại một nhà đương kim vô địch cờ vua thế giới, Garry Kasparov, vào ngày 11 tháng 5 năm 1997.

Năm 2011, tại một chương trình truyền hình thi đấu trả lời câu hỏi biểu diễn Jeopardy!, hệ thống trả lời câu hỏi của IBM, Watson, đã đánh bại hai nhà vô địch Brad Rutter và Ken Jennings, với tỷ số chênh lệch đáng kể. Máy tính nhanh hơn, cải tiến thuật toán và truy cập vào lượng lớn dữ liệu cho phép có được các tiến bộ trong học tập và nhận thức máy; phương pháp học sâu vốn đói dữ liệu bắt đầu thống trị các thử nghiệm liên quan đến độ chính xác vào khoảng năm 2012. Kinect, cung cấp giao diện chuyển động cơ thể 3D cho Xbox 360 và Xbox One, sử dụng các thuật toán xuất hiện từ nghiên cứu AI dài cũng như trợ lý cá nhân thông minh trong điện thoại thông minh. Vào tháng 3 năm 2016, AlphaGo đã thắng 4 trên 5 trận đấu cờ vây trong trận đấu với nhà vô địch cờ vây Lee Sedol, trở thành hệ thống chơi cờ vây trên máy tính đầu tiên đánh bại một người chơi cờ vây chuyên nghiệp mà không cần chấp quân. Trong Hội nghị Tương lai 2017, AlphaGo đã giành chiến thắng trong một trận đấu ba ván với Kha Khiết, kỳ thủ lúc đó liên tục giữ vị trí số 1 thế giới trong hai năm. Điều này đánh dấu sự hoàn thành một cột mốc quan trọng trong sự phát triển của trí tuệ nhân tạo vì cờ vây là một trò chơi tương đối phức tạp, hơn cả cờ vua.

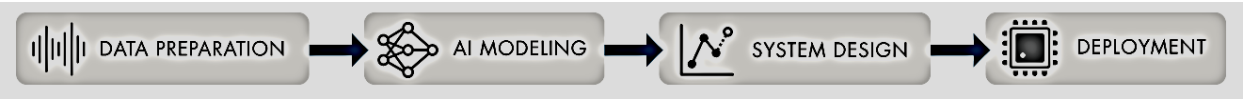
Theo Jack Clark của Bloomberg, năm 2015 là một năm mang tính bước ngoặt đối với trí tuệ nhân tạo, với số lượng dự án phần mềm sử dụng AI Google đã tăng từ "sử dụng lẻ tẻ" vào năm 2012 lên hơn 2.700 dự án. Clark cũng trình bày dữ liệu thực tế cho thấy những cải tiến của AI kể từ năm 2012 được hỗ trợ bởi tỷ lệ lỗi thấp hơn trong các tác vụ xử lý hình ảnh. Ông cho rằng sự gia tăng các mạng thần kinh giá cả phải chăng, do sự gia tăng cơ sở hạ tầng điện toán đám mây và sự gia tăng các công cụ nghiên cứu và bộ dữ liệu. Các ví dụ được trích dẫn khác bao gồm sự phát triển hệ thống Skype của Microsoft có thể tự động dịch từ ngôn ngữ này sang ngôn ngữ khác và hệ thống của Facebook có thể mô tả hình ảnh cho người mù. Trong một cuộc khảo sát năm 2017, một trong năm công ty báo cáo rằng họ đã "kết hợp AI trong một số dịch vụ hoặc quy trình".

Khoảng năm 2016, Trung Quốc đã tăng tốc rất nhiều tài trợ của chính phủ; với nguồn cung cấp dữ liệu lớn và sản lượng nghiên cứu tăng nhanh, một số nhà quan sát tin rằng nước này có thể đang trên đà trở thành một "siêu cường AI". Tuy nhiên, người ta đã thừa nhận rằng các báo cáo liên quan đến trí tuệ nhân tạo có xu hướng bị phóng đại.

1.3. Mô hình AI

1.3.1. Các thành phần chính của quy trình làm việc AI

Thành công với AI đòi hỏi nhiều hơn là đào tạo một mô hình AI, đặc biệt là trong các hệ thống do AI điều khiển đưa ra quyết định và hành động. Một quy trình công việc AI vững chắc bao gồm việc chuẩn bị dữ liệu, tạo mô hình, thiết kế hệ thống mà mô hình sẽ chạy trên đó và triển khai vào phần cứng hoặc hệ thống doanh nghiệp.



Hình 1.1. Các bước trong quy trình làm việc AI

1.3.1.1. Chuẩn bị dữ liệu

Lấy dữ liệu thô và làm cho nó hữu ích cho một mô hình chính xác, hiệu quả và có ý nghĩa là một bước quan trọng. Trên thực tế, nó đại diện cho phần lớn nỗ lực AI.

Việc chuẩn bị dữ liệu đòi hỏi kiến thức chuyên môn về miền, chẳng hạn như kinh nghiệm về tín hiệu giọng nói và âm thanh, điều hướng và kết hợp cảm biến, xử lý hình ảnh và video cũng như radar và lidar. Các kỹ sư trong các lĩnh vực này phù hợp nhất để xác định tính năng quan trọng của dữ liệu là gì, tính năng nào không quan trọng và những sự kiện hiếm gặp nào cần xem xét.

AI cũng liên quan đến lượng dữ liệu khổng lồ. Tuy nhiên, việc ghi nhãn dữ liệu và hình ảnh là tẻ nhạt và tốn thời gian. Trong trường hợp không có đủ dữ liệu, đặc biệt là đối với các hệ thống quan trọng về an toàn, việc tạo dữ liệu tổng hợp chính xác có thể cải thiện bộ dữ liệu.

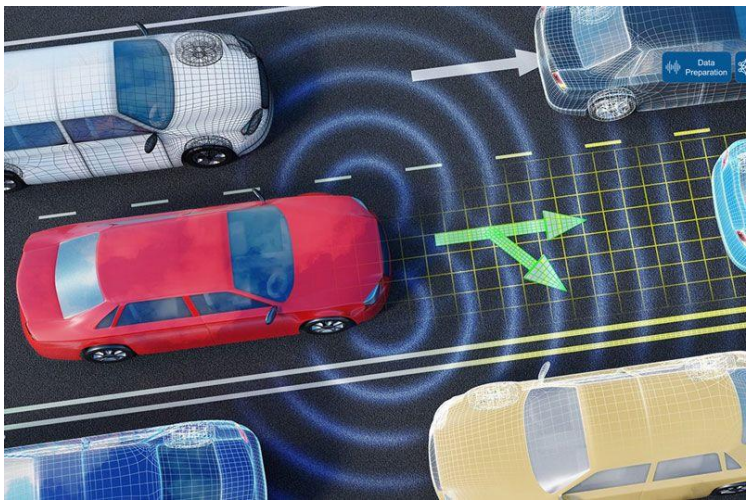
1.3.1.2. Mô hình hóa AI

Các yếu tố chính để thành công trong việc mô hình hóa các hệ thống AI là:

- Bắt đầu với một bộ hoàn chỉnh các thuật toán và mô hình dựng sẵn dành cho học máy, học sâu, học tăng cường và các kỹ thuật AI khác;
- Sử dụng các ứng dụng để thiết kế và phân tích hiệu quả;
- Làm việc trong một hệ sinh thái mở nơi các công cụ AI như MATLAB®, PyTorch và TensorFlow™ có thể được sử dụng cùng nhau;
- Quản lý độ phức tạp của điện toán với khả năng tăng tốc GPU và mở rộng quy mô cho các máy chủ song song và máy chủ đám mây cũng như trung tâm dữ liệu tại chỗ.

1.3.1.3. Thiết kế hệ thống

Các mô hình AI tồn tại trong một hệ thống hoàn chỉnh. Ví dụ, trong các hệ thống lái xe tự động, AI cho nhận thức phải tích hợp với các thuật toán để bản địa hóa, lập kế hoạch đường đi và kiểm soát phanh, tăng tốc và rẽ. Các hệ thống phức tạp, do AI điều khiển như thế này yêu cầu tích hợp và mô phỏng.



Hình 1.2. AI được sử dụng trong lái xe tự động

1.3.1.4. Triển khai mô hình AI

Các mô hình AI cần được triển khai cho CPU hoặc GPU, cho dù là một phần của thiết bị nhúng hoặc thiết bị biên, hệ thống doanh nghiệp hay đám mây. Các mô hình AI chạy trên thiết bị nhúng hoặc thiết bị biên cung cấp kết quả nhanh chóng cần thiết tại hiện trường, trong khi các mô hình AI chạy trong hệ thống doanh nghiệp và đám mây cung cấp kết quả từ dữ liệu được thu thập trên nhiều thiết bị. Thông thường, các mô hình AI được triển khai để kết hợp các hệ thống này.

Quá trình triển khai được tăng tốc khi tạo mã từ các mô hình và nhắm mục tiêu các thiết bị. Bằng cách sử dụng các kỹ thuật tối ưu hóa tạo mã và thư viện được tối ưu hóa cho phần cứng, mã có thể được điều chỉnh để phù hợp với cấu hình năng lượng thấp theo yêu cầu của thiết bị nhúng và thiết bị biên hoặc nhu cầu hiệu suất cao của hệ thống doanh nghiệp và đám mây.

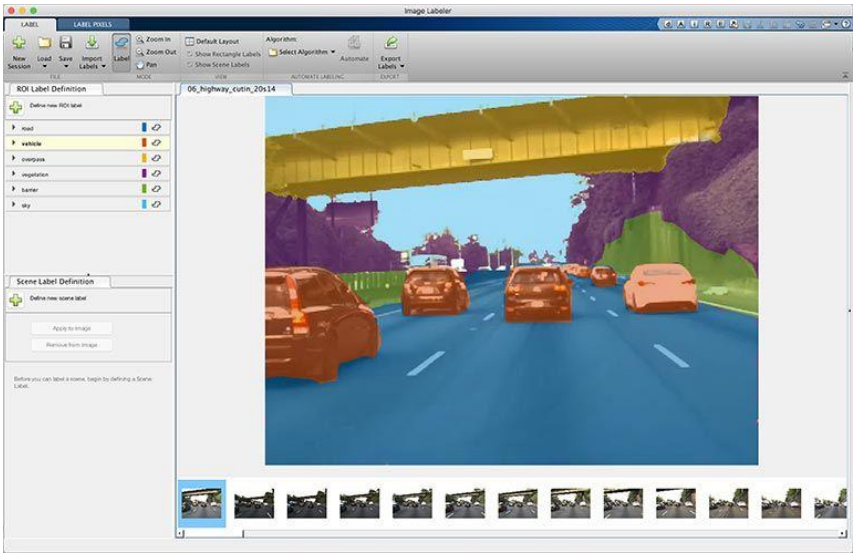
1.3.2. Phát triển các hệ thống AI với MATLAB

Có một sự thiếu hụt rõ ràng về các kỹ năng trong AI. Tuy nhiên, các kỹ sư và nhà khoa học sử dụng MATLAB hoặc Simulink® có các kỹ năng và công cụ cần thiết để tạo ra các hệ thống do AI điều khiển trong lĩnh vực chuyên môn của họ.

1.3.2.1. Tiền xử lý dữ liệu với MATLAB

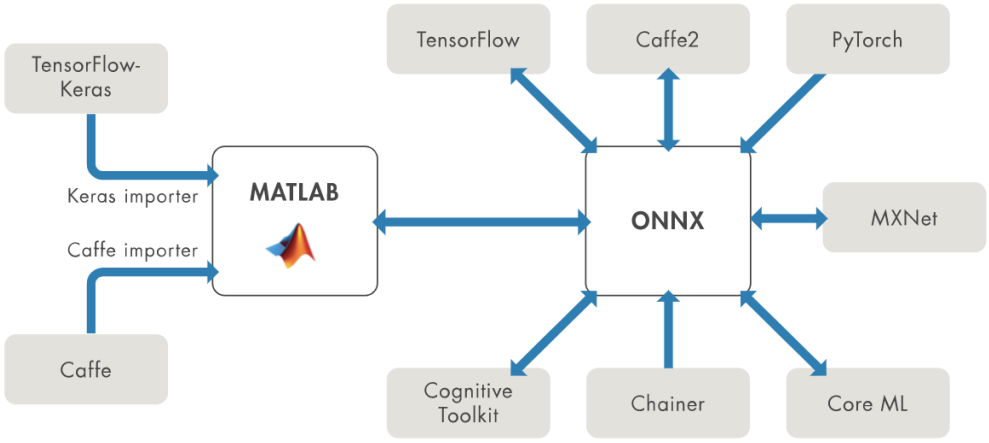
Tiền xử lý dữ liệu sẽ tiết kiệm được thời gian hoạt động của mô hình. Từ dữ liệu cảm biến chuỗi thời gian đến hình ảnh đến văn bản, các ứng dụng và kiểu dữ liệu MATLAB giảm đáng kể thời gian cần thiết để xử lý trước dữ liệu. Các chức năng cấp cao giúp dễ dàng đồng bộ hóa các chuỗi thời gian khác nhau, thay thế các giá trị ngoại lai bằng các giá trị được nội suy, lọc các tín hiệu nhiễu, tách văn bản thô thành các từ, v.v. Dữ liệu có thể được trực quan hóa nhanh chóng để hiểu xu hướng và xác định các vấn đề về chất lượng dữ liệu với các biểu đồ và Trình chỉnh sửa trực tiếp.

Các ứng dụng MATLAB tự động ghi nhãn sự thật cho dữ liệu hình ảnh, video và âm thanh.



Hình 1.3. Sử dụng các ứng dụng ghi nhãn cho quy trình học sâu như phân đoạn ngữ nghĩa

Để kiểm tra các thuật toán trước khi có dữ liệu từ cảm biến hoặc thiết bị khác, có thể tạo dữ liệu tổng hợp từ Simulink. Cách tiếp cận này thường được sử dụng trong các hệ thống lái xe tự động như kiểm soát hành trình thích ứng, hỗ trợ giữ làn đường và phanh khẩn cấp tự động.



Hình 1.4. Tương tác với các framework học sâu

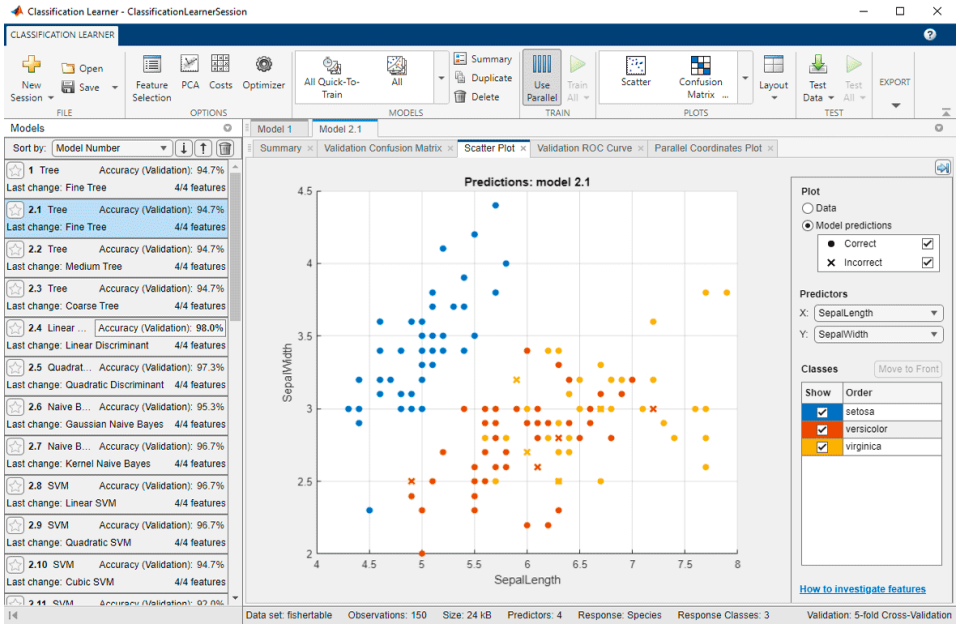
1.3.2.2. Lập mô hình AI với MATLAB

a) Học máy (Machine learning)

Người dùng MATLAB đã triển khai hàng nghìn ứng dụng để bảo trì dự đoán, phân tích cảm biến, tài chính và điện tử truyền thông. Hộp công cụ học máy và thống kê™ làm cho các phần khó của học máy trở nên dễ dàng với các ứng dụng để đào tạo và so sánh các mô hình, xử lý tín hiệu nâng cao và trích xuất tính năng, phân loại, hồi quy và thuật toán phân cụm cho học tập có giám sát và không giám sát.

ASML, một nhà sản xuất chất bán dẫn, đã sử dụng các kỹ thuật máy học để tạo ra công nghệ đo lường ảo nhằm cải thiện sự liên kết lớp phủ trong các cấu trúc phức tạp tạo nên một con chip. “Là một kỹ sư quy trình, tôi không có kinh nghiệm về mạng lưới thần kinh hoặc học máy. Tôi đã nghiên cứu các ví dụ MATLAB để tìm các chức năng máy học tốt nhất để tạo phép đo ảo. Kỹ sư Emil Schmitt-Weaver giải thích rằng tôi không thể làm điều này bằng C hoặc Python—sẽ mất quá nhiều thời gian để tìm, xác thực và tích hợp các gói phù hợp.

Các mô hình MATLAB cũng có khả năng thực thi nhanh hơn nguồn mở trên hầu hết các tính toán thống kê và máy học.

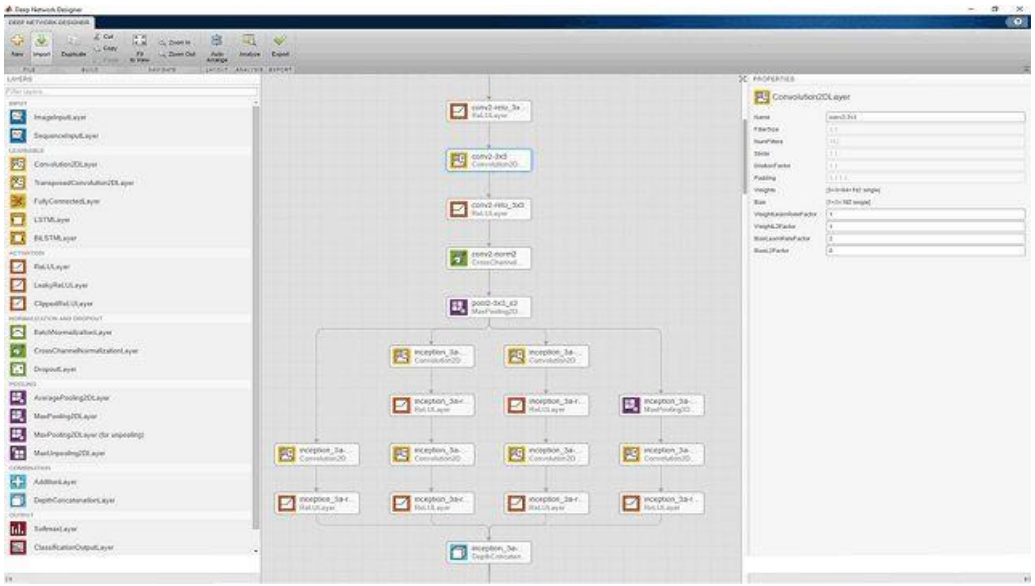


Hình 1.5. Ứng dụng Trình học phân loại, cho phép bạn thử các trình phân loại khác nhau và tìm ra cách phù hợp nhất cho tập dữ liệu của mình.

b) Học sâu (Deep learning)

Các kỹ sư sử dụng khả năng học sâu của MATLAB để lái xe tự động, thị giác máy tính, xử lý giọng nói và ngôn ngữ tự nhiên cũng như các ứng dụng khác. Deep Learning Toolbox™ cho phép người dung tạo, kết nối, đào tạo và đánh giá các lớp của mạng lưới thần kinh sâu. Các ví dụ và mạng được đào tạo trước giúp dễ dàng sử dụng MATLAB để học sâu, ngay cả khi không có kiến thức về các thuật toán thị giác máy tính nâng cao hoặc mạng thần kinh.

MATLAB cho phép các kỹ sư làm việc cùng nhau trên các khung học sâu khác nhau. Với sự hỗ trợ cho ONNX, MATLAB cho phép nhập và xuất các mô hình mới nhất sang và từ các khung được hỗ trợ khác, bao gồm cả TensorFlow.



Hình 1.6. Ứng dụng Deep Network Designer cho phép người dùng xây dựng, trực quan hóa và chỉnh sửa các mạng học sâu.

c) Học tăng cường (Reinforcement Learning)

Trong các hệ thống kiểm soát được hưởng lợi từ việc học dựa trên phần thưởng tích lũy, học tăng cường là một kỹ thuật lý tưởng. Reinforcement Learning Toolbox™ cho phép người dùng đào tạo các chính sách bằng DQN, A2C, DDPG và các thuật toán học tăng cường khác. Bạn có thể sử dụng các chính sách này để triển khai bộ điều khiển và thuật toán ra quyết định cho các hệ thống phức tạp như rô bốt và hệ thống tự trị. Bạn có thể triển khai các chính sách bằng cách sử dụng mạng nơ-ron sâu, đa thức hoặc bảng tra cứu.

CHƯƠNG 2

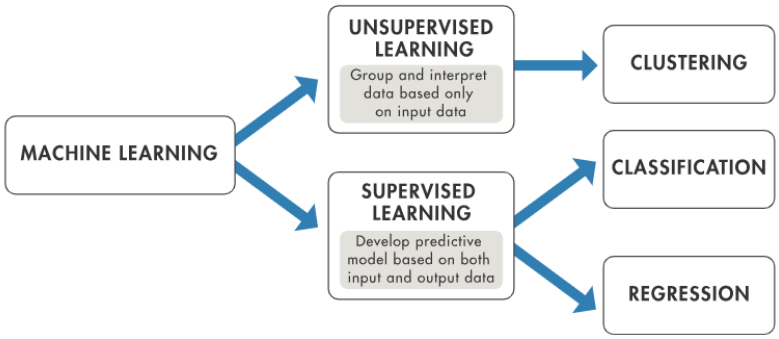
CÁC THUẬT TOÁN TRÍ TUỆ NHÂN TẠO

2.1. Học máy (Machine learning ML)

Machine Learning là một kỹ thuật AI dạy máy tính học hỏi kinh nghiệm. Các thuật toán học máy sử dụng các phương pháp tính toán để "học" thông tin trực tiếp từ dữ liệu mà không dựa vào một phương trình định sẵn làm mô hình. Các thuật toán cải thiện hiệu suất của chúng một cách thích ứng khi số lượng mẫu có sẵn để học tăng lên. Học sâu là một hình thức chuyên biệt của học máy.

2.1.1. Cách thức hoạt động của ML

Học máy sử dụng hai loại kỹ thuật: học có giám sát (supervised learning), đào tạo một mô hình trên dữ liệu đầu vào và đầu ra đã biết để có thể dự đoán kết quả đầu ra trong tương lai và học không giám sát (unsupervised learning), tìm các mẫu ẩn hoặc cấu trúc nội tại trong dữ liệu đầu vào.



Hình 2.1. Các kỹ thuật học máy bao gồm cả học không giám sát và học có giám sát

2.1.1.1. Học có giám sát

Học máy có giám sát xây dựng một mô hình đưa ra dự đoán dựa trên bằng chứng khi có sự không chắc chắn. Thuật toán học có giám sát lấy một tập hợp dữ liệu đầu vào đã biết và các phản hồi đã biết đối với dữ liệu (đầu ra) rồi huấn luyện một mô hình để tạo ra các dự đoán hợp lý cho phản hồi đối với dữ liệu mới. Sử dụng học có giám sát nếu đã biết dữ liệu cho đầu ra mà bạn đang cố gắng dự đoán.

Học có giám sát sử dụng các kỹ thuật phân loại và hồi quy để phát triển các mô hình học máy. Các kỹ thuật phân loại dự đoán các phản hồi rời rạc. Ví dụ: email là thật hay thư rác, hoặc khối u là ung thư hay lành tính. Các mô hình phân loại phân loại dữ liệu đầu vào thành các danh mục. Các ứng dụng điển hình bao gồm hình ảnh y tế, nhận dạng giọng nói và chấm điểm tín dụng.

Sử dụng phân loại nếu dữ liệu của bạn có thể được gắn thẻ, phân loại hoặc tách thành các nhóm hoặc lớp cụ thể. Ví dụ: các ứng dụng nhận dạng chữ viết tay sử dụng phân loại để nhận dạng các chữ cái và số. Trong xử lý hình ảnh và thị giác máy tính, các kỹ thuật nhận dạng mẫu không giám sát được sử dụng để phát hiện đối tượng và phân đoạn hình ảnh. Các thuật toán phổ biến nhất để thực hiện phân loại có thể được tìm thấy ở đây.

Các kỹ thuật hồi quy dự đoán các phản ứng liên tục. Ví dụ: các đại lượng vật lý khó đo lường như trạng thái sạc của pin, tải điện trên lưới hoặc giá của các tài sản tài chính. Các ứng dụng điển hình bao gồm cảm biến ảo, dự báo phụ tải điện và giao dịch theo thuật toán.

Sử dụng các kỹ thuật hồi quy nếu bạn đang làm việc với một phạm vi dữ liệu hoặc nếu bản chất phản hồi của bạn là một số thực, chẳng hạn như nhiệt độ hoặc thời gian cho đến khi một thiết bị gặp sự cố.

2.1.1.2. Học không giám sát

Học không giám sát tìm các mẫu ẩn hoặc cấu trúc nội tại trong dữ liệu. Nó được sử dụng để rút ra các suy luận từ các bộ dữ liệu bao gồm dữ liệu đầu vào không có phản hồi được gắn nhãn.

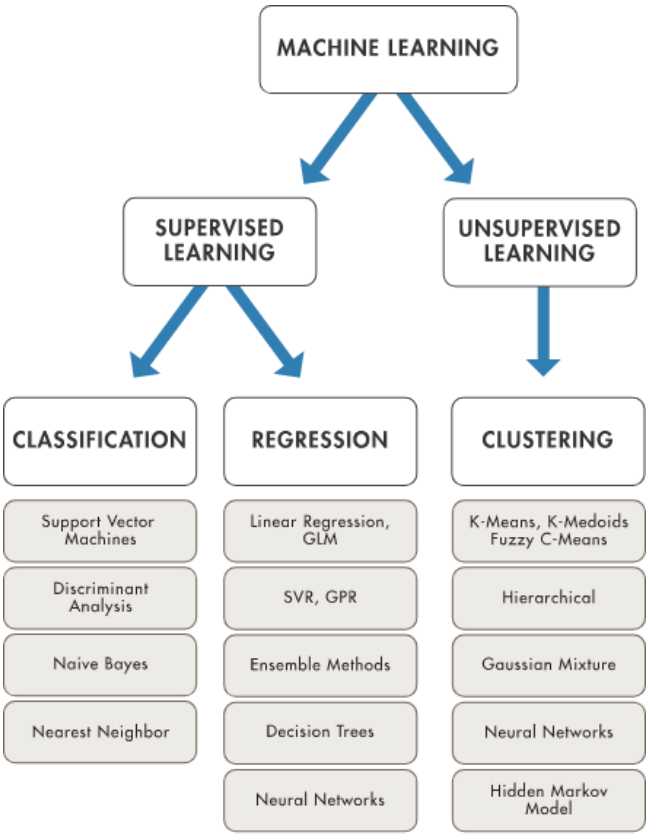
Phân cụm là kỹ thuật học tập không giám sát phổ biến nhất. Nó được sử dụng để phân tích dữ liệu khám phá để tìm các mẫu hoặc nhóm ẩn trong dữ liệu. Các ứng dụng để phân tích cụm bao gồm phân tích trình tự gen, nghiên cứu thị trường và nhận dạng đối tượng. Ví dụ: nếu một công ty điện thoại di động muốn tối ưu hóa các vị trí nơi họ xây dựng tháp điện thoại di động, họ có thể sử dụng máy học để ước tính số lượng cụm người dựa vào tháp của họ. Điện thoại chỉ có thể nói chuyện với một tháp tại một thời điểm, vì vậy nhóm sử dụng thuật toán phân cụm để thiết kế vị trí tốt nhất của các tháp di động nhằm tối ưu hóa khả năng nhận tín hiệu cho các nhóm hoặc cụm khách hàng của họ.

Hình 2.2. Phân cụm tìm các mẫu ẩn trong dữ liệu

2.1.1.3. Quyết định sử dụng thuật toán học máy

Việc chọn thuật toán phù hợp có vẻ là một vấn đề khó, có hàng tá thuật toán học máy được giám sát và không được giám sát, và mỗi thuật toán có một cách tiếp cận khác nhau để học.

Không có phương pháp tốt nhất hoặc một kích thước phù hợp với tất cả. Việc tìm ra thuật toán phù hợp một phần chỉ là thử và sai. Ngay cả các nhà khoa học dữ liệu có kinh nghiệm cao cũng không thể biết liệu một thuật toán có hoạt động hay không nếu không dùng thử. Tuy nhiên, việc lựa chọn thuật toán cũng phụ thuộc vào kích thước và loại dữ liệu bạn đang làm việc, thông tin chi tiết bạn muốn nhận được từ dữ liệu và cách sử dụng những thông tin chi tiết đó.



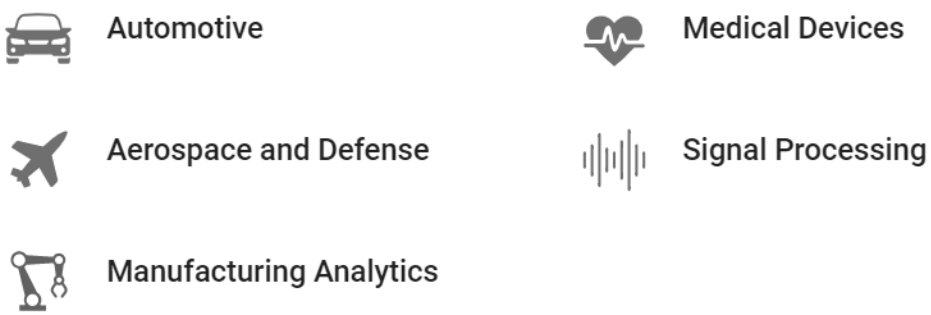
Hình 2.3. Các kỹ thuật học máy

Dưới đây là một số hướng dẫn về việc lựa chọn giữa học máy có giám sát và không giám sát:

- Chọn học có giám sát nếu bạn cần đào tạo một mô hình để đưa ra dự đoán, ví dụ: giá trị tương lai của một biến liên tục, chẳng hạn như nhiệt độ hoặc giá cổ phiếu hoặc phân loại, chẳng hạn như xác định các nhà sản xuất ô tô từ cảnh quay video trên webcam.
- Chọn học không giám sát nếu bạn cần khám phá dữ liệu của mình và muốn đào tạo một mô hình để tìm ra một biểu diễn bên trong tốt, chẳng hạn như chia dữ liệu thành các cụm.

2.1.2. Tầm quan trọng của ML

Với sự gia tăng của dữ liệu lớn, học máy đã trở thành một kỹ thuật chính để giải quyết các vấn đề trong nhiều lĩnh vực.



Hình 2.4. Một số lĩnh vực ứng dụng ML

a) Nhiều dữ liệu hơn, nhiều câu hỏi hơn, câu trả lời tốt hơn

Các thuật toán máy học tìm các mẫu tự nhiên trong dữ liệu để tạo ra thông tin chi tiết và giúp đưa ra các quyết định cũng như dự đoán tốt hơn. Chúng được sử dụng hàng ngày để đưa ra các quyết định quan trọng trong chẩn đoán y tế, giao dịch chứng khoán, dự báo tải năng lượng, v.v. Ví dụ: các trang web truyền thông dựa vào công nghệ máy học để sàng lọc hàng triệu tùy chọn nhằm cung cấp cho bạn bài hát hoặc phim đề xuất. Các nhà bán lẻ sử dụng nó để hiểu rõ hơn về hành vi mua hàng của khách hàng.

b) Khi nào bạn sử dụng máy học?

Cân nhắc sử dụng máy học khi có một nhiệm vụ hoặc vấn đề phức tạp liên quan đến một lượng lớn dữ liệu và nhiều biến số, nhưng không có công thức hoặc phương trình hiện có.

2.2. Học sâu (Deep learning DL)

Học sâu là một kỹ thuật học máy dạy máy tính làm những gì tự nhiên đến với con người: học bằng ví dụ. Học sâu là một công nghệ quan trọng đằng sau những chiếc xe không người lái, cho phép chúng nhận ra biển báo dừng hoặc phân biệt người đi bộ với cột đèn. Đây là chìa khóa để điều khiển bằng giọng nói trong các thiết bị tiêu dùng như điện thoại, máy tính bảng, TV và loa rảnh tay. Học sâu đang nhận được rất nhiều sự chú ý gần đây và vì lý do chính đáng. Nó đang đạt được những kết quả mà trước đây không thể đạt được.

Trong học sâu, một mô hình máy tính học cách thực hiện các tác vụ phân loại trực tiếp từ hình ảnh, văn bản hoặc âm thanh. Các mô hình học sâu có thể đạt được độ chính xác cao nhất, đôi khi vượt quá hiệu suất của con người. Các mô hình được đào tạo bằng cách sử dụng một tập hợp lớn dữ liệu được gắn nhãn và kiến trúc mạng thần kinh chứa nhiều lớp.

2.2.1. Tầm quan trọng của DL

a) Làm thế nào để học sâu đạt được kết quả ấn tượng như vậy?

Học sâu đạt được độ chính xác nhận dạng ở mức cao. Điều này giúp thiết bị điện tử tiêu dùng đáp ứng mong đợi của người dùng và điều này rất quan trọng đối với các ứng dụng quan trọng về an toàn như ô tô không người lái. Những tiến bộ gần đây trong học sâu đã được cải thiện đến mức học sâu vượt trội hơn con người trong một số tác vụ như phân loại đối tượng trong hình ảnh.

Mặc dù học sâu lần đầu tiên được đưa ra lý thuyết vào những năm 1980, nhưng có hai lý do chính khiến nó chỉ mới trở nên hữu ích trong thời gian gần đây:

Học sâu yêu cầu một lượng lớn dữ liệu được dán nhãn. Ví dụ, việc phát triển xe không người lái đòi hỏi hàng triệu hình ảnh và hàng nghìn giờ video.

Học sâu đòi hỏi sức mạnh tính toán đáng kể. GPU hiệu suất cao có kiến trúc song song hiệu quả cho việc học sâu. Khi được kết hợp với các cụm hoặc điện toán đám mây, điều này cho phép các nhóm phát triển giảm thời gian đào tạo cho mạng học sâu từ hàng tuần xuống hàng giờ hoặc ít hơn.

Ví dụ về Deep Learning tại nơi làm việc:

- Các ứng dụng học sâu được sử dụng trong các ngành từ lái xe tự động đến các thiết bị y tế.
- Lái xe tự động: Các nhà nghiên cứu ô tô đang sử dụng học sâu để tự động phát hiện các đối tượng như biển báo dừng và đèn giao thông. Ngoài ra, học sâu được sử dụng để phát hiện người đi bộ, giúp giảm tai nạn.

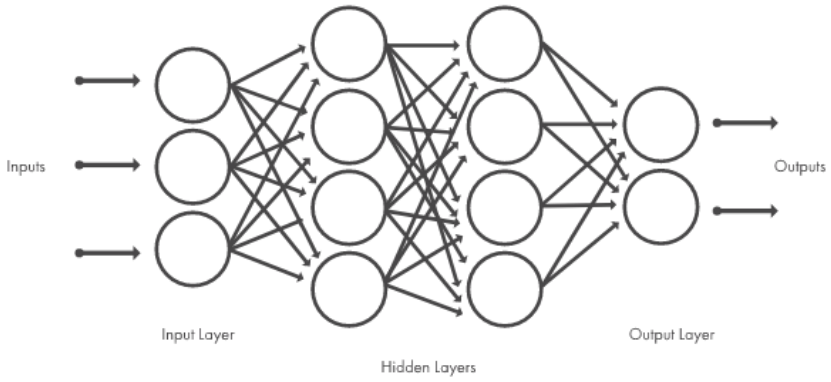
- Không gian vũ trụ và Quốc phòng: Học sâu được sử dụng để xác định các đối tượng từ các vệ tinh định vị các khu vực quan tâm và xác định các khu vực an toàn hoặc không an toàn cho quân đội.
- Nghiên cứu y học: Các nhà nghiên cứu ung thư đang sử dụng học sâu để tự động phát hiện các tế bào ung thư. Các nhóm tại UCLA đã chế tạo một kính hiển vi tiên tiến mang lại bộ dữ liệu nhiều chiều được sử dụng để đào tạo một ứng dụng học sâu nhằm xác định chính xác các tế bào ung thư.
- Tự động hóa công nghiệp: Học sâu đang giúp cải thiện sự an toàn của công nhân khi làm việc với máy móc hạng nặng bằng cách tự động phát hiện khi có người hoặc vật thể ở trong khoảng cách không an toàn với máy móc.
- Điện tử: Học sâu đang được sử dụng trong dịch thuật nghe và nói tự động. Ví dụ: các thiết bị hỗ trợ tại nhà phản hồi giọng nói của bạn và biết các tùy chọn của bạn được cung cấp bởi các ứng dụng học sâu.

2.2.2. Cách thức hoạt động của DL

Hầu hết các phương pháp học sâu đều sử dụng kiến trúc mạng thần kinh, đó là lý do tại sao các mô hình học sâu thường được gọi là mạng lưới thần kinh sâu.

Thuật ngữ "sâu" thường đề cập đến số lớp ẩn trong mạng lưới thần kinh. Mạng thần kinh truyền thống chỉ chứa 2-3 lớp ẩn, trong khi mạng sâu có thể có tới 150.

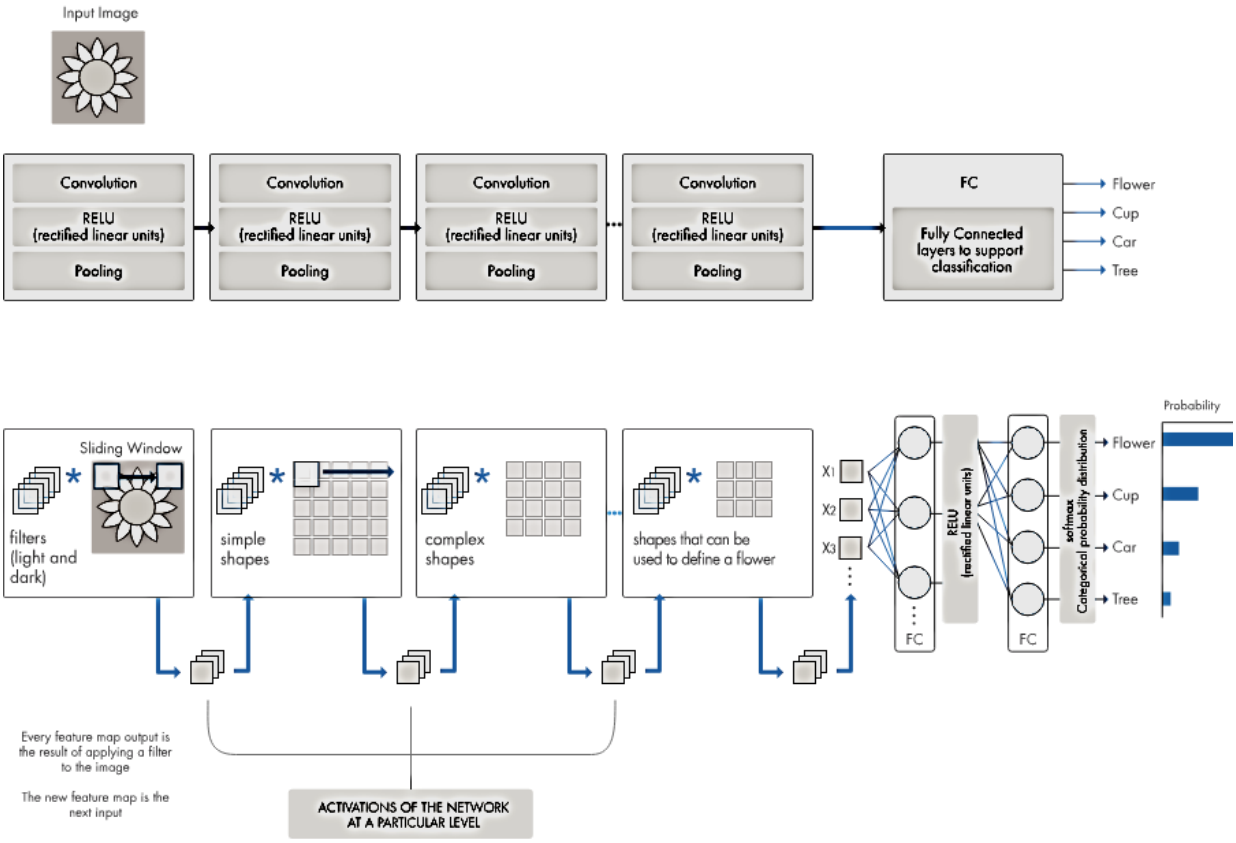
Các mô hình học sâu được đào tạo bằng cách sử dụng tập hợp lớn dữ liệu được gắn nhãn và kiến trúc mạng nơ-ron học các tính năng trực tiếp từ dữ liệu mà không cần trích xuất tính năng thủ công.



Hình 2.5. Mạng lưới thần kinh, được tổ chức theo lớp bao gồm một tập hợp các nút được kết nối với nhau. Mạng có thể có hàng chục hoặc hàng trăm lớp ẩn.

Một trong những loại mạng nơ-ron sâu phổ biến nhất được gọi là mạng nơ-ron tích chập (CNN hoặc ConvNet). CNN tích hợp các tính năng đã học với dữ liệu đầu vào và sử dụng các lớp tích chập 2D, làm cho kiến trúc này rất phù hợp để xử lý dữ liệu 2D, chẳng hạn như hình ảnh.

CNN loại bỏ nhu cầu trích xuất tính năng thủ công, do đó bạn không cần xác định các tính năng được sử dụng để phân loại hình ảnh. CNN hoạt động bằng cách trích xuất các tính năng trực tiếp từ hình ảnh. Các tính năng liên quan không được đào tạo trước; chúng được học trong khi mạng huấn luyện trên một tập hợp các hình ảnh. Tính năng trích xuất tính năng tự động này làm cho các mô hình học sâu có độ chính xác cao đối với các tác vụ thị giác máy tính, chẳng hạn như phân loại đối tượng.



Hình 2.6. Ví dụ về một mạng có nhiều lớp tích chập. Các bộ lọc được áp dụng cho từng hình ảnh đào tạo ở các độ phân giải khác nhau và đầu ra của từng hình ảnh được tích hợp đóng vai trò là đầu vào cho lớp tiếp theo.

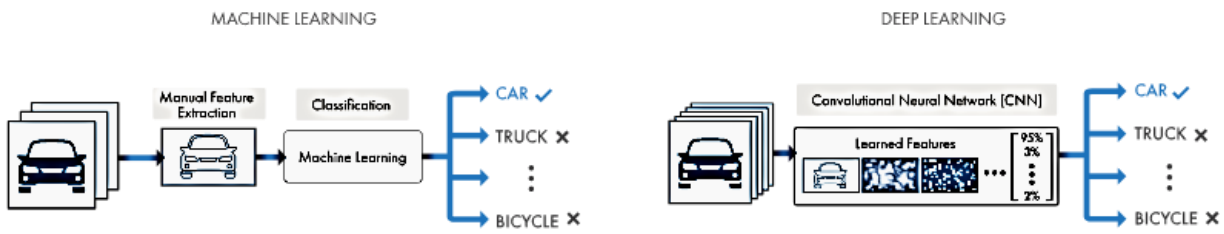
CNN học cách phát hiện các đặc điểm khác nhau của hình ảnh bằng cách sử dụng hàng chục hoặc hàng trăm lớp ẩn. Mỗi lớp ẩn làm tăng độ phức tạp của các tính năng hình ảnh đã học. Ví dụ: lớp ẩn đầu tiên có thể học cách phát hiện các cạnh và lớp cuối cùng học cách phát hiện các hình dạng phức tạp hơn phục vụ riêng cho hình dạng của đối tượng mà chúng ta đang cố gắng nhận dạng.

2.2.3. Sự khác nhau giữa ML và DL

Học sâu là một hình thức chuyên biệt của học máy. Quy trình làm việc của máy học bắt đầu với các tính năng liên quan được trích xuất thủ công từ hình ảnh. Sau đó, các tính năng này được sử dụng để tạo mô hình phân loại các đối tượng trong ảnh. Với quy trình học sâu, các tính năng liên quan sẽ tự động được trích xuất từ hình ảnh. Ngoài ra, học sâu thực hiện "học từ đầu đến cuối" – trong đó mạng được cung cấp dữ liệu thô và một nhiệm vụ cần thực hiện, chẳng hạn như phân loại và mạng học cách thực hiện việc này một cách tự động.

Một điểm khác biệt chính nữa là thuật toán học sâu mở rộng theo dữ liệu, trong khi học nông hội tụ. Học nông đề cập đến các phương pháp học máy ổn định ở một mức hiệu suất nhất định khi bạn thêm nhiều ví dụ và dữ liệu huấn luyện vào mạng.

Một lợi thế chính của các mạng học sâu là chúng thường tiếp tục cải thiện khi kích thước dữ liệu của bạn tăng lên.



Hình 2.7. So sánh phương pháp học máy để phân loại phương tiện (trái) với học sâu (phải).

Trong ML, các tính năng và bộ phân loại được chọn thủ công để sắp xếp hình ảnh. Với DL, các bước trích xuất tính năng và lập mô hình là tự động.

2.2.4. Lựa chọn giữa ML và DL

ML cung cấp nhiều kỹ thuật và mô hình mà bạn có thể chọn dựa trên ứng dụng của mình, kích thước dữ liệu bạn đang xử lý và loại vấn đề bạn muốn giải quyết. Một ứng dụng DL thành công yêu cầu một lượng dữ liệu rất lớn (hàng nghìn hình ảnh) để đào tạo mô hình, cũng như GPU hoặc bộ xử lý đồ họa để xử lý dữ liệu của bạn một cách nhanh chóng.

Khi lựa chọn giữa ML và DL, hãy cân nhắc xem GPU hiệu suất cao và nhiều dữ liệu được gắn nhãn hay không. Nếu không có một trong hai thứ đó, thì việc sử dụng máy học thay vì học sâu có thể hợp lý hơn. Học sâu thường phức tạp hơn, vì vậy sẽ cần ít nhất vài nghìn hình ảnh để có được kết quả đáng tin cậy. Có GPU hiệu suất cao có nghĩa là mô hình sẽ mất ít thời gian hơn để phân tích tất cả những hình ảnh đó.

2.3. Học tăng cường (Reinforcement learning RL)

Học tăng cường là một loại kỹ thuật máy học trong đó tác nhân máy tính học cách thực hiện một tác vụ thông qua các tương tác thử và sai lặp đi lặp lại với môi trường động. Phương pháp học tập này cho phép tác nhân đưa ra một loạt quyết định nhằm tối đa hóa chỉ số phần thưởng cho nhiệm vụ mà không cần sự can thiệp của con người và không được lập trình rõ ràng để đạt được nhiệm vụ.

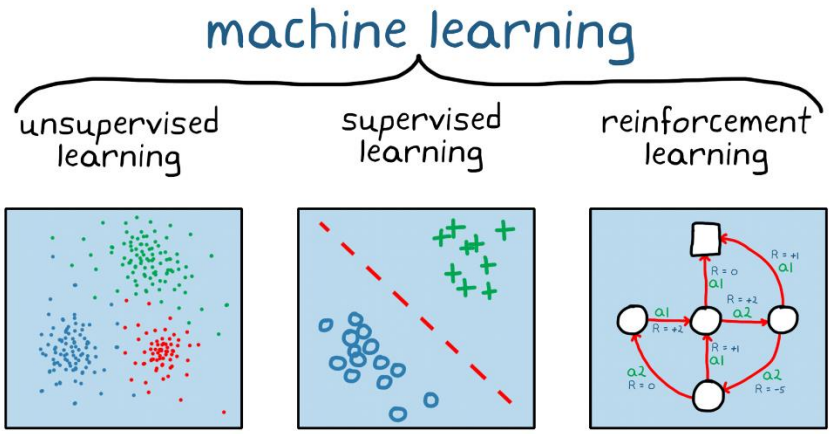
Các chương trình AI được đào tạo bằng phương pháp học tăng cường đã đánh bại người chơi trong các trò chơi cờ như cờ vây và cờ vua, cũng như các trò chơi điện tử. Mặc dù học tăng cường hoàn toàn không phải là một khái niệm mới, nhưng những tiến bộ gần đây về khả năng học sâu và khả năng tính toán đã giúp đạt được một số kết quả đáng chú ý trong lĩnh vực trí tuệ nhân tạo.

2.3.1. Tầm quan trọng của học tăng cường

2.3.1.1. Học tăng cường so với Học máy so với Học sâu

Học tăng cường là một nhánh của học máy (Hình 2.8). Không giống như học máy có giám sát và không giám sát, học tăng cường không dựa vào tập dữ liệu tĩnh mà hoạt động trong môi trường động và học hỏi từ kinh nghiệm thu thập được. Điểm dữ liệu hoặc trải nghiệm được thu thập trong quá trình đào tạo thông qua các tương tác thử và sai giữa môi trường và tác nhân phần mềm. Khía cạnh này của học tăng cường rất quan trọng, vì nó giảm bớt nhu cầu thu thập dữ liệu, tiền xử lý và ghi nhãn trước khi đào tạo, nếu không thì cần thiết trong học có giám sát và không giám sát. Trên thực tế, điều này có nghĩa là, nếu được khuyến khích phù hợp, một mô hình học tăng cường có thể bắt đầu tự học một hành vi mà không cần sự giám sát (của con người).

Học sâu kéo dài cả ba loại máy học; học tăng cường và học sâu không loại trừ lẫn nhau. Các vấn đề học tăng cường phức tạp thường dựa vào mạng lưới thần kinh sâu, một lĩnh vực được gọi là học tăng cường sâu.



Hình 2.8. Ba loại học máy chính: học không giám sát, học có giám sát và học tăng cường

2.3.1.2. Các ứng dụng của học tăng cường

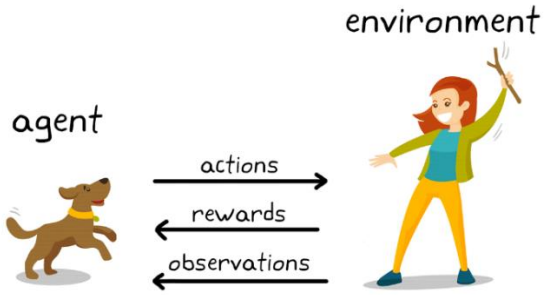
Mạng lưới thần kinh sâu được đào tạo với học tăng cường có thể mã hóa các hành vi phức tạp. Điều này cho phép một cách tiếp cận thay thế đối với các ứng dụng khó xử lý hoặc khó giải quyết hơn bằng các phương pháp truyền thống hơn. Ví dụ, trong lái xe tự động, một mạng lưới thần kinh có thể thay thế người lái xe và quyết định cách xoay vô lăng bằng cách xem xét đồng thời nhiều cảm biến như khung camera và phép đo nắp. Nếu không có mạng nơ-ron, vấn đề thường sẽ được chia thành các phần nhỏ hơn như trích xuất các tính năng từ khung máy ảnh, lọc các phép đo lidar, hợp nhất các đầu ra của cảm biến và đưa ra quyết định "lái xe" dựa trên đầu vào của cảm biến.

Trong khi phương pháp học tăng cường vẫn đang được đánh giá cho các hệ thống sản xuất, một số ứng dụng công nghiệp là những ứng cử viên sáng giá cho công nghệ này.

- Điều khiển nâng cao: Điều khiển các hệ thống phi tuyến tính là một bài toán thách thức thường được giải quyết bằng cách tuyến tính hóa hệ thống tại các điểm vận hành khác nhau. Học tăng cường có thể được áp dụng trực tiếp cho hệ thống phi tuyến tính.
- Lái xe tự động: Đưa ra quyết định lái xe dựa trên thông tin đầu vào của camera là một lĩnh vực mà học tăng cường phù hợp khi xem xét sự thành công của mạng lưới thần kinh sâu trong các ứng dụng hình ảnh.
- Robotics: Học tăng cường có thể trợ giúp với các ứng dụng như rô bốt cầm nắm, chẳng hạn như dạy một cánh tay rô bốt cách thao tác với nhiều đối tượng khác nhau cho các ứng dụng gắp và đặt. Các ứng dụng rô-bốt khác bao gồm sự hợp tác giữa người-rô-bốt và rô-bốt-rô-bốt.
- Lập kế hoạch: Các vấn đề về lập lịch trình xuất hiện trong nhiều tình huống bao gồm điều khiển đèn giao thông và điều phối các nguồn lực trên sàn nhà máy hướng tới một số mục tiêu. Học tăng cường là một giải pháp thay thế tốt cho các phương pháp tiến hóa để giải các bài toán tối ưu tổ hợp này.
- Hiệu chuẩn: Các ứng dụng liên quan đến hiệu chuẩn thủ công các thông số, chẳng hạn như hiệu chuẩn bộ điều khiển điện tử (ECU), có thể là ứng cử viên tốt cho việc học tăng cường.

2.3.2. Cách thức hoạt động của học tăng cường

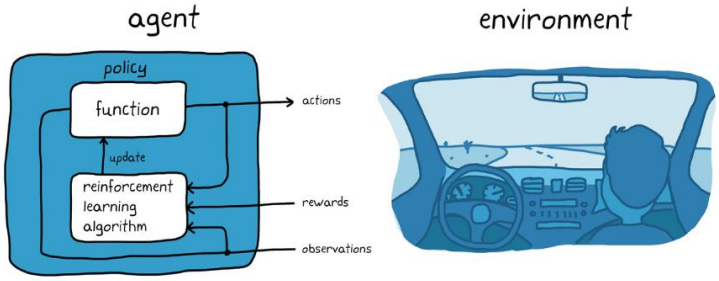
Cơ chế đào tạo đằng sau việc học tăng cường phản ánh nhiều kịch bản trong thế giới thực. Ví dụ, hãy xem xét việc huấn luyện thú cưng thông qua củng cố tích cực.



Hình 2.9. Học tăng cường trong huấn luyện chó

Sử dụng thuật ngữ học tăng cường (Hình 2.9), mục tiêu của việc học trong trường hợp này là huấn luyện chó (tác nhân) hoàn thành nhiệm vụ trong một môi trường, bao gồm môi trường xung quanh chó cũng như người huấn luyện. Đầu tiên, người huấn luyện đưa ra mệnh lệnh hoặc gợi ý để chó tuân theo (quan sát). Con chó sau đó phản ứng bằng cách thực hiện một hành động. Nếu hành động gần với hành vi mong muốn, người huấn luyện có thể sẽ đưa ra phần thưởng, chẳng hạn như đồ ăn hoặc đồ chơi; nếu không, sẽ không có phần thưởng nào được cung cấp. Khi bắt đầu huấn luyện, con chó có thể sẽ thực hiện nhiều hành động ngẫu nhiên hơn như lăn lộn khi lệnh được đưa ra là "ngồi xuống", vì nó đang cố gắng liên kết các quan sát cụ thể với các hành động và phần thưởng. Sự liên kết này, hoặc ánh xạ, giữa các quan sát và hành động được gọi là chính sách. Từ quan điểm của con chó, trường hợp lý tưởng sẽ là trường hợp nó sẽ phản ứng chính xác với mọi gợi ý để nó nhận được càng nhiều phần thưởng càng tốt. Vì vậy, toàn bộ ý nghĩa của việc huấn luyện học tăng cường là "điều chỉnh" chính sách của con chó để nó học các hành vi mong muốn sẽ tối đa hóa một số phần thưởng. Sau khi huấn luyện xong, con chó sẽ có thể quan sát chủ và thực hiện hành động thích hợp, chẳng hạn như ngồi khi được lệnh "ngồi" bằng cách sử dụng chính sách nội bộ mà nó đã phát triển. Đến thời điểm này, các món ăn được hoan nghênh nhưng về mặt lý thuyết, không cần thiết.

Ghi nhớ ví dụ huấn luyện chó, hãy xem xét nhiệm vụ đỗ xe bằng hệ thống lái tự động (Hình 2.10). Mục tiêu là dạy cho máy tính của phương tiện (tác nhân) đỗ xe đúng chỗ bằng cách học tăng cường. Như trong trường hợp huấn luyện chó, môi trường là mọi thứ bên ngoài tác nhân và có thể bao gồm động lực học của phương tiện, các phương tiện khác có thể ở gần, điều kiện thời tiết, v.v. Trong quá trình huấn luyện, tác nhân sử dụng thông tin đọc từ các cảm biến như máy ảnh, GPS và lidar (quan sát) để tạo các lệnh (hành động) lái, phanh và tăng tốc. Để tìm hiểu cách tạo ra các hành động chính xác từ các quan sát (điều chỉnh chính sách), nhân viên liên tục cố gắng đỗ xe bằng quy trình thử và sai. Một tín hiệu phần thưởng có thể được cung cấp để đánh giá mức độ tốt của một thử nghiệm và để hướng dẫn quá trình học tập.



Hình 2.10. Học tăng cường trong bãi đậu xe tự động

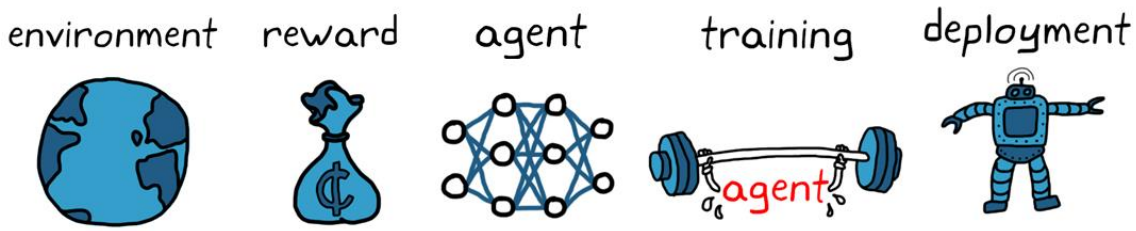
Trong ví dụ huấn luyện chó, quá trình huấn luyện diễn ra bên trong não của chó. Trong ví dụ về bãi đậu xe tự động, quá trình đào tạo được xử lý bằng thuật toán đào tạo. Thuật toán đào tạo

chịu trách nhiệm điều chỉnh chính sách của tác nhân dựa trên các bài đọc, hành động và phần thưởng của cảm biến được thu thập. Sau khi quá trình đào tạo hoàn tất, máy tính của xe sẽ có thể đỗ xe chỉ bằng cách sử dụng chính sách đã điều chỉnh và chỉ số cảm biến.

Một điều cần lưu ý là học tăng cường không hiệu quả theo mẫu. Nghĩa là, nó đòi hỏi một số lượng lớn các tương tác giữa tác nhân và môi trường để thu thập dữ liệu cho việc huấn luyện. Ví dụ, AlphaGo, chương trình máy tính đầu tiên đánh bại một nhà vô địch thế giới trong trò chơi cờ vây, đã được huấn luyện không ngừng nghỉ trong khoảng thời gian vài ngày bằng cách chơi hàng triệu ván cờ, tích lũy kiến thức hàng nghìn năm của loài người. Ngay cả đối với các ứng dụng tương đối đơn giản, thời gian đào tạo có thể mất từ vài phút đến vài giờ hoặc vài ngày. Ngoài ra, việc thiết lập vấn đề một cách chính xác có thể là một thách thức vì có một danh sách các quyết định thiết kế cần được thực hiện, có thể yêu cầu một vài lần lặp lại để giải quyết đúng. Chúng bao gồm, ví dụ, chọn kiến trúc thích hợp cho mạng thần kinh, điều chỉnh siêu tham số và định hình tín hiệu phần thưởng.

2.3.2.1. Quy trình học tăng cường

Quy trình chung để đào tạo một tác nhân sử dụng học tăng cường bao gồm các bước sau (Hình 2.11):



Hình 2.11. Quy trình học tăng cường

1. Tạo môi trường

Trước tiên, bạn cần xác định môi trường trong đó tác nhân học tăng cường hoạt động, bao gồm giao diện giữa tác nhân và môi trường. Môi trường có thể là mô hình mô phỏng hoặc hệ thống vật lý thực, nhưng môi trường mô phỏng thường là bước khởi đầu tốt vì chúng an toàn hơn và cho phép thử nghiệm.

2. Xác định phần thưởng

Tiếp theo, chỉ định tín hiệu phần thưởng mà tổng đài viên sử dụng để đo lường hiệu suất của nó so với các mục tiêu của nhiệm vụ và cách tính toán tín hiệu này từ môi trường. Việc định hình phần thưởng có thể phức tạp và có thể cần lặp đi lặp lại một vài lần để làm cho đúng.

3. Tạo tác nhân

Sau đó, bạn tạo tác nhân, bao gồm chính sách và thuật toán đào tạo học tăng cường. Vì vậy, bạn cần phải:

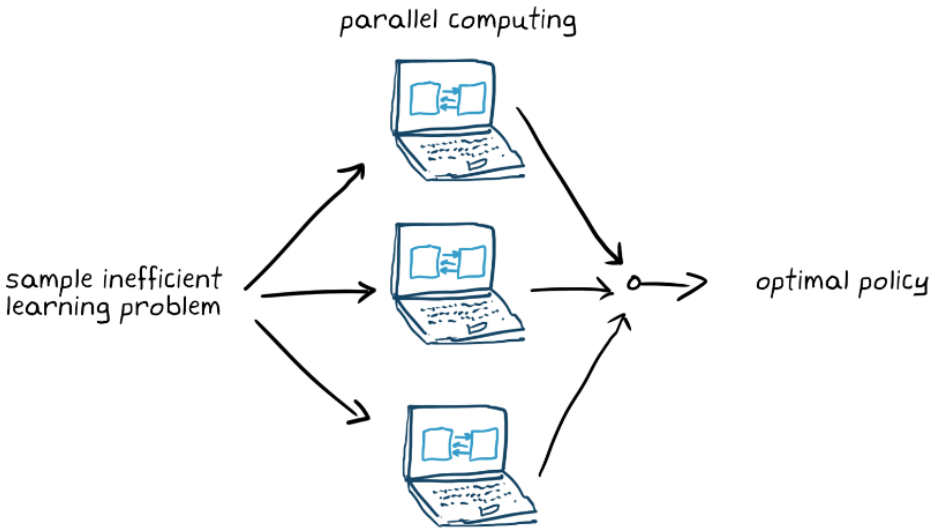
a) Chọn cách thể hiện chính sách (chẳng hạn như sử dụng mạng thần kinh hoặc bảng tra cứu).

b) Chọn thuật toán huấn luyện thích hợp. Các biểu diễn khác nhau thường được gắn với các loại thuật toán đào tạo cụ thể. Nhưng nói chung, hầu hết các thuật toán học tăng cường hiện

đại đều dựa vào mạng thần kinh vì chúng là những ứng cử viên sáng giá cho không gian trạng thái/hành động lớn và các vấn đề phức tạp.

4. Đào tạo và xác nhận tác nhân

Thiết lập các tùy chọn đào tạo (như tiêu chí dừng) và đào tạo tổng đài viên để điều chỉnh chính sách. Đảm bảo xác thực chính sách được đào tạo sau khi đào tạo kết thúc. Nếu cần, hãy xem lại các lựa chọn thiết kế như kiến trúc chính sách và tín hiệu phần thưởng và đào tạo lại. Học tăng cường thường được biết là không hiệu quả; đào tạo có thể mất từ vài phút đến vài ngày tùy thuộc vào ứng dụng. Đối với các ứng dụng phức tạp, đào tạo song song trên nhiều CPU, GPU và cụm máy tính sẽ tăng tốc mọi thứ (Hình 2.12).



Hình 2.12. Huấn luyện bài toán học không hiệu quả mẫu với tính toán song song

5. Triển khai chính sách

Triển khai biểu diễn chính sách được đào tạo bằng cách sử dụng, ví dụ: mã C/C++ hoặc CUDA được tạo. Tại thời điểm này, chính sách là một hệ thống ra quyết định độc lập.

Đào tạo một tác nhân sử dụng học tăng cường là một quá trình lặp đi lặp lại. Các quyết định và kết quả trong các giai đoạn sau có thể yêu cầu bạn quay lại giai đoạn trước trong quy trình học tập. Ví dụ: nếu quy trình đào tạo không hội tụ thành một chính sách tối ưu trong một khoảng thời gian hợp lý, bạn có thể phải cập nhật bất kỳ điều nào sau đây trước khi đào tạo lại nhân viên hỗ trợ:

+ Cài đặt đào tạo

+ Cấu hình thuật toán học tăng cường

+ Đại diện chính sách

+ Định nghĩa tín hiệu phần thưởng

+ Tín hiệu hành động và quan sát

+ Động lực học môi trường

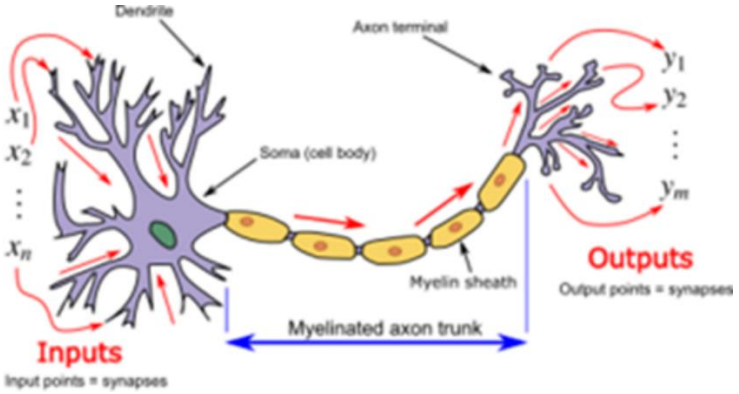
CHƯƠNG 3

MẠNG NƠ-RON NHÂN TẠO

3.1. Tổng quan về mạng nơ-ron nhân tạo

3.1.1. Khái niệm mạng nơ-ron nhân tạo

Mạng nơ-ron nhân tạo (Artificial neural networks ANNs), thường được gọi đơn giản là mạng nơ-ron (NNs), là một hệ thống tính toán lấy cảm hứng từ mạng nơ-ron sinh học mà cấu tạo nên bộ não của động vật. Một mạng nơ-ron được dựa trên cơ sở thu thập những đơn vị kết nối hay nút, được gọi là nơ-ron nhân tạo. Trong đó, mỗi nơ-ron giống như nơ-ron trong bộ não sinh học, mỗi kết nối giống như các khớp thần kinh trong bộ não sinh học, và được gọi là cạnh.

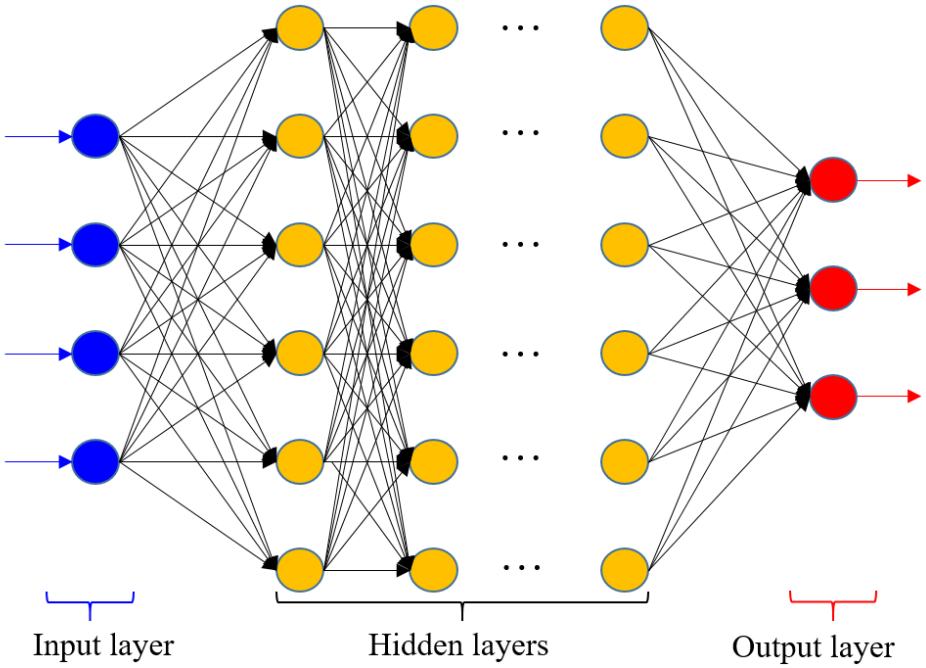


Hình 3.1. Neuron và sợi trục có myelin hóa, với dòng tín hiệu từ đầu vào ở đuôi gai đến đầu ra ở đầu cuối sợi trục

Một nơ-ron nhận được tín hiệu, sau đó, xử lý tín hiệu này và thông báo cho các tế bào thần kinh được kết nối với nó.

- Tín hiệu tại mỗi kết nối là một số thực;
- Các nơ-ron và các cạnh có một trọng số mà được điều chỉnh như là tiến trình học;
- Trọng số tăng hoặc giảm cường độ tín hiệu tại một kết nối.

Thông thường, các nơ-ron được tổng hợp thành các lớp (layer). Các lớp khác nhau có thể thực hiện các phép biến đổi khác nhau trên đầu vào của chúng. Các tín hiệu đi từ lớp đầu tiên (input layer) đến lớp cuối cùng (output layer), có thể sau khi đi qua các lớp nhiều lần.



Hình 3.2. Cấu trúc của mạng nơ-ron

Mạng nơ-ron chia đầu vào thành các lớp trừu tượng. Ví dụ, nó có thể được đào tạo bằng cách sử dụng nhiều ví dụ để nhận dạng các mẫu trong lời nói hoặc hình ảnh, giống như bộ não con người. Hành vi của nó được xác định bằng cách các phần tử riêng lẻ của nó được kết nối và bằng cường độ hoặc trọng số của các kết nối đó. Các trọng số này được tự động điều chỉnh trong quá trình đào tạo theo một quy tắc học cụ thể cho đến khi mạng thần kinh nhân tạo thực hiện chính xác nhiệm vụ mong muốn.

Mạng lưới nơ-ron là một loại phương pháp học máy lấy cảm hứng từ cách các tế bào thần kinh truyền tín hiệu cho nhau trong não người. Mạng lưới thần kinh đặc biệt phù hợp để mô hình hóa các mối quan hệ phi tuyến tính và chúng thường được sử dụng để thực hiện nhận dạng mẫu và phân loại các đối tượng hoặc tín hiệu trong các hệ thống điều khiển, tầm nhìn và giọng nói.

Một số ứng dụng của mạng nơ-ron có thể kể đến là:

- Phân đoạn hình ảnh và video theo ngữ nghĩa
- Phát hiện các đối tượng trong hình ảnh, bao gồm cả người đi bộ và người đi xe đạp
- Huấn luyện robot hai chân đi bộ bằng cách học tăng cường
- Phát hiện ung thư bằng cách hướng dẫn các nhà nghiên cứu bệnh học phân loại khối u là lành tính hay ác tính, dựa trên tính đồng nhất của kích thước tế bào, độ dày của khối, quá trình nguyên phân và các yếu tố khác.

Mạng nơ-ron, đặc biệt là mạng nơ-ron sâu, đã trở nên nổi tiếng nhờ khả năng thành thạo các ứng dụng nhận dạng phức tạp như nhận dạng khuôn mặt, dịch văn bản và nhận dạng giọng nói. Những cách tiếp cận này là công nghệ then chốt thúc đẩy sự đổi mới trong các nhiệm vụ và hệ thống hỗ trợ người lái tiên tiến bao gồm phân loại làn đường và nhận dạng biển báo giao thông.

3.1.2. Cách thức hoạt động của mạng nơ-ron

Một mạng nơ-ron với các nơ-ron (nút) được kết nối chuyển tiếp từ lớp này sang lớp tiếp theo. Như thể hiện trong Hình 3.2, một mạng nơ-ron bao gồm lớp đầu vào, lớp đầu ra và các lớp ẩn. Lớp đầu vào lấy dữ liệu đầu vào để xử lý, trong khi lớp đầu ra chứa các dự đoán hoặc kết quả phân loại. Một hoặc nhiều lớp ẩn nằm giữa lớp đầu vào và lớp đầu ra là công cụ khớp phù hợp

của mạng nơ-ron. Trong mạng nơ-ron, luồng dữ liệu từ lớp đầu vào đến lớp đầu ra theo hướng chuyển tiếp. Các trọng số liên quan được điều chỉnh khi các nơ-ron được huấn luyện bằng thuật toán học lan truyền ngược. Mạng nơ-ron được phát triển để xấp xỉ bất kỳ hàm liên tục nào và có thể giải các bài toán phi tuyến tính.

Đầu ra y của mạng n lớp với đầu vào x có thể được mô tả như sau:

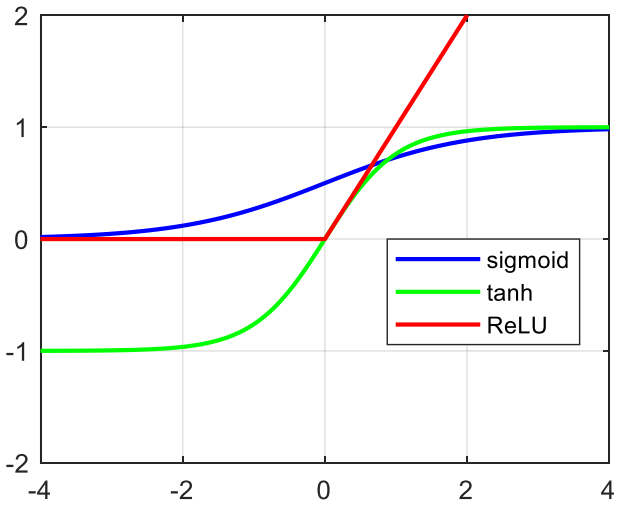
$$y = f(\mathbf{W}_n f(\mathbf{W}_{n-1} \dots f(\mathbf{W}_1 x + b_1) + \dots + b_{n-1}) + b_n)$$

trong đó Wi là ma trận trọng số liên kết với lớp thứ i, vectơ bn (i=1,2, …, n) biểu thị các giá trị sai lệch cho mỗi nút trong lớp thứ i và f là hàm kích hoạt phi tuyến. Một số hàm kích hoạt điển hình là hàm sigmoid (sigmoid), hàm tiếp tuyến hyperbol (tanh) và hàm đơn vị tuyến tính được chỉnh lưu (ReLU). Dưới đây là các phương trình mô tả các hàm này và đồ thị minh họa.

$$\text{sigmoid}(x_j) = \frac{e^{x_j}}{1 + e^{-x_j}}$$

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

$$\text{ReLU}(x) = \max(x, 0)$$



Hình 3.3. Đồ thị của các hàm kích hoạt phổ biến

Giống như các thuật toán học máy khác, mạng nơ-ron có thể được sử dụng để học có giám sát (phân loại, hồi quy) và học không giám sát (nhận dạng mẫu, phân cụm). Các tham số mô hình được thiết lập bằng cách tính trọng số cho mạng thần kinh thông qua việc "học" trên dữ liệu huấn luyện, thường bằng cách tối ưu hóa các trọng số để giảm thiểu lỗi dự đoán.

3.1.3. Các kiểu mạng nơ-ron

Mạng nơ-ron đầu tiên và đơn giản nhất là perceptron, được giới thiệu bởi Frank Rosenblatt vào năm 1958. Nó bao gồm một nơ-ron đơn lẻ và về cơ bản là một mô hình hồi quy tuyến tính với chức năng kích hoạt sigmoid. Kể từ đó, các mạng thần kinh ngày càng phức tạp đã được khám phá, dẫn đến các mạng sâu ngày nay, có thể chứa hàng trăm lớp.

Học sâu (DL) đề cập đến mạng nơ-ron có nhiều lớp, trong khi mạng nơ-ron chỉ có hai hoặc ba lớp nơ-ron được kết nối còn được gọi là mạng nơ-ron nông. DL đã trở nên phổ biến vì nó loại bỏ nhu cầu trích xuất các tính năng từ hình ảnh, điều trước đây đã thách thức việc áp dụng học máy vào xử lý hình ảnh và tín hiệu. Tuy nhiên, mặc dù trích xuất tính năng có thể được bỏ qua trong các ứng dụng xử lý ảnh, một số hình thức trích xuất tính năng vẫn thường được áp dụng cho các tác vụ xử lý tín hiệu để cải thiện độ chính xác của mô hình.

Các loại mạng nơ-ron thường được sử dụng cho các ứng dụng kỹ thuật bao gồm:

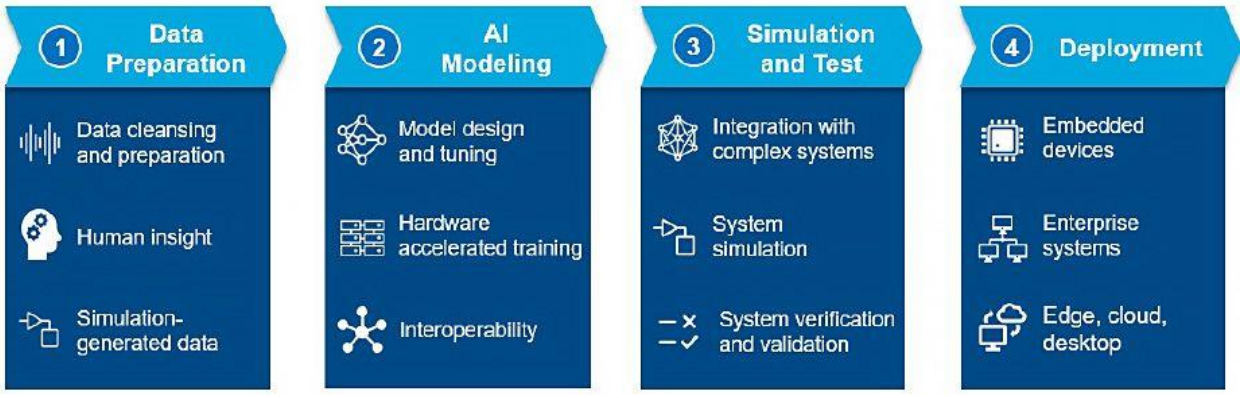
- Mạng nơ ron chuyển tiếp: Bao gồm một lớp đầu vào, một hoặc một vài lớp ẩn và một lớp đầu ra (một mạng nơ ron nông điển hình)
- Mạng thần kinh tích chập (CNN): Kiến trúc mạng thần kinh sâu được áp dụng rộng rãi cho xử lý hình ảnh và được đặc trưng bởi các lớp tích chập giúp dịch chuyển các cửa sổ trên đầu vào với các nút chia sẻ trọng số, trừu tượng hóa đầu vào (thường là hình ảnh) thành bản đồ đặc trưng
- Mạng thần kinh tái phát (RNN): Kiến trúc mạng thần kinh với các vòng phản hồi mô hình hóa các phụ thuộc tuần tự trong đầu vào, chẳng hạn như trong chuỗi thời gian, cảm biến và dữ liệu văn bản; loại RNN phổ biến nhất là mạng bộ nhớ ngắn hạn dài (LSTM).

3.2. Phát triển mạng nơ-ron với MATLAB

MATLAB® cung cấp các hộp công cụ chuyên dụng cho học máy, mạng thần kinh, học sâu, thị giác máy tính và các ứng dụng lái xe tự động. Chỉ với một vài dòng mã, MATLAB cho phép bạn phát triển mạng lưới thần kinh mà không cần phải là một chuyên gia. Bắt đầu nhanh chóng, tạo và trực quan hóa các mô hình mạng thần kinh, tích hợp chúng vào các ứng dụng hiện có của bạn và triển khai chúng cho các máy chủ, hệ thống doanh nghiệp, cụm, đám mây và thiết bị nhúng.

3.2.1. Quy trình công việc điển hình để xây dựng mạng nơ-ron

Việc phát triển các ứng dụng AI và cụ thể là các mạng nơ-ron thường bao gồm các bước sau:



Hình 3.3. Các bước ứng dụng mô hình AI

a) Chuẩn bị dữ liệu
- Dữ liệu được đào tạo cần đủ lớn và được gắn nhãn, với nhiều yêu cầu hơn nữa để đào tạo mạng DL; các ứng dụng gắn nhãn như hình ảnh, video và tín hiệu được gắn nhãn;
- Dữ liệu cũng có thể được tạo bởi mô phỏng, đặc biệt nếu việc thu thập dữ liệu từ các hệ thống thực là không thực tế (ví dụ: điều kiện lỗi);
- Dữ liệu có thể được tăng cường để thể hiện nhiều thay đổi hơn trong dữ liệu đào tạo.

b) Mô hình AI

- Các mạng thần kinh nông có thể được huấn luyện một cách tương tác trong Classification and Regression Learner từ Statistics và Machine Learning Toolbox™ hoặc có thể sử dụng các chức năng dòng lệnh; điều này được khuyến nghị nếu muốn so sánh hiệu suất của mạng thần kinh nông với các thuật toán học máy thông thường khác, chẳng hạn như Cây quyết định hoặc SVM hoặc nếu chỉ có sẵn dữ liệu đào tạo được gắn nhãn hạn chế;
- Chỉ định và huấn luyện mạng nơ-ron (nông hoặc sâu) một cách tương tác bằng cách sử dụng Deep Network Designer hoặc các chức năng dòng lệnh từ Deep Learning Toolbox™, đặc biệt phù hợp với mạng nơ-ron sâu hoặc nếu bạn cần linh hoạt hơn trong việc tùy chỉnh kiến trúc mạng và bộ giải.

c) Mô phỏng và thử nghiệm

Các mạng thần kinh được tích hợp trong các mô hình Simulink® dưới dạng các khối, điều này có thể tạo điều kiện thuận lợi cho việc tích hợp với một hệ thống lớn hơn, thử nghiệm và triển khai cho nhiều loại phần cứng.

d) Triển khai

- Tạo mã C/C++ đơn giản từ các mạng thần kinh nông được đào tạo trong Statistics and Machine Learning Toolbox để triển khai cho các hệ thống máy tính hiệu năng cao và phần cứng nhúng
- Tạo CUDA được tối ưu hóa và mã C/C++ đơn giản từ các mạng thần kinh được đào tạo trong Deep Learning Toolbox để suy luận nhanh về GPU và các loại phần cứng công nghiệp khác (ARM, FPGA).

3.3. Mạng nơ-ron Tuyến tính

Trước khi tìm hiểu chi tiết về mạng nơ-ron sâu, chúng ta cần nắm vững những kiến thức căn bản của việc huấn luyện mạng nơ-ron. Mục này sẽ đề cập đến toàn bộ quá trình huấn luyện, bao gồm xác định kiến trúc mạng nơ-ron đơn giản, xử lý dữ liệu, chỉ rõ hàm mất mát và huấn luyện mô hình. Để mọi thứ dễ dàng hơn, ta sẽ bắt đầu với một số khái niệm đơn giản nhất. May thay, một số phương pháp học thống kê cổ điển như hồi quy tuyến tính, hồi quy logistic có thể được xem như những mạng nơ-ron nông. Hãy bắt đầu bằng những thuật toán cổ điển này, chúng tôi sẽ giới thiệu những nội dung căn bản nhằm tạo nền tảng cho những kỹ thuật phức tạp hơn như Hồi quy Softmax (sẽ được giới thiệu ở cuối chương này) và Perceptron đa tầng (sẽ được giới thiệu ở chương sau).

3.3.1. Hồi quy Tuyến tính

Hồi quy ám chỉ các phương pháp để xây dựng mối quan hệ giữa điểm dữ liệu x và mục tiêu với giá trị số thực y. Trong khoa học tự nhiên và khoa học xã hội, mục tiêu của hồi quy thường là đặc trưng hóa mối quan hệ của đầu vào và đầu ra. Mặt khác, học máy lại thường quan tâm đến việc dự đoán.

Bài toán hồi quy xuất hiện mỗi khi chúng ta muốn dự đoán một giá trị số. Các ví dụ phổ biến bao gồm dự đoán giá cả (nhà, cổ phiếu, …), thời gian bệnh nhân nằm viện, nhu cầu trong ngành bán lẻ và vô vàn thứ khác. Không phải mọi bài toán dự đoán đều là bài toán hồi quy cổ điển. Trong các phần tiếp theo, chúng tôi sẽ giới thiệu bài toán phân loại, khi mục tiêu là dự đoán lớp đúng trong một tập các lớp cho trước.

3.3.1.1. Các Thành phần Cơ bản của Hồi quy Tuyến tính

Hồi quy tuyến tính có lẽ là công cụ tiêu chuẩn đơn giản và phổ biến nhất được sử dụng cho bài toán hồi quy. Xuất hiện từ đầu thế kỉ 19, hồi quy tuyến tính được phát triển từ một vài giả thuyết đơn giản. Đầu tiên, ta giả sử quan hệ giữa các đặc trưng x và mục tiêu y là tuyến tính, do

đó y có thể được biểu diễn bằng tổng trọng số của đầu vào x, cộng hoặc trừ thêm nhiễu của các quan sát. Thứ hai, ta giả sử nhiễu có quy tắc (tuân theo phân phối Gauss). Để tạo động lực, hãy bắt đầu với một ví dụ. Giả sử ta muốn ước lượng giá nhà (bằng đô la) dựa vào diện tích (đơn vị feet vuông) và tuổi đời (theo năm).

Để khớp một mô hình dự đoán giá nhà, chúng ta cần một tập dữ liệu các giao dịch mà trong đó ta biết giá bán, diện tích, tuổi đời cho từng căn nhà. Trong thuật ngữ của học máy, tập dữ liệu này được gọi là dữ liệu huấn luyện hoặc tập huấn luyện, và mỗi hàng (tương ứng với dữ liệu của một giao dịch) được gọi là một ví dụ hoặc mẫu. Thứ mà chúng ta muốn dự đoán (giá nhà) được gọi là mục tiêu hoặc nhãn. Các biến (tuổi đời và diện tích) mà những dự đoán dựa vào được gọi là các đặc trưng hoặc hiệp biến.

Thông thường, chúng ta sẽ dùng n để kí hiệu số lượng mẫu trong tập dữ liệu. Chỉ số i được dùng để xác định một mẫu cụ thể. Ta ký hiệu mỗi điểm dữ liệu đầu vào là $x^{(i)}=[x^{(i)}_1,x^{(i)}_2]$ và nhãn tương ứng là $y^{(i)}$.

3.3.1.1.1. Mô hình Tuyến tính

Giả định tuyến tính trên cho thấy rằng mục tiêu (giá nhà) có thể được biểu diễn bởi tổng có trọng số của các đặc trưng (diện tích và tuổi đời):

$$\text{price} = w_{\text{area}} \cdot \text{area} + w_{\text{age}} \cdot \text{age} + b.$$

Ở đây, w_{area} và w_{age} được gọi là các trọng số, và b được gọi là hệ số điều chỉnh (còn được gọi là độ dời). Các trọng số xác định mức độ đóng góp của mỗi đặc trưng tới đầu ra, còn hệ số điều chỉnh là dự đoán của giá nhà khi tất cả các đặc trưng đều bằng 0. Ngay cả khi không bao giờ có một ngôi nhà có diện tích hoặc tuổi đời bằng không, ta vẫn cần sử dụng hệ số điều chỉnh; nếu không khả năng biểu diễn của mô hình tuyến tính sẽ bị suy giảm.

Cho một tập dữ liệu, mục đích của chúng ta là chọn được các trọng số w và hệ số điều chỉnh b sao cho dự đoán của mô hình khớp nhất với giá nhà thực tế quan sát được trong dữ liệu.

Trong các bài toán mà tập dữ liệu thường chỉ có một vài đặc trưng, biểu diễn tường minh mô hình ở dạng biểu thức dài như trên khá là phổ biến. Trong học máy, chúng ta thường làm việc với các tập dữ liệu nhiều chiều, vì vậy sẽ tốt hơn nếu ta tận dụng các ký hiệu trong đại số tuyến tính. Khi đầu vào của mô hình có d đặc trưng, ta biểu diễn dự đoán $\hat{y}$ bởi

$$\hat{y} = w_1 x_1 + \ldots + w_d x_d + b.$$

Thu thập toàn bộ các đặc trưng vào một vector **x** và toàn bộ các trọng số vào một vector **w**, ta có thể biểu diễn mô hình một cách gọn gàng bằng tích vô hướng:

$$\hat{y} = \mathbf{w}^\top \mathbf{x} + b.$$

Ở đây, vector **x** tương ứng với một điểm dữ liệu. Chúng ta sẽ thấy rằng việc truy cập đến toàn bộ tập dữ liệu sẽ tiện hơn nếu ta biểu diễn tập dữ liệu bằng ma trận **X**. Mỗi hàng của ma trận **X** thể hiện một mẫu và mỗi cột thể hiện một đặc trưng.

Với một tập hợp điểm dữ liệu **X**, kết quả dự đoán $\hat{y}$ có thể được biểu diễn bằng phép nhân giữa ma trận và vector:

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b,$$

Cho một tập dữ liệu huấn luyện **X** và các giá trị mục tiêu đã biết trước **y**, mục tiêu của hồi quy tuyến tính là tìm vector trọng số **w** và hệ số điều chỉnh b sao cho với một điểm dữ liệu mới $\mathbf{x_i}$ được lấy mẫu từ cùng phân phối của tập huấn luyện, giá trị mục tiêu yi sẽ được dự đoán với sai số nhỏ nhất (theo kỳ vọng).

Kể cả khi biết rằng mô hình tuyến tính là lựa chọn tốt nhất để dự đoán **y** từ **x**, chúng ta cũng không kỳ vọng tìm được dữ liệu thực tế mà ở đó y đúng bằng $\mathbf{w}^\top \mathbf{x} + b$ với mọi điểm (x,y). Để dễ hình dung, mọi thiết bị đo lường dùng để quan sát đặc trưng **X** và nhãn **y** đều có sai số nhất định. Chính vì vậy, kể cả khi ta chắc chắn rằng mối quan hệ ẩn sau tập dữ liệu là tuyến tính, chúng ta sẽ thêm một thành phần nhiễu để giải thích các sai số đó.

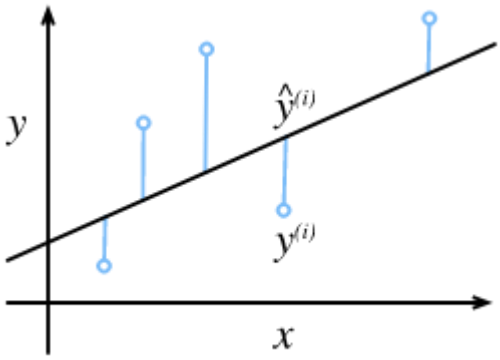
Trước khi tiến hành tìm các giá trị tốt nhất cho **w** và b, chúng ta sẽ cần thêm hai thứ nữa: (i) một phép đo đánh giá chất lượng mô hình và (ii) quy trình cập nhật mô hình để cải thiện chất lượng.

3.3.1.1.2. Hàm mất mát

Trước khi suy nghĩ về việc làm thế nào để khớp mô hình với dữ liệu, ta cần phải xác định một phương pháp để đo mức độ khớp. Hàm mất mát định lượng khoảng cách giữa giá trị thực và giá trị dự đoán của mục tiêu. Độ mất mát thường là một số không âm và có giá trị càng nhỏ càng tốt. Khi các dự đoán hoàn hảo, chúng sẽ có độ mất mát sẽ bằng 0. Hàm mất mát thông dụng nhất trong các bài toán hồi quy là hàm tổng bình phương các lỗi. Khi giá trị dự đoán của một điểm dữ liệu huấn luyện i là $\hat{y}^{(i)}$ và nhãn tương ứng là $y^{(i)}$, bình phương của lỗi được xác định như sau:

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2.$$

Hằng số 1/2 không tạo ra sự khác biệt thực sự nào nhưng sẽ giúp ký hiệu thuận tiện hơn: nó sẽ được triệt tiêu khi lấy đạo hàm của hàm mất mát. Vì các dữ liệu trong tập huấn luyện đã được xác định trước và không thể thay đổi, sai số thực nghiệm chỉ là một hàm của các tham số mô hình. Để tìm hiểu cụ thể hơn, hãy xét ví dụ dưới đây về một bài toán hồi quy cho trường hợp một chiều trong Hình 3.4.



Hình 3.4. Khớp dữ liệu với một mô hình tuyến tính.

Lưu ý rằng khi hiệu giữa giá trị ước lượng $\hat{y}^{(i)}$ và giá trị quan sát $y^{(i)}$ lớn, giá trị hàm mất mát sẽ tăng một lượng còn lớn hơn thế do sự phụ thuộc bậc hai. Để đo chất lượng của mô hình trên toàn bộ tập dữ liệu, ta đơn thuần lấy trung bình (hay tương đương là lấy tổng) các giá trị mất mát của từng mẫu trong tập huấn luyện.

$$L(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)})^2.$$

Khi huấn luyện mô hình, ta muốn tìm các tham số (w∗,b∗) sao cho tổng độ mất mát trên toàn bộ các mẫu huấn luyện được cực tiểu hóa:

$$\mathbf{w}^*, \mathrm{b}^* = \underset{\mathbf{w},\mathrm{b}}{\operatorname{argmin}} \boldsymbol{L}(\mathbf{w}, \mathrm{b})$$

3.3.1.1.3. Nghiệm theo Công thức

Hóa ra hồi quy tuyến tính chỉ là một bài toán tối ưu hóa đơn giản. Khác với hầu hết các mô hình được giới thiệu trong cuốn sách này, hồi quy tuyến tính có thể được giải bằng cách áp dụng một công thức đơn giản, cho một nghiệm tối ưu toàn cục. Để bắt đầu, chúng ta có thể gộp hệ số điều chỉnh b vào tham số w bằng cách thêm một cột toàn 1 vào ma trận dữ liệu. Khi đó bài toán dự đoán trở thành bài toán cực tiểu hóa $\|\mathbf{y}-\mathbf{X}\mathbf{w}\|$. Bởi vì biểu thức này có dạng toàn phương, nó là một hàm số lồi, và miễn là bài toán này không suy biến (các đặc trưng độc lập tuyến tính), nó là một hàm số lồi chặt.

Bởi vậy chỉ có một điểm cực trị trên mặt mất mát và nó tương ứng với giá trị mất mát nhỏ nhất. Lấy đạo hàm của hàm mất mát theo **w** và giải phương trình đạo hàm này bằng 0, ta sẽ được nghiệm theo công thức:

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}.$$

Tuy những bài toán đơn giản như hồi quy tuyến tính có thể có nghiệm theo công thức, bạn không nên làm quen với sự may mắn này. Mặc dù các nghiệm theo công thức giúp ta phân tích toán học một cách thuận tiện, các điều kiện để có được nghiệm này chặt chẽ đến nỗi không có phương pháp học sâu nào thoả mãn được.

3.3.1.1.4. Hạ Gradient

Trong nhiều trường hợp ở đó ta không thể giải quyết các mô hình theo phép phân tích, và thậm chí khi mặt mất mát là các mặt bậc cao và không lồi, trên thực tế ta vẫn có thể huấn luyện các mô hình này một cách hiệu quả. Hơn nữa, trong nhiều tác vụ, những mô hình khó để tối ưu hóa này hoá ra lại tốt hơn các phương pháp khác nhiều, vậy nên việc bỏ công sức để tìm cách tối ưu chúng là hoàn toàn xứng đáng.

Kỹ thuật chính để tối ưu hóa gần như bất kỳ mô hình học sâu nào, sẽ được sử dụng xuyên suốt cuốn sách này, bao gồm việc giảm thiểu lỗi qua các vòng lặp bằng cách cập nhật tham số theo hướng làm giảm dần hàm mất mát. Thuật toán này được gọi là hạ gradient. Trên các mặt mất mát lồi, giá trị mất mát cuối cùng sẽ hội tụ về giá trị nhỏ nhất. Tuy điều tương tự không thể áp dụng cho các mặt không lồi, ít nhất thuật toán sẽ dẫn tới một cực tiểu (hy vọng là tốt).

Ứng dụng đơn giản nhất của hạ gradient bao gồm việc tính đạo hàm của hàm mất mát, tức trung bình của các giá trị mất mát được tính trên mỗi mẫu của tập dữ liệu. Trong thực tế, việc này có thể cực kì chậm. Chúng ta phải duyệt qua toàn bộ tập dữ liệu trước khi thực hiện một lần cập nhật. Vì thế, thường ta chỉ muốn lấy một minibatch ngẫu nhiên các mẫu mỗi khi ta cần tính bước cập nhật. Phương pháp biến thể này được gọi là hạ gradient ngẫu nhiên.

Trong mỗi vòng lặp, đầu tiên chúng ta lấy ngẫu nhiên một minibatch B dữ liệu huấn luyện với kích thước cố định. Sau đó, chúng ta tính đạo hàm (gradient) của hàm mất mát trên minibatch

đó theo các tham số của mô hình. Cuối cùng, gradient này được nhân với tốc độ học $\eta>0$ và kết quả này được trừ đi từ các giá trị tham số hiện tại.

Chúng ta có thể biểu diễn việc cập nhật bằng công thức toán như sau (∂ là ký hiệu đạo hàm riêng của hàm số):

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \partial_{(\mathbf{w},b)} l^{(i)}(\mathbf{w}, b).$$

Tổng kết lại, các bước của thuật toán như sau: (i) khởi tạo các giá trị tham số của mô hình, thường thì sẽ được chọn ngẫu nhiên. (ii) tại mỗi vòng lặp, ta lấy ngẫu nhiên từng batch từ tập dữ liệu (nhiều lần), rồi tiến hành cập nhật các tham số của mô hình theo hướng ngược với gradient.

Khi sử dụng hàm mất mát bậc hai và mô hình tuyến tính, chúng ta có thể biểu diễn bước này một cách tường minh như sau: Lưu ý rằng $\mathbf{w}$ và $\mathbf{x}$ là các vector. Ở đây, việc ký hiệu bằng các vector giúp công thức dễ đọc hơn nhiều so với việc biểu diễn bằng các hệ số như $w_1, w_2, \ldots, w_d$.

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \partial_{\mathbf{w}} l^{(i)}(\mathbf{w}, b) = \mathbf{w} - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \mathbf{x}^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}), \\ b &\leftarrow b - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \partial_{b} l^{(i)}(\mathbf{w}, b) = b - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} (\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}). \end{aligned}$$

Trong phương trình trên, |B| là số ví dụ trong mỗi minibatch (kích thước batch) và η là tốc độ học. Cũng cần phải nhấn mạnh rằng các giá trị của kích thước batch và tốc độ học được lựa chọn trước một cách thủ công và thường không được học thông qua quá trình huấn luyện mô hình. Các tham số này tuy điều chỉnh được nhưng không được cập nhật trong vòng huấn luyện, và được gọi là siêu tham số. Điều chỉnh siêu tham số là quá trình lựa chọn chúng, thường dựa trên kết quả của vòng lặp huấn luyện được đánh giá trên một tập kiểm định riêng biệt.

Sau khi huấn luyện đủ số vòng lặp được xác định trước (hoặc đạt được một tiêu chí dừng khác), ta sẽ ghi lại các tham số mô hình đã được ước lượng, ký hiệu là $\hat{\mathbf{w}}, \hat{b}$ (ký hiệu "mũ" thường thể hiện các giá trị ước lượng). Lưu ý rằng ngay cả khi hàm số thực sự tuyến tính và không có nhiễu, các tham số này sẽ không cực tiểu hóa được hàm mất mát. Mặc dù thuật toán dần dần hội tụ đến một điểm cực tiểu, nó vẫn không thể tới chính xác được cực tiểu đó với số bước hữu hạn.

Hồi quy tuyến tính thực ra là một bài toán tối ưu lồi, do đó chỉ có một cực tiểu (toàn cục). Tuy nhiên, đối với các mô hình phức tạp hơn, như mạng sâu, mặt của hàm mất mát sẽ có nhiều cực tiểu. May mắn thay, vì một lý do nào đó mà những người làm về học sâu hiếm khi phải vật lộn để tìm ra các tham số cực tiểu hóa hàm mất mát trên dữ liệu huấn luyện. Nhiệm vụ khó khăn hơn là tìm ra các tham số dẫn đến giá trị mất mát thấp trên dữ liệu mà mô hình chưa từng thấy trước đây, một thử thách được gọi là sự khái quát hóa. Chúng ta sẽ gặp lại chủ đề này xuyên suốt cuốn sách.

3.3.1.1.5. Dự đoán bằng Mô hình đã được Huấn luyện

Với mô hình hồi quy tuyến tính đã được huấn luyện $\hat{\mathbf{w}}^\top \mathbf{x} + \hat{b}$, ta có thể ước lượng giá của một căn nhà mới (ngoài bộ dữ liệu dùng để huấn luyện) với diện tích x_1 và tuổi đời x_2 của nó. Việc ước lượng mục tiêu khi biết trước những đặc trưng thường được gọi là dự đoán hay suy luận (inference).

Ở đây ta sẽ dùng từ dự đoán thay vì suy luận, dù suy luận là một thuật ngữ khá phổ biến trong học sâu, áp dụng thuật ngữ này ở đây lại không phù hợp. Trong thống kê, suy luận thường

được dùng cho việc ước lượng thông số dựa trên tập dữ liệu. Việc dùng sai thuật ngữ này là nguyên nhân gây ra sự hiểu nhầm giữa những người làm học sâu và các nhà thống kê.

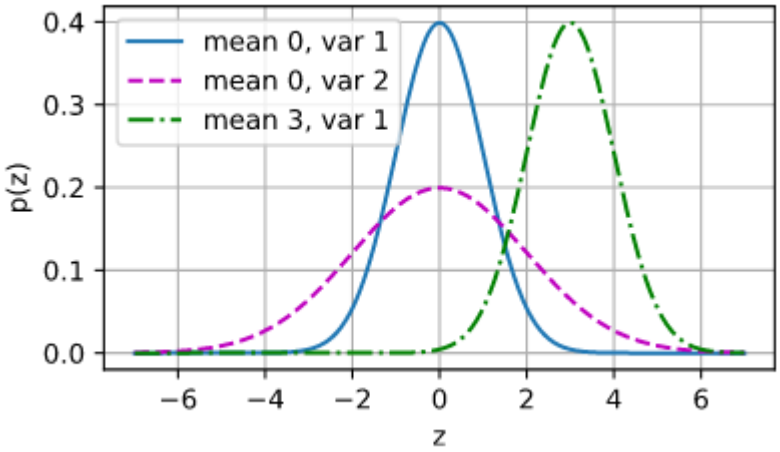
3.3.1.2. Phân phối Chuẩn và Hàm mất mát Bình phương

Mặc dù bạn đã có thể thực hành với kiến thức được trình bày phía trên, trong phần tiếp theo chúng ta sẽ làm rõ hơn nguồn gốc của hàm mất mát bình phương thông qua các giả định về phân phối của nhiễu.

Nhắc lại ở trên rằng hàm mất mát bình phương $l\left(y,\hat{y}\right)=\frac{1}{2}\left(y-\hat{y}\right)^2$ có nhiều thuộc tính tiện lợi. Việc nó có đạo hàm đơn giản $\partial_{\hat{y}}l\left(y,\hat{y}\right)=\left(\hat{y}-y\right)$ là một trong số đó.

Như được đề cập trước đó, hồi quy tuyến tính được phát minh bởi Gauss vào năm 1795. Ông cũng là người khám phá ra phân phối chuẩn (còn được gọi là phân phối Gauss). Hóa ra là mối liên hệ giữa phân phối chuẩn và hồi quy tuyến tính không chỉ dừng lại ở việc chúng có chung cha đẻ. Để gợi nhớ lại cho bạn, mật độ xác suất của phân phối chuẩn với trung bình μ và phương sai σ^2 được cho bởi:

$$p(x)=\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right).$$



Hình 3.5. Mật độ xác suất của phân phối chuẩn

Có thể thấy rằng, thay đổi giá trị trung bình tương ứng với việc dịch chuyển phân phối dọc theo trục x, tăng giá trị phương sai sẽ trải rộng phân phối và hạ thấp đỉnh của nó.

Để thấy rõ hơn mối quan hệ giữa hồi quy tuyến tính và hàm mất mát trung bình bình phương sai số (MSE), ta có thể giả định rằng các quan sát bắt nguồn từ những quan sát nhiễu, và giá trị nhiễu này tuân theo phân phối chuẩn như sau:

$$y=\mathbf{w}^\top\mathbf{x}+b+\epsilon \text{ với } \epsilon\sim\mathcal{N}(0,\sigma^2).$$

Do đó, chúng ta có thể viết khả năng thu được một giá trị cụ thể của **y** khi biết trước **x** là

$$P(y\mid\mathbf{x})=\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2\sigma^2}(y-\mathbf{w}^\top\mathbf{x}-b)^2\right).$$

Dựa vào nguyên lý hợp lý cực đại, giá trị tốt nhất của b và **w** là những giá trị giúp cực đại hóa sự hợp lý của toàn bộ tập dữ liệu:

$$P(\mathbf{y} \mid \mathbf{X}) = \prod_{i=1}^{n} p(y^{(i)}|\mathbf{x}^{(i)}).$$

Bộ ước lượng được chọn theo nguyên lý hợp lý cực đại được gọi là bộ ước lượng hợp lý cực đại (Maximum Likelihood Estimators – MLE). Dù việc cực đại hóa tích của nhiều hàm mũ trông có vẻ khó khăn, chúng ta có thể khiến mọi thứ đơn giản hơn nhiều mà không làm thay đổi mục tiêu ban đầu bằng cách cực đại hóa log của hàm hợp lý. Vì lý do lịch sử, các bài toán tối ưu thường được biểu diễn dưới dạng bài toán cực tiểu hóa thay vì cực đại hóa. Do đó chúng ta có thể cực tiểu hóa hàm đối log hợp lý (Negative Log-Likelihood - NLL) $-\log P(\mathbf{y} \mid \mathbf{X})$ mà không cần thay đổi gì thêm. Kết nối các công thức trên, ta có:

$$-\log P(\mathbf{y} \mid \mathbf{X}) = \sum_{i=1}^{n} \frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)} - b)^2.$$

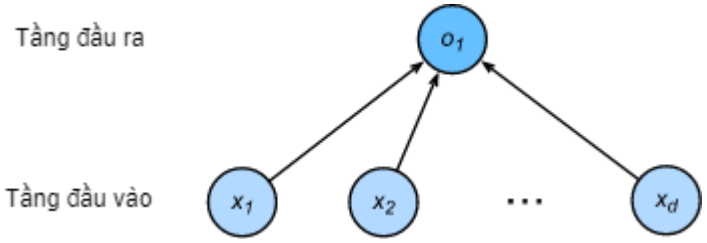
Giờ ta chỉ cần thêm một giả định nữa: σ là một hằng số cố định. Do đó, ta có thể bỏ qua số hạng đầu tiên bởi nó không phụ thuộc vào **w** hoặc b. Còn số hạng thứ hai thì giống hệt hàm bình phương sai số đã được giới thiệu ở trên, nhưng được nhân thêm với hằng số $\frac{1}{\sigma^2}$. May mắn thay, nghiệm không phụ thuộc vào σ. Điều này dẫn tới việc cực tiểu hóa bình phương sai số tương đương với việc ước lượng hợp lý cực đại cho mô hình tuyến tính dưới giả định có nhiễu cộng Gauss.

3.3.1.3. Từ Hồi quy Tuyến tính tới Mạng Học sâu

Cho đến nay, chúng ta mới chỉ đề cập về các hàm tuyến tính. Trong khi mạng nơ-ron có thể xấp xỉ rất nhiều họ mô hình, ta có thể bắt đầu coi mô hình tuyến tính như một mạng nơ-ron và biểu diễn nó theo ngôn ngữ của mạng nơ-ron. Để bắt đầu, hãy cùng viết lại mọi thứ theo ký hiệu 'tầng' (layer).

3.3.1.3.1. Giản đồ Mạng Nơ-ron

Những người làm học sâu thích vẽ giản đồ để trực quan hóa những gì đang xảy ra trong mô hình của họ. Trong Hình 3.6, mô hình tuyến tính được minh họa như một mạng nơ-ron. Những giản đồ này chỉ ra cách kết nối (ở đây, mỗi đầu vào được kết nối tới đầu ra) nhưng không có giá trị của các trọng số và các hệ số điều chỉnh.

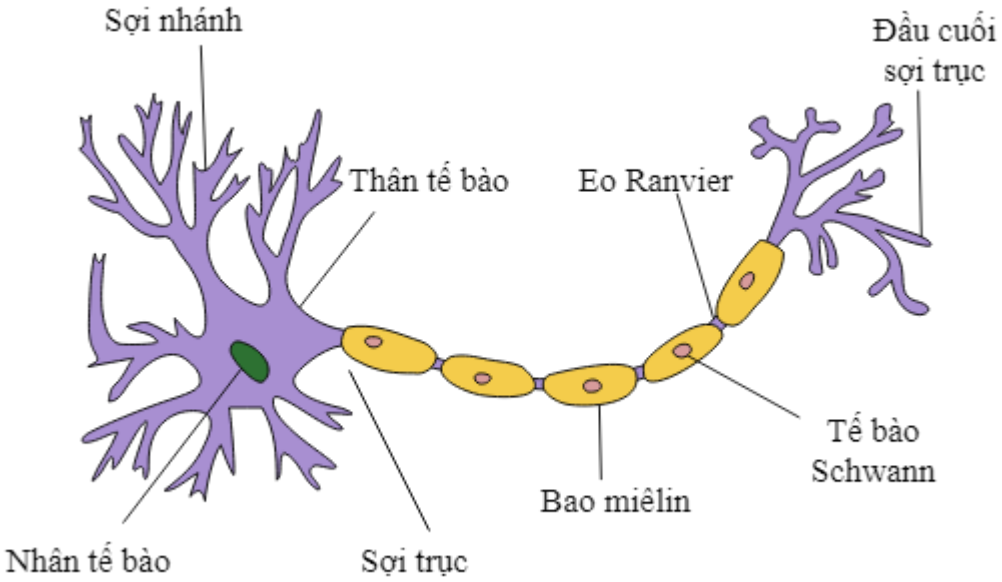


Hình 3.6. Hồi quy tuyến tính là một mạng nơ-ron đơn tầng.

Vì chỉ có một nơ-ron tính toán (một nút) trong đồ thị (các giá trị đầu vào không cần tính mà được cho trước), chúng ta có thể coi mô hình tuyến tính như mạng nơ-ron với chỉ một nơ-ron nhân tạo duy nhất. Với mô hình này, mọi đầu vào đều được kết nối tới mọi đầu ra (trong trường hợp này chỉ có một đầu ra!), ta có thể coi phép biến đổi này là một tầng kết nối đầy đủ, hay còn gọi là tầng kết nối dày đặc. Chúng ta sẽ nói nhiều hơn về các mạng nơ-ron cấu tạo từ những tầng như vậy trong chương kế tiếp về mạng perceptron đa tầng.

3.3.1.3.2. Sinh vật học

Vì hồi quy tuyến tính (được phát minh vào năm 1795) được phát triển trước ngành khoa học thần kinh tính toán, nên việc mô tả hồi quy tuyến tính như một mạng nơ-ron có vẻ hơi ngược thời. Để hiểu tại sao nhà nghiên cứu sinh vật học/thần kinh học Warren McCulloch và Walter Pitts tìm đến các mô hình tuyến tính để làm điểm khởi đầu nghiên cứu và phát triển các mô hình nơ-ron nhân tạo, hãy xem ảnh của một nơ-ron sinh học tại Hình 3.7. Mô hình này bao gồm sợi nhánh (cổng đầu vào), nhân tế bào (bộ xử lý trung tâm), sợi trục (dây đầu ra), và đầu cuối sợi trục (cổng đầu ra), cho phép kết nối với các tế bào thần kinh khác thông qua synapses.



Hình 3.7. Nơ-ron trong thực tế

Thông tin x_i đến từ các nơ-ron khác (hoặc các cảm biến môi trường như võng mạc) được tiếp nhận tại các sợi nhánh. Cụ thể, thông tin đó được nhân với các trọng số của synapses w_i để xác định mức ảnh hưởng của từng đầu vào (ví dụ: kích hoạt hoặc ức chế thông qua tích $x_i w_i$). Các đầu vào có trọng số đến từ nhiều nguồn được tổng hợp trong nhân tế bào dưới dạng tổng có trọng số $y = \sum_i x_i w_i + b$ và thông tin này sau đó được gửi đi để xử lý thêm trong sợi trục y, thường là sau một vài xử lý phi tuyến tính qua σ(y). Từ đó, nó có thể được gửi đến đích (ví dụ, cơ bắp) hoặc được đưa vào một tế bào thần kinh khác thông qua các sợi nhánh.

Dựa trên các nghiên cứu thực tế về các hệ thống thần kinh sinh học, ta chắc chắn một điều rằng nhiều đơn vị như vậy khi được kết hợp với nhau theo đúng cách, cùng với thuật toán học phù hợp, sẽ tạo ra các hành vi thú vị và phức tạp hơn nhiều so với bất kỳ nơ-ron đơn lẻ nào có thể làm được.

Đồng thời, hầu hết các nghiên cứu trong học sâu ngày nay chỉ lấy một phần cảm hứng nhỏ từ ngành thần kinh học. Như trong cuốn sách kinh điển về AI Trí tuệ Nhân tạo: Một hướng Tiếp cận Hiện đại [Russell & Norvig, 2016] của Stuart Russell và Peter Norvig, họ đã chỉ ra rằng: mặc dù máy bay có thể được lấy cảm hứng từ loài chim, ngành điểu học không phải động lực chính làm đổi mới ngành hàng không trong nhiều thế kỷ qua. Tương tự, cảm hứng trong học sâu hiện nay chủ yếu đến từ ngành toán học, thống kê và khoa học máy tính.

3.3.1.4. Kết luận

- Nguyên liệu của một mô hình học máy bao gồm dữ liệu huấn luyện, một hàm mất mát, một thuật toán tối ưu, và tất nhiên là cả chính mô hình đó.

- Vector hóa giúp mọi thứ trở nên dễ hiểu hơn (về mặt toán học) và nhanh hơn (về mặt lập trình).

- Cực tiểu hóa hàm mục tiêu và thực hiện phương pháp hợp lý cực đại có ý nghĩa giống nhau.

- Các mô hình tuyến tính cũng là các mạng nơ-ron.

3.3.4. Hồi quy Softmax

Trong phần trước, chúng ta đã giới thiệu về hồi quy tuyến tính, từ việc tự xây dựng mô hình hồi quy tuyến tính cho đến xây dựng mô hình hồi quy tuyến tính với Gluon thực hiện phần việc nặng nhọc.

Hồi quy là công cụ đắc lực có thể sử dụng khi ta muốn trả lời câu hỏi bao nhiêu?. Nếu bạn muốn dự đoán một ngôi nhà sẽ được bán với giá bao nhiêu tiền (Đô la), hay số trận thắng mà một đội bóng có thể đạt được, hoặc số ngày một bệnh nhân phải điều trị nội trú trước khi được xuất viện, thì có lẽ bạn đang cần một mô hình hồi quy.

Trong thực tế, chúng ta thường quan tâm đến việc phân loại hơn: không phải câu hỏi bao nhiêu? mà là loại nào?

- Email này có phải thư rác hay không?

- Khách hàng này nhiều khả năng đăng ký hay không đăng ký một dịch vụ thuê bao?

- Hình ảnh này mô tả một con lừa, một con chó, một con mèo hay một con gà trống?

- Bộ phim nào có khả năng cao nhất được Aston xem tiếp theo?

Thông thường, những người làm về học máy dùng từ phân loại để mô tả đôi chút sự khác nhau giữa hai bài toán: (i) ta chỉ quan tâm đến việc gán cứng một danh mục cho mỗi ví dụ: là chó, là gà, hay là mèo?; và (ii) ta muốn gán mềm tất cả các danh mục cho mỗi ví dụ, tức đánh giá xác suất một ví dụ rơi vào từng danh mục khả dĩ: là chó (92%), là gà (1%), là mèo (7%). Sự khác biệt này thường không rõ ràng, một phần bởi vì thông thường ngay cả khi chúng ta chỉ quan tâm đến việc gán cứng, chúng ta vẫn sử dụng các mô hình thực hiện các phép gán mềm.

3.3.4.1. Bài toán Phân loại

Hãy khởi động với một bài toán phân loại hình ảnh đơn giản. Ở đây, mỗi đầu vào là một ảnh xám có kích thước 2×2. Bằng cách biểu diễn mỗi giá trị điểm ảnh bởi một số vô hướng, ta thu được bốn đặc trưng x_1, x_2, x_3, x_4. Hơn nữa, giả sử rằng mỗi hình ảnh đều thuộc về một trong các danh mục “mèo”, “gà” và “chó”.

Tiếp theo, ta cần phải chọn cách biểu diễn nhãn. Ta có hai cách làm hiển nhiên. Cách tự nhiên nhất có lẽ là chọn $y \in \{1, 2, 3\}$ lần lượt ứng với {chó, mèo, gà}. Đây là một cách lưu trữ thông tin tuyệt vời trên máy tính. Nếu các danh mục có một thứ tự tự nhiên giữa chúng, chẳng hạn như {trẻ sơ sinh, trẻ tập đi, thiếu niên, thanh niên, người trưởng thành, người cao tuổi}, sẽ là tự nhiên hơn nếu coi bài toán này là một bài toán hồi quy và nhãn sẽ được giữ nguyên dưới dạng số.

Nhưng nhìn chung các lớp của bài toán phân loại không tuân theo một trật tự tự nhiên nào. May mắn thay, các nhà thông kê từ lâu đã tìm ra một cách đơn giản để có thể biểu diễn dữ liệu danh mục: biểu diễn one-hot. Biểu diễn one-hot là một vector với số lượng thành phần bằng số

danh mục mà ta có. Thành phần tương ứng với từng danh mục cụ thể sẽ được gán giá trị 1 và tất cả các thành phần khác sẽ được gán giá trị 0.

$$y \in \{(1,0,0), (0,1,0), (0,0,1)\}.$$

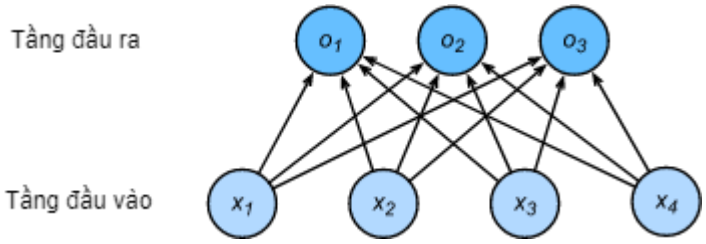
Trong trường hợp này, y sẽ là một vector 3 chiều, với (1,0,0) tương ứng với "mèo", (0,1,0) ứng với "gà" và (0,0,1) ứng với "chó".

3.3.4.1.1. Kiến trúc mạng

Để tính xác suất có điều kiện ứng với mỗi lớp, chúng ta cần một mô hình có nhiều đầu ra với một đầu ra cho mỗi lớp. Để phân loại với các mô hình tuyến tính, chúng ta cần số hàm tuyến tính tương đương số đầu ra. Mỗi đầu ra sẽ tương ứng với hàm tuyến tính của chính nó. Trong trường hợp này, vì có 4 đặc trưng và 3 đầu ra, chúng ta sẽ cần 12 số vô hướng để thể hiện các trọng số, (w với các chỉ số dưới) và 3 số vô hướng để thể hiện các hệ số điều chỉnh (b với các chỉ số dưới). Chúng ta sẽ tính ba logits, o_1,o_2, và o_3, cho mỗi đầu vào:

$$\begin{aligned} o_1 &= x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1, \\ o_2 &= x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2, \\ o_3 &= x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3. \end{aligned}$$

Chúng ta có thể mô tả phép tính này với biểu đồ mạng nơ-ron được thể hiện trong Hình 3.8. Như hồi quy tuyến tính, hồi quy softmax cũng là một mạng nơ-ron đơn tầng. Và vì sự tính toán của mỗi đầu ra, o_1, o_2, và o_3, phụ thuộc vào tất cả đầu vào, x_1, x_2, x_3, và x_4, tầng đầu ra của hồi quy softmax cũng có thể được xem như một tầng kết nối đầy đủ.



Hình 3.8. Hồi quy sofmax là một mạng nơ-ron đơn tầng

Để biểu diễn mô hình gọn hơn, chúng ta có thể sử dụng ký hiệu đại số tuyến tính. Ở dạng vector, ta có **o**=**Wx**+b, một dạng phù hợp hơn cho cả toán và lập trình. Chú ý rằng chúng ta đã tập hợp tất cả các trọng số vào một ma trận 3×4 và với một mẫu cho trước **x**, các đầu ra được tính bởi tích ma trận-vector của các trọng số và đầu vào cộng với vector hệ số điều chỉnh b.

3.3.4.1.2. Hàm Softmax

Chúng ta sẽ xem các giá trị đầu ra của mô hình là các giá trị xác suất. Ta sẽ tối ưu hóa các tham số của mô hình sao cho khả năng xuất hiện dữ liệu quan sát được là cao nhất. Sau đó, ta sẽ đưa ra dự đoán bằng cách đặt ngưỡng xác suất, ví dụ dự đoán nhãn đúng là nhãn có xác suất cao nhất (dùng hàm argmax).

Nói một cách chính quy hơn, ta mong muốn diễn dịch kết quả $\hat{y}_k$ là xác suất để một điểm dữ liệu cho trước thuộc về một lớp k nào đó. Sau đó, ta có thể chọn lớp cho điểm đó tương ứng với giá trị lớn nhất mà mô hình dự đoán $\text{argmax}_k y_k$. Ví dụ, nếu $\hat{y}_1$, $\hat{y}_2$, và $\hat{y}_3$ lần lượt là 0.1, 0.8, và 0.1, thì ta có thể dự đoán điểm đó thuộc về lớp số 2 là "gà" (ứng với trong ví dụ trước).

Bạn có thể muốn đề xuất rằng ta lấy trực tiếp logit **o** làm đầu ra mong muốn. Tuy nhiên, sẽ có vấn đề khi coi kết qủa trả về trực tiếp từ tầng tuyến tính như là các giá trị xác suất. Lý do là

không có bất cứ điều kiện nào để ràng buộc tổng của những con số này bằng 1. Hơn nữa, tùy thuộc vào đầu vào mà ta có thể nhận được giá trị âm. Các lý do trên khiến kết quả của tầng tuyến tính vi phạm vào các tiên đề cơ bản của xác xuất đã được nhắc đến trong Section 2.6.

Để có thể diễn dịch kết quả đầu ra là xác xuất, ta phải đảm bảo rằng các kết quả không âm và tổng của chúng phải bằng 1 (điều này phải đúng trên cả dữ liệu mới). Hơn nữa, ta cần một hàm mục tiêu trong quá trình huấn luyện để cho mô hình có thể ước lượng xác suất một cách chính xác. Trong tất cả các trường hợp, khi kết quả phân lớp cho ra xác suất là 0.5 thì ta hy vọng phân nửa số mẫu đó thực sự thuộc về đúng lớp được dự đoán. Đây được gọi là hiệu chuẩn.

Hàm softmax, được phát minh vào năm 1959 bởi nhà khoa học xã hội R Duncan Luce với chủ đề mô hình lựa chọn, thỏa mãn chính xác những điều trên. Để biến đổi kết quả logit thành kết quả không âm và có tổng là 1, trong khi vẫn giữ tính chất khả vi, đầu tiên ta cần lấy hàm mũ cho từng logit (để chắc chắn chúng không âm) và sau đó ta chia cho tổng của chúng (để chắc rằng tổng của chúng luôn bằng 1).

$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o}) \text{ trong đó } \hat{y}_j = \frac{\exp(o_j)}{\sum_k \exp(o_k)}.$$

Dễ thấy rằng $\hat{y}_1 + \hat{y}_2 + \hat{y}_3 = 1$ với $0 \leq \hat{y}_j \leq 1$ với mọi j. Do đó, $\hat{\mathbf{y}}$ là phân phối xác suất phù hợp và các giá trị của $\hat{\mathbf{y}}$ có thể được hiểu theo đó. Lưu ý rằng hàm softmax không thay đổi thứ tự giữa các logit và do đó ta vẫn có thể chọn ra lớp phù hợp nhất bằng cách:

$$\underset{j}{\mathrm{argmax}} \hat{y}_j = \underset{j}{\mathrm{argmax}} o_j.$$

Các logit **o** đơn giản chỉ là các giá trị trước khi cho qua hàm softmax để xác định xác xuất thuộc về mỗi danh mục. Tóm tắt lại, ta có ký hiệu dưới dạng vector như sau: $o_j = Wx_j + b$, với $\hat{y}_j = softmax(o_j)$.

3.3.4.1.3. Vector hóa Minibatch

Để cải thiện hiệu suất tính toán và tận dụng GPU, ta thường phải thực hiện các phép tính vector cho các minibatch dữ liệu. Giả sử, ta có một minibatch **X** của mẫu với số chiều d và kích thước batch là n. Thêm vào đó, chúng ta có q lớp đầu ra. Như vậy, minibatch đặc trưng **X** sẽ thuộc $\mathbf{R}^{n\times d}$, trọng số $W \in \mathbf{R}^{d\times q}$, và độ chệch sẽ thỏa mãn $b \in \mathbf{R}^{q}$.

$$\begin{aligned} \mathbf{O} &= \mathbf{XW} + \mathbf{b}, \\ \hat{\mathbf{Y}} &= \mathrm{softmax}(\mathbf{O}). \end{aligned}$$

Việc tăng tốc diễn ra chủ yếu tại tích ma trận - ma trận **WX** so với tích ma trận - vector nếu chúng ta xử lý từng mẫu một. Bản thân softmax có thể được tính bằng cách lũy thừa tất cả các mục trong **O** và sau đó chuẩn hóa chúng theo tổng.

3.3.4.2. Hàm mất mát

Tiếp theo, chúng ta cần một hàm mất mát để đánh giá chất lượng các dự đoán xác suất. Chúng ta sẽ dựa trên hợp lý cực đại, khái niệm tương tự đã gặp khi đưa ra lý giải xác suất cho hàm mục tiêu bình phương nhỏ nhất trong hồi quy tuyến tính (Section 3.1).

3.3.4.2.1. Log hợp lý

Hàm softmax cho chúng ta một vector $\hat{\mathbf{y}}$, có thể được hiểu như các xác suất có điều kiện của từng lớp với đầu vào **x**. Ví dụ: $P(y = \text{cat} \mid \mathbf{x})$. Để biết các ước lượng có sát với thực tế hay không, ta kiểm tra xác suất mà mô hình gán cho lớp thật sự khi biết các đặc trưng.

$$P(\mathbf{Y} \mid \mathbf{X}) = \prod_{i=1}^{n} P(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}).$$

Và vì vậy, $-\log P(\mathbf{Y} \mid \mathbf{X}) = \sum_{i=1}^{n} -\log P(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}) = \sum_{i=1}^{n} l(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)})$,

Cực đại hoá $P(\mathbf{Y} \mid \mathbf{X})$ (và vì vậy tương đương với cực tiểu hóa $-\log P(\mathbf{Y} \mid \mathbf{X})$ giúp việc dự đoán nhãn tốt hơn. Điều này dẫn đến hàm mất mát (chúng tôi lược bỏ chỉ số trên (i) để tránh sự nhập nhằng về kí hiệu):

$$l\left(\mathbf{y}, \hat{\mathbf{y}}\right) = -\log P(\mathbf{y} \mid \mathbf{x}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j.$$

Bởi vì những lý do sẽ được giải thích sau đây, hàm mất mát này thường được gọi là mất mát entropy chéo. Ở đây, chúng ta đã sử dụng nó bằng cách xây dựng $\hat{\mathbf{y}}$ giống như một phân phối xác suất rời rạc và vector **y** là một vector one-hot. Do đó, tổng các số hạng với chỉ số j sẽ tiêu biến tạo thành một giá trị duy nhất. Bởi mọi $\hat{y}_j$ đều là xác suất, log của chúng không bao giờ lớn hơn 0. Vì vậy, hàm mất mát sẽ không thể giảm thêm được nữa nếu chúng ta dự đoán chính xác **y** với độ chắc chắn tuyệt đối, tức **P(y|x)=1** cho nhãn đúng. Chú ý rằng điều này thường không khả thi. Ví dụ, nhãn bị nhiễu sẽ xuất hiện trong tập dữ liệu (một vài mẫu bị dán nhầm nhãn). Điều này cũng khó xảy ra khi những đặc trưng đầu vào không chứa đủ thông tin để phân loại các mẫu một cách hoàn hảo.

3.3.4.2.2. Softmax và Đạo hàm

Vì softmax và hàm mất mát softmax rất phổ biến, nên việc hiểu cách tính giá trị các hàm này sẽ có ích về sau. Thay **o** vào định nghĩa của hàm mất mát l và dùng định nghĩa của softmax, ta được:

$$\begin{aligned} l(\mathbf{y}, \hat{\mathbf{y}}) &= -\sum_{j=1}^{q} y_j \log \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} \\ &= \sum_{j=1}^{q} y_j \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j \\ &= \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j \,. \end{aligned}$$

Để hiểu rõ hơn, hãy cùng xét đạo hàm riêng của l theo **o**. Ta có:

$$\partial_{o_j} l(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} - y_j = \text{softmax}(\mathbf{o})_j - y_j.$$

Nói cách khác, gradient chính là hiệu giữa xác xuất mô hình gán cho lớp đúng **P(y|x)**, và nhãn của dữ liệu **y**. Điều này cũng tương tự như trong bài toán hồi quy, khi gradient là hiệu giữa

dữ liệu quan sát được y và kết quả ước lượng $\hat{\mathbf{y}}$. Đây không phải là ngẫu nhiên. Trong mọi mô hình họ lũy thừa, gradient của hàm log hợp lý đều có dạng như thế này. Điều này giúp cho việc tính toán gradient trong thực tế trở nên dễ dàng hơn.

3.3.4.2.3. Hàm mất mát Entropy chéo

Giờ hãy xem xét trường hợp mà ta quan sát được toàn bộ phân phối của đầu ra thay vì chỉ một giá trị đầu ra duy nhất. Ta có thể biểu diễn **y** giống hệt như trước. Sự khác biệt duy nhất là thay vì có một vector chỉ chứa các phần tử nhị phân, giả sử như (0,0,1), giờ ta có một vector xác suất tổng quát, ví dụ như (0.1,0.2,0.7). Các công thức toán học ta dùng trước đó để định nghĩa hàm mất mát l vẫn áp dụng tốt ở đây nhưng khái quát hơn một chút. Giá trị của các phần tử trong vector tương ứng giá trị kỳ vọng của hàm mất mát trên phân phối của nhãn.

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j.$$

Hàm trên được gọi là hàm mát mát entropy chéo và là một trong những hàm mất mát phổ biến nhất dùng cho bài toán phân loại đa lớp. Ta có thể làm sáng tỏ cái tên entropy chéo bằng việc giới thiệu các kiến thức cơ bản trong lý thuyết thông tin.

3.3.4.3. Lý thuyết Thông tin Cơ bản

Lý thuyết thông tin giải quyết các bài toán mã hóa, giải mã, truyền tải và xử lý thông tin (hay còn được gọi là dữ liệu) dưới dạng ngắn gọn nhất có thể.

3.3.4.3.1. Entropy

Ý tưởng cốt lõi trong lý thuyết thông tin chính là việc định lượng lượng thông tin chứa trong dữ liệu. Giá trị định lượng này chỉ ra giới hạn tối đa cho khả năng nén dữ liệu (khi tìm biểu diễn ngắn gọn nhất mà không mất thông tin). Giá trị định lượng này gọi là entropy, xác định trên phân phối p của bộ dữ liệu, được định nghĩa bằng phương trình dưới đây:

$$H[p] = \sum_{j} -p(j) \log p(j)$$

Một định lý căn bản của lý thuyết thông tin là để có thể mã hóa dữ liệu thu thập ngẫu nhiên từ phân phối p, chúng ta cần sử dụng ít nhất $H[p]$ "nat". "nat" là đơn vị biểu diễn dữ liệu sử dụng cơ số e, tương tự với bit biểu diễn dữ liệu sử dụng cơ số 2. Một nat bằng $\frac{1}{\log(2)} \approx 1.44$ bit. $H[p]/2$ thường được gọi là entropy nhị phân.

3.3.4.3.2. Lượng tin

Có lẽ bạn sẽ tự hỏi việc nén dữ liệu thì liên quan gì với việc đưa ra dự đoán? Hãy tưởng tượng chúng ta có một luồng (stream) dữ liệu cần nén. Nếu ta luôn có thể dễ dàng đoán được đơn vị dữ liệu (token) kế tiếp thì dữ liệu này rất dễ nén! Ví như tất cả các đơn vị dữ liệu trong dòng dữ liệu luôn có một giá trị cố định thì đây là một dòng dữ liệu tẻ nhạt! Không những tẻ nhạt, mà nó còn dễ đoán nữa. Bởi vì chúng luôn có cùng giá trị, ta sẽ không phải truyền bất cứ thông tin nào để trao đổi nội dung của dòng dữ liệu này. Dễ đoán thì cũng dễ nén là vậy.

Tuy nhiên, nếu ta không thể dự đoán một cách hoàn hảo cho mỗi sự kiện, thì thi thoảng ta sẽ thấy ngạc nhiên. Sự ngạc nhiên trong chúng ta sẽ lớn hơn khi ta gán một xác suất thấp hơn cho sự kiện. Vì nhiều lý do mà chúng ta sẽ nghiên cứu trong phần phụ lục, Claude Shannon đã đưa ra

giải pháp $\log \frac{1}{p(j)} = -\log(j)$ để định lượng sự ngạc nhiên của một người lúc quan sát sự kiện j sau khi đã gán cho sự kiện đó một xác suất (chủ quan) $p(j)$. Entropy lúc này sẽ là lượng tin (độ ngạc nhiên) kỳ vọng khi mà xác suất của các sự kiện đó được gán chính xác, khớp với phân phối sinh dữ liệu. Nói cách khác, entropy là lượng thông tin hay mức độ ngạc nhiên tối thiểu mà dữ liệu sẽ đem lại theo kỳ vọng.

3.4.3.3. Xem xét lại Entropy chéo

Nếu entropy là mức độ ngạc nhiên trải nghiệm bởi một người nắm rõ xác suất thật, thì bạn có thể băn khoăn rằng entropy chéo là gì? Entropy chéo từ p đến q, ký hiệu H(p,q), là sự ngạc nhiên kỳ vọng của một người quan sát với các xác suất chủ quan q đối với dữ liệu sinh ra dựa trên các xác suất p. Giá trị entropy chéo thấp nhất có thể đạt được khi p=q. Trong trường hợp này, entropy chéo từ p đến q là H(p,p)=H(p). Liên hệ điều này lại với mục tiêu phân loại của chúng ta, thậm chí khi ta có khả năng dự đoán tốt nhất có thể và cho rằng việc này là khả thi, thì ta sẽ không bao giờ đạt đến mức hoàn hảo. Mất mát của ta bị giới hạn dưới (lower-bounded) bởi entropy tạo bởi các phân phối thực tế có điều kiện P(y|x).

3.4.3.4. Phân kỳ Kullback Leibler

Có lẽ cách thông dụng nhất để đo lường khoảng cách giữa hai phân phối là tính toán phân kỳ Kullback Leibler $D(p||q)$. Phân kỳ Kullback Leibler đơn giản là sự khác nhau giữa entropy chéo và entropy, có nghĩa là giá trị entropy chéo bổ sung phát sinh so với giá trị nhỏ nhất không thể giảm được mà nó có thể nhận:

$$D(p||q) = H(p,q) - H(p) = \sum_j p(j)\log\frac{p(j)}{q(j)}$$

Lưu ý rằng trong bài toán phân loại, ta không biết giá trị thật của p, vì thế mà ta không thể tính toán entropy trực tiếp được. Tuy nhiên, bởi vì entropy nằm ngoài tầm kiểm soát của chúng ta, việc giảm thiểu $D(p||q)$ so với q là tương đương với việc giảm thiểu mất mát entropy chéo.

Tóm lại, chúng ta có thể nghĩ đến mục tiêu của phân loại entropy chéo theo hai hướng: (i) cực đại hóa khả năng xảy ra của dữ liệu được quan sát; và (ii) giảm thiểu sự ngạc nhiên của ta (cũng như số lượng các bit) cần thiết để truyền đạt các nhãn.

3.3.4.4. Sử dụng Mô hình để Dự đoán và Đánh giá

Sau khi huấn luyện mô hình hồi quy softmax với các đặc trưng đầu vào bất kì, chúng ta có thể dự đoán xác suất đầu ra ứng với mỗi lớp. Thông thường, chúng ta sử dụng lớp với xác suất dự đoán cao nhất làm lớp đầu ra. Một dự đoán được xem là chính xác nếu nó trùng khớp hay tương thích với lớp thật sự (nhãn). Ở phần tiếp theo, chúng ta sẽ sử dụng độ chính xác để đánh giá chất lượng của mô hình. Giá trị này là tỉ lệ giữa số mẫu được dự đoán chính xác so với tổng số mẫu được dự đoán.

3.3.4.5. Kết luận

- Chúng tôi đã giới thiệu về hàm softmax giúp ánh xạ một vector đầu vào sang các giá trị xác suất.

- Hồi quy softmax được áp dụng cho các bài toán phân loại. Nó sử dụng phân phối xác suất của các lớp đầu ra thông qua hàm softmax.

- Entropy chéo là một phép đánh giá tốt cho sự khác nhau giữa 2 phân phối xác suất. Nó đo lường số lượng bit cần để biểu diễn dữ liệu cho mô hình.

Chương 4

PERCEPTRON ĐA TẦNG (MULTILAYER PERCEPTRONS)

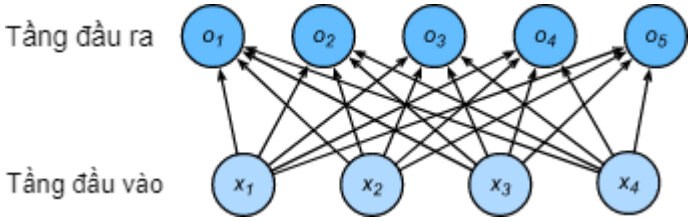
Trong chương trước, chúng tôi đã giới thiệu hồi quy softmax. Cùng với đó, chúng ta đã học cách xử lý dữ liệu, ép buộc các giá trị đầu ra tạo thành một phân phối xác suất hợp lệ (thông qua hàm softmax), áp dụng một hàm mất mát phù hợp và cực tiểu hoá nó theo các tham số mô hình. Bây giờ, sau khi đã thành thạo các cơ chế nêu trên trong ngữ cảnh của những mô hình tuyến tính đơn giản, chúng ta có thể bắt đầu khám phá trọng tâm của cuốn sách này: lớp mô hình phong phú của các mạng nơ-ron sâu.

4.1. Tổng quan Perceptron đa tầng

4.1.1. Các tầng ẩn

Để bắt đầu, hãy nhớ lại kiến trúc mô hình trong ví dụ của hồi quy softmax, được minh hoạ trong Hình 4.1 bên dưới. Mô hình này ánh xạ trực tiếp các đầu vào của chúng ta sang các giá trị đầu ra thông qua một phép biến đổi tuyến tính duy nhất:

$$\hat{o} = \text{softmax}(\mathbf{W}x + b)$$



Hình 4.1. Perceptron đơn tầng với 5 nút đầu ra.

Nếu các nhãn của chúng ta thật sự có mối quan hệ tuyến tính với dữ liệu đầu vào, thì cách tiếp cận này là đủ. Nhưng tính tuyến tính là một giả định chặt.

Ví dụ, tính tuyến tính ngụ ý về giả định yếu hơn của tính đơn điệu: tức giá trị đặc trưng tăng luôn dẫn đến việc đầu ra mô hình tăng (nếu trọng số tương ứng dương), hoặc đầu ra mô hình giảm (nếu trọng số tương ứng âm). Điều này đôi khi cũng hợp lý. Ví dụ, nếu chúng ta đang dự đoán liệu một người có trả được khoản vay hay không, chúng ta có thể suy diễn một cách hợp lý như sau: bỏ qua mọi yếu tố khác, ứng viên nào có thu nhập cao hơn sẽ có khả năng trả được nợ cao hơn so với những ứng viên khác có thu nhập thấp hơn. Dù có tính đơn điệu, mối quan hệ này khả năng cao là không liên quan tuyến tính tới xác suất trả nợ. Khả năng trả được nợ thường sẽ có mức tăng lớn hơn khi thu nhập tăng từ \$0 lên \$50k so với khi tăng từ \$1M lên \$1.05M. Một cách để giải quyết điều này là tiền xử lý dữ liệu để tính tuyến tính trở nên hợp lý hơn, ví dụ như sử dụng logarit của thu nhập để làm đặc trưng.

Lưu ý rằng chúng ta có thể dễ dàng đưa ra các ví dụ vi phạm tính đơn điệu. Ví dụ như, ta muốn dự đoán xác suất tử vong của một người dựa trên thân nhiệt. Đối với người có thân nhiệt trên 37°C (98.6°F), nhiệt độ càng cao gây ra nguy cơ tử vong càng cao. Tuy nhiên, với những người có thân nhiệt thấp hơn 37°C, khi gặp nhiệt độ cao hơn thì nguy cơ tử vong lại thấp hơn! Trong bài toán này, ta có thể giải quyết nó bằng một vài bước tiền xử lý thật khéo léo. Cụ thể, ta có thể sử dụng khoảng cách từ 37°C tới thân nhiệt làm đặc trưng.

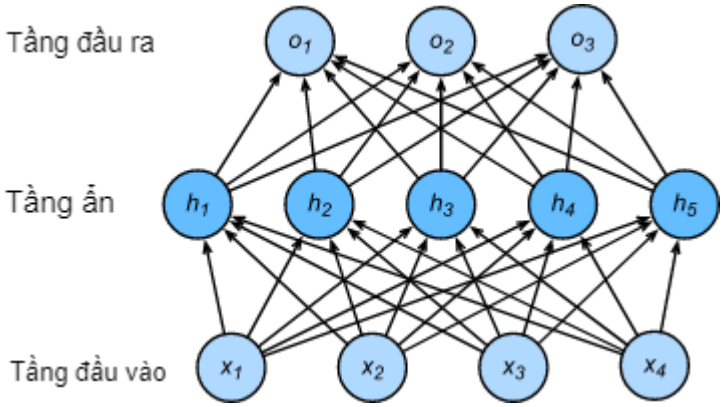
Nhưng còn với bài toán phân loại hình ảnh chó mèo thì sao? Liệu việc tăng cường độ sáng của điểm ảnh tại vị trí (13, 17) sẽ luôn tăng (hoặc giảm) khả năng đó là hình một con chó? Sử dụng mô hình tuyến tính trong trường hợp này tương ứng với việc ngầm giả định rằng chỉ cần

đánh giá độ sáng của từng pixel để phân biệt giữa mèo và chó. Cách tiếp cận này chắc chắn sẽ không chính xác khi các hình ảnh bị đảo ngược màu sắc.

Tuy nhiên, ta bỏ qua sự phi lý của tuyến tính ở đây, so với các ví dụ trước, rõ ràng là ta không thể giải quyết bài toán này với vài bước tiền xử lý chỉnh sửa đơn giản. Bởi vì ý nghĩa của các điểm ảnh phụ thuộc một cách phức tạp vào bối cảnh xung quanh nó (các giá trị xung quanh của điểm ảnh). Có thể vẫn tồn tại một cách biểu diễn dữ liệu nào đó nắm bắt được sự tương tác giữa các đặc trưng liên quan (và quan trọng nhất là phù hợp với mô hình tuyến tính), ta đơn giản là không biết làm thế nào để tính toán nó một cách thủ công. Với các mạng nơ-ron sâu, ta sử dụng dữ liệu đã quan sát được để đồng thời học cách biểu diễn (thông qua các tầng ẩn) và học một bộ dự đoán tuyến tính hoạt động dựa trên biểu diễn đó.

4.1.1.1. Kết hợp các Tầng ẩn

Ta có thể vượt qua những hạn chế của mô hình tuyến tính và làm việc với một lớp hàm tổng quát hơn bằng cách thêm vào một hoặc nhiều tầng ẩn. Cách dễ nhất để làm điều này là xếp chồng nhiều tầng kết nối đầy đủ lên nhau. Giá trị đầu ra của mỗi tầng được đưa làm giá trị đầu vào cho tầng bên trên, cho đến khi ta tạo được một đầu ra. Ta có thể xem L−1 tầng đầu tiên như các tầng học biểu diễn dữ liệu và tầng cuối cùng là bộ dự đoán tuyến tính. Kiến trúc này thường được gọi là perceptron đa tầng (MultiLayer Perceptron), hay được viết tắt là MLP. Dưới đây, ta mô tả sơ đồ MLP (Hình. 4.2).



Hình 4.2. Perceptron đa tầng với các tầng ẩn. Ví dụ này chứa một tầng ẩn với 5 nút ẩn.

Perceptron đa tầng này có 4 đầu vào, 3 đầu ra và tầng ẩn của nó chứa 5 nút ẩn. Vì tầng đầu vào không cần bất kỳ tính toán nào, do đó đối với mạng này để tạo đầu ra đòi hỏi phải lập trình các phép tính cho hai tầng còn lại (tầng ẩn và tầng đầu ra). Lưu ý, tất cả tầng này đều kết nối đầy đủ. Mỗi đầu vào đều ảnh hưởng đến mọi nơ-ron trong tầng ẩn và mỗi nơ-ron này lại ảnh hưởng đến mọi nơ-ron trong tầng đầu ra.

4.1.1.2. Từ Tuyến tính đến Phi tuyến

Về mặt hình thức, chúng ta tính toán mỗi tầng trong MLP một-tầng-ẩn này như sau:

$$\begin{aligned}\mathbf{H} &= \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)},\\ \mathbf{O} &= \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}.\end{aligned}$$

Chú ý rằng sau khi thêm tầng này vào, mô hình lập tức yêu cầu chúng ta phải theo dõi và cập nhật thêm hai tập tham số. Vậy thì đổi lại ta sẽ nhận được gì? Bạn có thể bất ngờ khi phát hiện ra rằng—trong mô hình định nghĩa bên trên—chúng ta chẳng thu được lợi ích gì từ những rắc rối thêm vào! Lý do rất đơn giản. Các nút ẩn bên trên được định nghĩa bởi một hàm tuyến tính của các đầu vào, và các đầu ra (tiền Softmax) chỉ là một hàm tuyến tính của các nút ẩn. Một hàm tuyến

tính của một hàm tuyến tính bản thân nó cũng chính là một hàm tuyến tính. Hơn nữa, mô hình tuyến tính của chúng ta vốn dĩ đã có khả năng biểu diễn bất kỳ hàm tuyến tính nào rồi.

Ta có thể thấy sự tương đồng về mặt hình thức bằng cách chứng minh rằng với mọi giá trị của các trọng số, ta đều có thể loại bỏ tầng ẩn và tạo ra một mô hình đơn-tầng với các tham số

$$\mathbf{W} = \mathbf{W}^{(1)}\mathbf{W}^{(2)}$$

$$\mathbf{b} = \mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$

$$\mathbf{O} = (\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\mathbf{W}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\mathbf{W} + \mathbf{b}.$$

Để hiện thực được tiềm năng của các kiến trúc đa tầng, chúng ta cần một thành phần quan trọng nữa—một hàm kích hoạt phi tuyến theo từng phần tử σ để áp dụng lên từng nút ẩn (theo sau phép biến đổi tuyến tính). Hiện nay, lựa chọn phổ biến nhất cho tính phi tuyến là đơn vị tuyến tính chỉnh lưu (ReLU)max(x,0). Nhìn chung, với việc sử dụng các hàm kích hoạt này, chúng ta sẽ không thể biến MLP thành một mô hình tuyến tính được nữa.

$$\begin{aligned}\mathbf{H} &= \sigma(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}),\\ \mathbf{O} &= \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}.\end{aligned}$$

Để xây dựng các MLP tổng quan hơn, chúng ta có thể tiếp tục chồng thêm các tầng ẩn, ví dụ, $\mathbf{H}^{(1)} = \sigma_1(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})$ và $\mathbf{H}^{(2)} = \sigma_2(\mathbf{H}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)})$, kế tiếp nhau, tạo ra các mô hình có khả năng biểu diễn càng cao (giả sử chiều rộng cố định).

Các MLP có thể biểu diễn được những tương tác phức tạp giữa các đầu vào thông qua các nơ-ron ẩn, các nơ-ron ẩn này phụ thuộc vào giá trị của mỗi đầu vào. Chúng ta có thể dễ dàng thiết kế các nút ẩn để thực hiện bất kỳ tính toán nào, ví dụ, các phép tính logic cơ bản trên một cặp đầu vào. Ngoài ra, với một số hàm kích hoạt cụ thể, các MLP được biết đến rộng rãi như là các bộ xấp xỉ vạn năng. Thậm chí với một mạng chỉ có một tầng ẩn, nếu có đủ số nút (có thể nhiều một cách vô lý) và một tập các trọng số thích hợp, chúng ta có thể mô phỏng bất kỳ một hàm nào. Thật ra thì việc học được hàm đó mới là phần khó khăn. Bạn có thể tưởng tượng mạng nơ-ron của mình có nét giống với ngôn ngữ lập trình C. Ngôn ngữ này giống như bất kỳ ngôn ngữ hiện đại nào khác, có khả năng biểu diễn bất kỳ chương trình tính toán nào. Tuy nhiên việc tạo ra một chương trình đáp ứng được các các chỉ tiêu kỹ thuật mới là phần việc khó khăn.

Hơn nữa, chỉ vì một mạng đơn-tầng có thể học bất kỳ hàm nào không có nghĩa rằng bạn nên cố gắng giải quyết tất cả các vấn đề của mình bằng các mạng đơn-tầng. Thực tế, chúng ta có thể ước lượng các hàm một cách gọn gàng hơn rất nhiều bằng cách sử dụng mạng sâu hơn (thay vì rộng hơn). Chúng ta sẽ đề cập đến những lập luận chặt chẽ hơn trong các chương tiếp theo, nhưng trước tiên hãy lập trình một MLP. Trong ví dụ này, chúng ta lập trình một MLP với hai tầng ẩn và một tầng đầu ra.

4.1.1.3. Vector hoá và Minibatch

Giống như trước, chúng ta dùng ma trận $\mathbf{X}$ để ký hiệu một minibatch các giá trị đầu vào. Các phép tính toán dẫn đến các giá trị đầu ra từ một MLP với hai tầng ẩn khi đó có thể được biểu diễn như sau:

$$\mathbf{H}^{(1)} = \sigma_1\left(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}\right)$$

$$\mathbf{H}^{(2)} = \sigma_2(\mathbf{H}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)})$$

$$\boldsymbol{O} = \text{softmax}(\mathbf{H}^{(2)}\mathbf{W}^{(3)} + \mathbf{b}^{(3)})$$

Bằng việc lạm dụng ký hiệu một chút, chúng ta định nghĩa hàm phi tuyến σ là một phép toán áp dụng theo từng hàng, tức lần lượt từng điểm dữ liệu một. Cần chú ý rằng ta cũng sử dụng quy ước này cho hàm softmax để ký hiệu toán tử tính theo từng hàng. Thông thường, như trong mục này, các hàm kích hoạt không chỉ đơn thuần được áp dụng vào tầng ẩn theo từng hàng mà còn theo từng phần tử. Điều đó có nghĩa là sau khi tính toán xong phần tuyến tính của tầng, chúng ta có thể tính giá trị kích hoạt của từng nút mà không cần đến giá trị của các nút còn lại. Điều này cũng đúng đối với hầu hết các hàm kích hoạt.

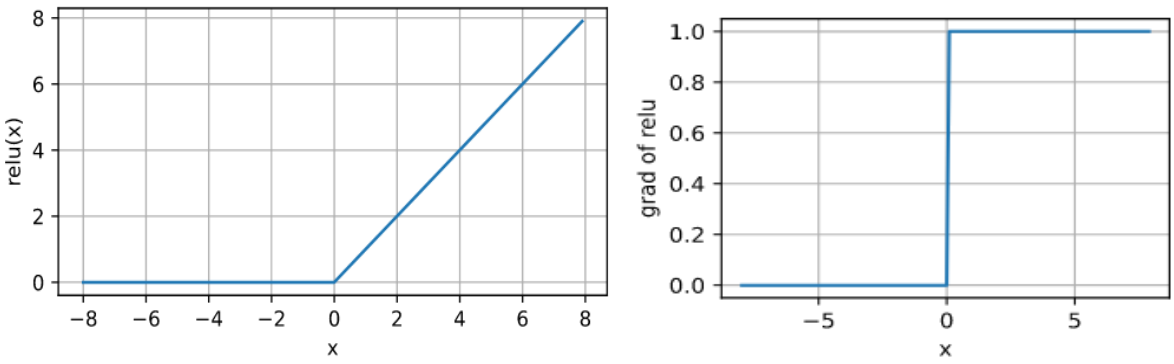
4.1.2. Các hàm Kích hoạt

Các hàm kích hoạt quyết định một nơ-ron có được kích hoạt hay không bằng cách tính tổng có trọng số và cộng thêm hệ số điều chỉnh vào nó. Chúng là các toán tử khả vi và hầu hết đều biến đổi các tín hiệu đầu vào thành các tín hiệu đầu ra theo một cách phi tuyến tính. Bởi vì các hàm kích hoạt rất quan trọng trong học sâu, hãy cùng tìm hiểu sơ lược một số hàm kích hoạt thông dụng.

4.1.2.1. Hàm ReLU

Như đã đề cập trước đó, đơn vị tuyến tính chỉnh lưu (ReLU) là sự lựa chọn phổ biến nhất do tính đơn giản khi lập trình và hiệu quả trong nhiều tác vụ dự đoán. ReLU là một phép biến đổi phi tuyến đơn giản. Cho trước một phần tử x, ta định nghĩa hàm ReLU là giá trị lớn nhất giữa chính phần tử đó và 0.

$$\text{ReLU}(x) = max(x, 0).$$



Hình 4.3. Đồ thị hàm ReLU và đạo hàm

Khi đầu vào mang giá trị âm thì đạo hàm của hàm ReLu bằng 0 và khi đầu vào mang giá trị dương thì đạo hàm của hàm ReLu bằng 1. Lưu ý rằng, hàm ReLU không khả vi tại 0. Trong thường hợp này, ta mặc định lấy đạo hàm trái (left-hand-side – LHS) và nói rằng đạo hàm của hàm ReLU tại 0 thì bằng 0. Chỗ này có thể du di được vì đầu vào thông thường không có giá trị chính xác bằng không. Có một ngạn ngữ xưa nói rằng, nếu ta quan tâm nhiều đến điều kiện biên thì có lẽ ta chỉ đang làm toán (thuần túy), chứ không phải đang làm kỹ thuật. Và trong trường hợp này, ngạn ngữ đó đúng. Đồ thị đạo hàm của hàm ReLU như hình dưới.

Lưu ý rằng, có nhiều biến thể của hàm ReLU, bao gồm ReLU được tham số hóa (pReLU) của He et al., 2015. Phiên bản này thêm một thành phần tuyến tính vào ReLU, do đó một số thông tin vẫn được giữ lại ngay cả khi đối số là âm.

$$\text{pReLU}(x) = max(0, x) + \alpha min(0, x).$$

Ta sử dụng hàm ReLU bởi vì đạo hàm của nó khá đơn giản: hoặc là chúng biến mất hoặc là chúng cho đối số đi qua. Điều này làm cho việc tối ưu trở nên tốt hơn và giảm thiểu được nhược

điểm tiêu biến gradient đã từng gây khó khăn trong các phiên bản trước của mạng nơ-ron (sẽ được đề cập lại sau này).

4.1.2.2. Hàm Sigmoid

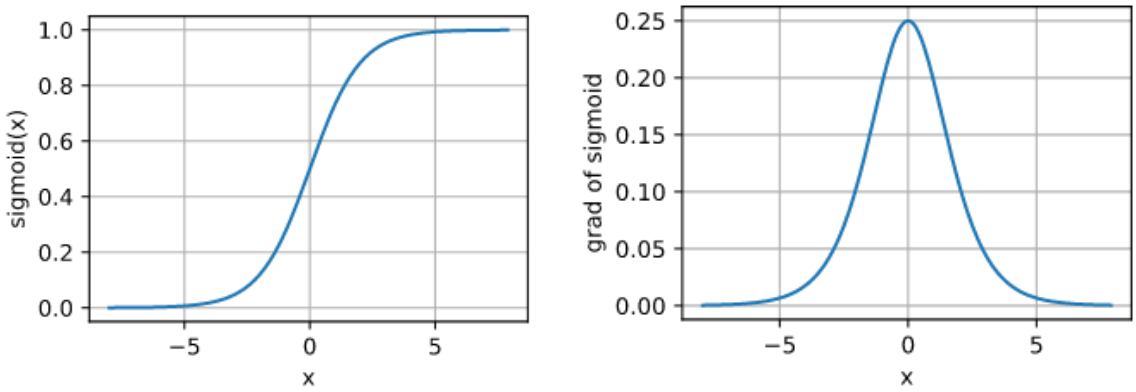
Hàm sigmoid biến đổi các giá trị đầu vào có miền giá trị thuộc R thành các giá trị đầu ra nằm trong khoảng (0,1). Vì vậy, hàm sigmoid thường được gọi là hàm ép: nó ép một giá trị đầu vào bất kỳ nằm trong khoảng $(-\infty, \infty)$ thành một giá trị đầu ra nằm trong khoảng (0, 1).

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}.$$

Các nơ-ron sinh học mà có thể ở một trong hai trạng thái kích hoạt hoặc không kích hoạt, là một chủ đề mô hình hoá rất được quan tâm từ những nghiên cứu đầu tiên về mạng nơ-ron. Vì vậy mà những người tiên phong trong lĩnh vực này, bao gồm McCulloch và Pitts, những người phát minh ra nơ-ron nhân tạo, đã tập trung nghiên cứu về các đơn vị ngưỡng. Một kích hoạt ngưỡng có giá trị là 0 khi đầu vào của nó ở dưới mức ngưỡng và giá trị là 1 khi đầu vào vượt mức ngưỡng đó.

Khi phương pháp học dựa trên gradient trở nên phổ biến, hàm sigmoid là một lựa chọn tất yếu của đơn vị ngưỡng bởi tính liên tục và khả vi của nó. Hàm sigmoid vẫn là hàm kích hoạt được sử dụng rộng rãi ở các đơn vị đầu ra, khi ta muốn biểu diễn kết quả đầu ra như là các xác suất của bài toán phân loại nhị phân (bạn có thể xem sigmoid như một trường hợp đặc biệt của softmax). Tuy nhiên, trong các tầng ẩn, hàm sigmoid hầu hết bị thay thế bằng hàm ReLU vì nó đơn giản hơn và giúp cho việc huấn luyện trở nên dễ dàng hơn. Trong chương "Mạng nơ-ron hồi tiếp", chúng tôi sẽ mô tả các mô hình sử dụng đơn vị sigmoid để kiểm soát luồng thông tin theo thời gian.

Dưới đây, ta vẽ đồ thị hàm sigmoid. Cần chú ý rằng, khi đầu vào có giá trị gần bằng 0, hàm sigmoid tiến tới một phép biến đổi tuyến tính.



Hình 4.4. Đồ thị hàm sigmoid và đạo hàm

Đạo hàm của hàm sigmoid được tính bởi phương trình sau:

$$\frac{d}{dx}\text{sigmoid}(x) = \frac{\exp(-x)}{(1 + \exp(-x))^2} = \text{sigmoid}(x)(1 - \text{sigmoid}(x)).$$

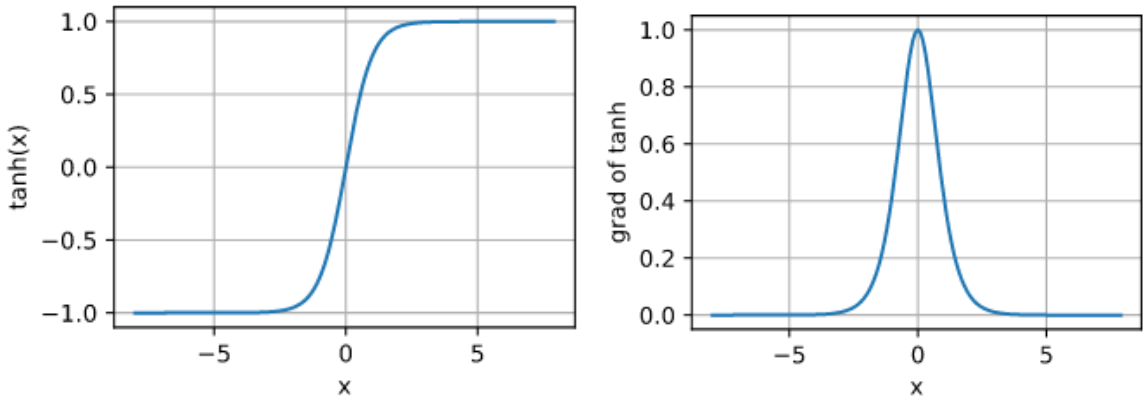
Đồ thị đạo hàm của hàm sigmoid được vẽ ở dưới. Chú ý rằng khi đầu vào là 0, đạo hàm của hàm sigmoid đạt giá trị lớn nhất là 0.25. Khi đầu vào phân kỳ từ 0 theo một trong hai hướng, đạo hàm sẽ tiến tới 0.

4.1.2.3. Hàm “Tanh”

Tương tự như hàm sigmoid, hàm tanh (Hyperbolic Tangent) cũng ép các biến đầu vào và biến đổi chúng thành các phần tử nằm trong khoảng -1 và 1:

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}.$$

Chúng ta sẽ vẽ hàm tanh như sau. Chú ý rằng nếu đầu vào có giá trị gần bằng 0, hàm tanh sẽ tiến đến một phép biến đổi tuyến tính. Mặc dù hình dạng của hàm tanh trông khá giống hàm sigmoid, hàm tanh lại thể hiện tính đối xứng tâm qua gốc của hệ trục tọa độ.



Hình 4.5. Đồ thị hàm Tanh và đạo hàm

Đạo hàm của hàm Tanh là:

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x).$$

Đạo hàm của hàm tanh được vẽ như sau. Khi đầu vào có giá trị gần bằng 0, đạo hàm của hàm tanh tiến tới giá trị lớn nhất là 1. Tương tự như hàm sigmoid, khi đầu vào phân kỳ từ 0 theo bất kỳ hướng nào, đạo hàm của hàm tanh sẽ tiến đến 0.

Tóm lại, bây giờ chúng ta đã biết cách kết hợp các hàm phi tuyến để xây dựng các kiến trúc mạng nơ-ron đa tầng mạnh mẽ. Một lưu ý bên lề đó là, kiến thức của bạn bây giờ cung cấp cho bạn cách sử dụng một bộ công cụ tương đương với của một người có chuyên môn về học sâu vào những năm 1990. Xét theo một khía cạnh nào đó, bạn còn có lợi thế hơn bất kỳ ai làm việc trong những năm 1990, bởi vì bạn có thể tận dụng triệt để các framework học sâu nguồn mở để xây dựng các mô hình một cách nhanh chóng, chỉ với một vài dòng mã. Trước đây, việc huấn luyện các mạng nơ-ron đòi hỏi các nhà nghiên cứu phải viết đến hàng ngàn dòng mã C và Fortran.

4.1.3. Kết luận

- Perceptron đa tầng sẽ thêm một hoặc nhiều tầng ẩn được kết nối đầy đủ giữa các tầng đầu ra và các tầng đầu vào nhằm biến đổi đầu ra của tầng ẩn thông qua hàm kích hoạt.

- Các hàm kích hoạt thường được sử dụng bao gồm hàm ReLU, hàm sigmoid, và hàm tanh.

4.2. Lựa Chọn Mô Hình, Dưới Khớp và Quá Khớp

Là những nhà khoa học học máy, mục tiêu của chúng ta là khám phá ra các khuôn mẫu. Nhưng làm sao có thể chắc chắn rằng chúng ta đã thực sự khám phá ra một khuôn mẫu khái quát chứ không chỉ đơn giản là ghi nhớ dữ liệu. Ví dụ, thử tưởng tượng rằng chúng ta muốn săn lùng

các khuôn mẫu liên kết các dấu hiệu di truyền của bệnh nhân và tình trạng mất trí của họ, với nhãn được trích ra từ tập {mất trí nhớ, suy giảm nhận thức mức độ nhẹ, khỏe mạnh}. Bởi vì các gen của mỗi người định dạng họ theo cách độc nhất vô nhị (bỏ qua các cặp song sinh giống hệt nhau), nên việc ghi nhớ toàn bộ tập dữ liệu là hoàn toàn khả thi.

Chúng ta không muốn mô hình của mình nói rằng "Bob kìa! Tôi nhớ anh ta! Anh ta bị mất trí nhớ! Lý do tại sao rất đơn giản. Khi triển khai mô hình trong tương lai, chúng ta sẽ gặp các bệnh nhân mà mô hình chưa bao giờ gặp trước đó. Các dự đoán sẽ chỉ có ích khi mô hình của chúng ta thực sự khám phá ra một khuôn mẫu khái quát.

Để tóm tắt một cách chính thức hơn, mục tiêu của chúng ta là khám phá các khuôn mẫu mà chúng mô tả được các quy tắc trong tập dữ liệu mà từ đó tập huấn luyện đã được trích ra. Nếu thành công trong nỗ lực này, thì chúng ta có thể đánh giá thành công rủi ro ngay cả đối với các cá nhân mà chúng ta chưa bao giờ gặp phải trước đây. Vấn đề này—làm cách nào để khám phá ra các mẫu mà khái quát hóa—là vấn đề nền tảng của học máy.

Nguy hiểm là khi huấn luyện các mô hình, chúng ta chỉ truy cập một tập dữ liệu nhỏ. Các tập dữ liệu hình ảnh công khai lớn nhất chứa khoảng một triệu ảnh. Thường thì chúng ta phải học chỉ từ vài ngàn hoặc vài chục ngàn điểm dữ liệu. Trong một hệ thống bệnh viện lớn, chúng ta có thể truy cập hàng trăm ngàn hồ sơ y tế. Khi làm việc với các tập mẫu hữu hạn, chúng ta gặp phải rủi ro sẽ khám phá ra các mối liên kết rõ ràng mà hóa ra lại không đúng khi thu thập thêm dữ liệu.

Hiện tượng mô hình khớp với dữ liệu huấn luyện chính xác hơn nhiều so với phân phối thực được gọi là quá khớp (overfitting), và kỹ thuật sử dụng để chống lại quá khớp được gọi là điều chuẩn (regularization). Trong các phần trước, bạn có thể đã quan sát hiệu ứng này khi thử nghiệm với tập dữ liệu Fashion-MNIST. Nếu bạn đã sửa đổi cấu trúc mô hình hoặc siêu tham số trong quá trình thử nghiệm, bạn có thể đã nhận ra rằng với đủ các nút, các tầng, và các epoch huấn luyện, mô hình ấy có thể cuối cùng cũng đạt đến sự chính xác hoàn hảo trên tập huấn luyện, ngay cả khi độ chính xác trên dữ liệu kiểm tra giảm đi.

4.2.1. Lỗi huấn luyện và Lỗi khái quát

Để thảo luận hiện tượng này một cách chuyên sâu hơn, ta cần phân biệt giữa lỗi huấn luyện (training error) và lỗi khái quát (generalization error). Lỗi huấn luyện là lỗi của mô hình được tính toán trên tập huấn luyện, trong khi đó lỗi khái quát là lỗi kỳ vọng của mô hình khi áp dụng nó cho một luồng vô hạn các điểm dữ liệu mới được lấy từ cùng một phân phối dữ liệu với các mẫu ban đầu.

Vấn đề là chúng ta không bao giờ có thể tính toán chính xác lỗi khái quát vì luồng vô hạn dữ liệu chỉ có trong tưởng tượng. Trên thực tế, ta phải ước tính lỗi khái quát bằng cách áp dụng mô hình vào một tập kiểm tra độc lập bao gồm các điểm dữ liệu ngẫu nhiên ngoài tập huấn luyện.

Ba thí nghiệm sau sẽ giúp minh họa tình huống này tốt hơn. Hãy xem xét một sinh viên đại học đang cố gắng chuẩn bị cho kỳ thi cuối cùng của mình. Một sinh viên chăm chỉ sẽ cố gắng luyện tập tốt và kiểm tra khả năng của cô ấy bằng việc luyện tập những bài kiểm tra của các năm trước. Tuy nhiên, làm tốt các bài kiểm tra trước đây không đảm bảo rằng cô ấy sẽ làm tốt bài kiểm tra thật. Ví dụ, sinh viên có thể cố gắng chuẩn bị bằng cách học tủ các câu trả lời cho các câu hỏi. Điều này đòi hỏi sinh viên phải ghi nhớ rất nhiều thứ. Cô ấy có lẽ còn ghi nhớ đáp án cho các bài kiểm tra cũ một cách hoàn hảo. Một học sinh khác có thể chuẩn bị bằng việc cố gắng hiểu lý do mà một số đáp án nhất định được đưa ra. Trong hầu hết các trường hợp, sinh viên sau sẽ làm tốt hơn nhiều.

Tương tự như vậy, hãy xem xét một mô hình đơn giản chỉ sử dụng một bảng tra cứu để trả lời các câu hỏi. Nếu tập hợp các đầu vào cho phép là rời rạc và đủ nhỏ, thì có lẽ sau khi xem nhiều ví dụ huấn luyện, phương pháp này sẽ hoạt động tốt. Tuy nhiên mô hình này không có khả năng thể hiện tốt hơn so với việc đoán ngẫu nhiên khi phải đối mặt với các ví dụ chưa từng gặp trước đây. Trong thực tế, không gian đầu vào là quá lớn để có thể ghi nhớ mọi đáp án tương ứng của từng đầu vào khả dĩ. Ví dụ, hãy xem xét các ảnh 28×28 đen trắng. Nếu mỗi điểm ảnh có thể lấy một trong số các giá trị xám trong thang 256, thì có thể có 256784 ảnh khác nhau. Điều đó nghĩa là số lượng ảnh độ phân giải thấp còn lớn hơn nhiều so với số lượng nguyên tử trong vũ trụ. Thậm chí nếu có thể xem qua toàn bộ điểm dữ liệu, ta cũng không thể lưu trữ chúng trong bảng tra cứu.

Cuối cùng, hãy xem xét bài toán phân loại kết quả của việc tung đồng xu (lớp 0: ngửa, lớp 1: xấp) dựa trên một số đặc trưng theo ngữ cảnh sẵn có. Bất kể thuật toán nào được đưa ra, lỗi khái quát sẽ luôn là 12. Tuy nhiên, đối với hầu hết các thuật toán, lỗi huấn luyện sẽ thấp hơn đáng kể, tùy thuộc vào sự may mắn của ta khi lấy dữ liệu, ngay cả khi ta không có bất kỳ đặc trưng nào! Hãy xem xét tập dữ liệu {0, 1, 1, 1, 0, 1}. Việc không có đặc trưng có thể khiến ta luôn dự đoán lớp chiếm đa số, đối với ví dụ này thì đó là 1. Trong trường hợp này, mô hình luôn dự đoán lớp 1 sẽ có lỗi huấn luyện là 13, tốt hơn đáng kể so với lỗi khái quát. Khi ta tăng lượng dữ liệu, xác suất nhận được mặt ngửa sẽ dần tiến về 12 và lỗi huấn luyện sẽ tiến đến lỗi khái quát.

4.2.1.1. Lý thuyết Học Thống kê

Bởi khái quát hóa là một vấn đề nền tảng trong học máy, không quá ngạc nhiên khi nhiều nhà toán học và nhà lý thuyết học dành cả cuộc đời để phát triển các lý thuyết hình thức mô tả vấn đề này. Trong định lý cùng tên, Glivenko và Cantelli đã tìm ra tốc độ học mà tại đó lỗi huấn luyện sẽ hội tụ về lỗi khái quát. Trong chuỗi các bài báo đầu ngành, Vapnik và Chervonenkis đã mở rộng lý thuyết này cho nhiều lớp hàm tổng quát hơn. Công trình này là nền tảng của ngành Lý thuyết học thống kê.

Trong một thiết lập chuẩn cho học có giám sát – chủ đề lớn nhất xuyên suốt cuốn sách, chúng ta giả sử rằng cả dữ liệu huấn luyện và dữ liệu kiểm tra đều được lấy mẫu độc lập từ các phân phối giống hệt nhau (independent & identically distributed, thường gọi là giả thiết i.i.d.). Điều này có nghĩa là quá trình lấy mẫu dữ liệu không hề có sự ghi nhớ. Mẫu lấy ra thứ hai cũng không tương quan với mẫu thứ ba hơn so với mẫu thứ hai triệu.

Trở thành một nhà khoa học học máy giỏi yêu cầu tư duy phản biện, và có lẽ bạn đã có thể "bóc mẽ" được giả thiết này, có thể đưa ra các tình huống thường gặp mà giả thiết này không thỏa mãn. Điều gì sẽ xảy ra nếu chúng ta huấn luyện một mô hình dự đoán tỉ lệ tử vong trên bộ dữ thu thập từ các bệnh nhân tại UCSF, và áp dụng nó trên các bệnh nhân tại Bệnh viện Đa khoa Massachusetts. Các phân phối này đơn giản là không giống nhau. Hơn nữa, việc lấy mẫu có thể có tương quan về mặt thời gian. Sẽ ra sao nếu chúng ta thực hiện phân loại chủ đề cho các bài Tweet. Vòng đời của các tin tức sẽ tạo nên sự phụ thuộc về mặt thời gian giữa các chủ đề được đề cập, vi phạm mọi giả định độc lập thống kê.

Đôi khi, chúng ta có thể bỏ qua một vài vi phạm nhỏ trong giả thiết i.i.d. mà mô hình vẫn có thể làm việc rất tốt. Nhìn chung, gần như tất cả các ứng dụng thực tế đều vi phạm một vài giả thiết i.i.d. nhỏ, nhưng đổi lại ta có được các công cụ rất hữu dụng như nhận dạng khuôn mặt, nhận dạng tiếng nói, dịch ngôn ngữ, v.v.

Các vi phạm khác thì chắc chắn dẫn tới rắc rối. Cùng hình dung ở ví dụ này, ta thử huấn luyện một hệ thống nhận dạng khuôn mặt sử dụng hoàn toàn dữ liệu của các sinh viên đại học và đem đi triển khai như một công cụ giám sát trong viện dưỡng lão. Cách này gần như không khả thi vì ngoại hình giữa hai độ tuổi quá khác biệt.

Trong các mục và chương kế tiếp, chúng ta sẽ đề cập tới các vấn đề gặp phải khi vi phạm giả thiết i.i.d. Hiện tại khi giả thiết i.i.d. thậm chí được đảm bảo, hiểu được sự khái quát hóa cũng là một vấn đề nan giải. Hơn nữa, việc làm sáng tỏ nền tảng lý thuyết để giải thích tại sao các mạng nơ-ron sâu có thể khái quát hóa tốt như vậy vẫn tiếp tục làm đau đầu những bộ óc vĩ đại nhất trong lý thuyết học.

Khi huấn luyện mô hình, ta đang cố gắng tìm kiếm một hàm số khớp với dữ liệu huấn luyện nhất có thể. Nếu hàm số này quá linh hoạt để có thể khớp với các khuôn mẫu giả cũng dễ như với các xu hướng thật trong dữ liệu, thì nó có thể quá khớp để có thể tạo ra một mô hình có tính khái quát hóa cao trên dữ liệu chưa nhìn thấy. Đây chính xác là những gì chúng ta muốn tránh (hay ít nhất là kiểm soát được). Rất nhiều kỹ thuật trong học sâu là các phương pháp dựa trên thực nghiệm và thủ thuật để chống lại vấn đề quá khớp.

4.2.1.2. Độ Phức tạp của Mô hình

Khi có các mô hình đơn giản và dữ liệu dồi dào, ta kỳ vọng lỗi khái quát sẽ giống với lỗi huấn luyện. Khi làm việc với mô hình phức tạp hơn và ít mẫu huấn luyện hơn, ta kỳ vọng các lỗi huấn luyện giảm xuống nhưng khoảng cách khái quát tăng. Việc chỉ ra chính xác điều gì cấu thành nên độ phức tạp của mô hình là một vấn đề nan giải. Có rất nhiều yếu tố ảnh hưởng đến việc một mô hình có khái quát hóa tốt hay không. Ví dụ một mô hình với nhiều tham số hơn sẽ được xem là phức tạp hơn. Một mô hình mà các tham số có miền giá trị rộng hơn thì được xem là phức tạp hơn. Thông thường với các mạng nơ-ron, ta nghĩ đến một mô hình có nhiều bước huấn luyện là mô hình phức tạp hơn, và mô hình dừng sớm là ít phức tạp hơn.

Rất khó để có thể so sánh sự phức tạp giữa các thành viên trong các lớp mô hình khác hẳn nhau (ví như cây quyết định so với mạng nơ-ron). Hiện tại, có một quy tắc đơn giản khá hữu ích sau: Một mô hình có thể giải thích các sự kiện bất kỳ thì được các nhà thống kê xem là phức tạp, trong khi một mô hình với năng lực biểu diễn giới hạn nhưng vẫn có thể giải thích tốt được dữ liệu thì hầu như chắc chắn là đúng đắn hơn. Trong triết học, điều này gần với tiêu chí của Popper về khả năng phủ định của một lý thuyết khoa học: một lý thuyết tốt nếu nó khớp dữ liệu và nếu có các kiểm định cụ thể có thể dùng để phản chứng nó. Điều này quan trọng bởi vì tất cả các ước lượng thống kê là post hoc, tức là ta đánh giá giả thuyết sau khi quan sát các sự thật, do đó dễ bị tác động bởi lỗi ngụy biện cùng tên. Từ bây giờ, ta sẽ đặt triết lý qua một bên và tập trung hơn vào các vấn đề hữu hình.

Trong phần này, để có cái nhìn trực quan, chúng ta sẽ tập trung vào một vài yếu tố có xu hướng ảnh hưởng đến tính khái quát của một lớp mô hình:

- Số lượng các tham số có thể điều chỉnh. Khi số lượng các tham số có thể điều chỉnh (đôi khi được gọi là bậc tự do) lớn thì mô hình sẽ dễ bị quá khớp hơn.
- Các giá trị được nhận bởi các tham số. Khi các trọng số có miền giá trị rộng hơn, các mô hình dễ bị quá khớp hơn.
- Số lượng các mẫu huấn luyện. Việc quá khớp một tập dữ liệu chứa chỉ một hoặc hai mẫu rất dễ dàng, kể cả khi mô hình đơn giản. Nhưng quá khớp một tập dữ liệu với vài triệu mẫu đòi hỏi mô hình phải cực kỳ linh hoạt.

4.2.2. Lựa chọn Mô hình

Trong học máy, ta thường lựa chọn mô hình cuối cùng sau khi cân nhắc nhiều mô hình ứng viên. Quá trình này được gọi là lựa chọn mô hình. Đôi khi các mô hình được đem ra so sánh khác nhau cơ bản về mặt bản chất (ví như, cây quyết định với các mô hình tuyến tính). Khi khác,

ta lại so sánh các thành viên của cùng một lớp mô hình được huấn luyện với các cài đặt siêu tham số khác nhau.

Lấy perceptron đa tầng làm ví dụ, ta mong muốn so sánh các mô hình với số lượng tầng ẩn khác nhau, số lượng nút ẩn khác nhau, và các lựa chọn hàm kích hoạt khác nhau áp dụng vào từng tầng ẩn. Để xác định được mô hình tốt nhất trong các mô hình ứng viên, ta thường sử dụng một tập kiểm định.

4.2.2.1. Tập Dữ liệu Kiểm định

Về nguyên tắc, ta không nên sử dụng tập kiểm tra cho đến khi chọn xong tất cả các siêu tham số. Nếu sử dụng dữ liệu kiểm tra trong quá trình lựa chọn mô hình, có một rủi ro là ta có thể quá khớp dữ liệu kiểm tra, và khi đó ta sẽ gặp rắc rối lớn. Nếu quá khớp dữ liệu huấn luyện, ta luôn có thể đánh giá mô hình trên tập kiểm tra để đảm bảo mình "trung thực". Nhưng nếu quá khớp trên dữ liệu kiểm tra, làm sao chúng ta có thể biết được?

Vì vậy, ta không bao giờ nên dựa vào dữ liệu kiểm tra để lựa chọn mô hình. Tuy nhiên, không thể chỉ dựa vào dữ liệu huấn luyện để lựa chọn mô hình vì ta không thể ước tính lỗi khái quát trên chính dữ liệu được sử dụng để huấn luyện mô hình.

Phương pháp phổ biến để giải quyết vấn đề này là phân chia dữ liệu thành ba phần, thêm một tập kiểm định ngoài các tập huấn luyện và kiểm tra.

Trong các ứng dụng thực tế, bức tranh trở nên mập mờ hơn. Mặc dù tốt nhất ta chỉ nên động đến dữ liệu kiểm tra đúng một lần, để đánh giá mô hình tốt nhất hoặc so sánh một số lượng nhỏ các mô hình với nhau, dữ liệu kiểm tra trong thế giới thực hiếm khi bị vứt bỏ chỉ sau một lần sử dụng. Ta hiếm khi có được một tập kiểm tra mới sau mỗi vòng thử nghiệm.

Kết quả là một thực tiễn âm u trong đó ranh giới giữa dữ liệu kiểm định và kiểm tra mơ hồ theo cách đáng lo ngại. Trừ khi có quy định rõ ràng thì, trong các thí nghiệm trong cuốn sách này, ta thật sự đang làm việc với cái được gọi là dữ liệu huấn luyện và dữ liệu kiểm định chứ không có tập kiểm tra thật. Do đó, độ chính xác được báo cáo trong mỗi thử nghiệm thật ra là độ chính xác kiểm định và không phải là độ chính xác của tập kiểm tra thật. Tin tốt là ta không cần quá nhiều dữ liệu trong tập kiểm định. Ta có thể chứng minh rằng sự bất định trong các ước tính thuộc bậc $\mathbf{O}(n^{-1/2})$.

4.2.2.2. Kiểm định chéo gập K-lần

Khi khan hiếm dữ liệu huấn luyện, có lẽ ta sẽ không thể dành ra đủ dữ liệu để tạo một tập kiểm định phù hợp. Một giải pháp phổ biến để giải quyết vấn đề này là kiểm định chéo gập K-lần. Ở phương pháp này, tập dữ liệu huấn luyện ban đầu được chia thành K tập con không chồng lên nhau. Sau đó việc huấn luyện và kiểm định mô hình được thực thi K lần, mỗi lần huấn luyện trên $K-1$ tập con và kiểm định trên tập con còn lại (tập không được sử dụng để huấn luyện trong lần đó). Cuối cùng, lỗi huấn luyện và lỗi kiểm định được ước lượng bằng cách tính trung bình các kết quả thu được từ K thí nghiệm.

4.2.3. Dưới khớp hay Quá khớp?

Khi so sánh lỗi huấn luyện và lỗi kiểm định, ta cần lưu ý hai trường hợp thường gặp sau: Đầu tiên, ta sẽ muốn chú ý trường hợp lỗi huấn luyện và lỗi kiểm định đều lớn nhưng khoảng cách giữa chúng lại nhỏ. Nếu mô hình không thể giảm thiểu lỗi huấn luyện, điều này có nghĩa là mô hình quá đơn giản (tức không đủ khả năng biểu diễn) để có thể xác định được khuôn mẫu mà ta đang cố mô hình hóa. Hơn nữa, do khoảng cách khái quát giữa lỗi huấn luyện và lỗi kiểm định

nhỏ, ta có lý do để tin rằng phương án giải quyết là một mô hình phức tạp hơn. Hiện tượng này được gọi là dưới khớp (underfitting).

Mặt khác, như ta đã thảo luận ở phía trên, ta cũng muốn chú ý tới trường hợp lỗi huấn luyện thấp hơn lỗi kiểm định một cách đáng kể, một biểu hiện của sự quá khớp nặng. Lưu ý rằng quá khớp không phải luôn là điều xấu. Đặc biệt là với học sâu, ta đều biết rằng mô hình dự đoán tốt nhất thường đạt chất lượng tốt hơn hẳn trên dữ liệu huấn luyện so với dữ liệu kiểm định. Sau cùng, ta thường quan tâm đến lỗi kiểm định hơn khoảng cách giữa lỗi huấn luyện và lỗi kiểm định.

Việc ta đang quá khớp hay dưới khớp có thể phụ thuộc vào cả độ phức tạp của mô hình lẫn kích thước của tập dữ liệu huấn luyện có sẵn, và hai vấn đề này sẽ được thảo luận ngay sau đây.

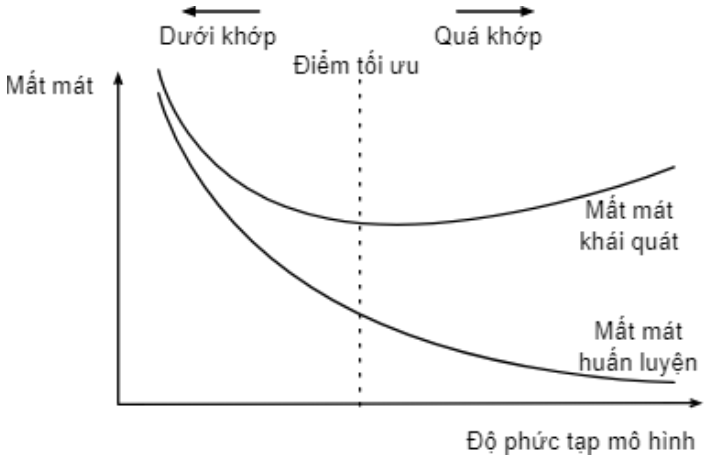
4.2.3.1. Độ phức tạp Mô hình

Để có thể hình dung một cách trực quan hơn về mối quan hệ giữa quá khớp và độ phức tạp mô hình, ta sẽ đưa ra một ví dụ sử dụng đa thức. Cho một tập dữ liệu huấn luyện có một đặc trưng duy nhất x và nhãn y tương ứng có giá trị thực, ta thử tìm bậc d của đa thức

$$\hat{y} = \sum_{i=0}^{d} x^i w_i$$

để ước tính nhãn y. Đây đơn giản là một bài toán hồi quy tuyến tính trong đó các đặc trưng được tính bằng cách lấy mũ của x, w_i là trọng số của mô hình, vì $x^0=1$ với mọi x nên w_0 là hệ số điều chỉnh. Vì đây là bài toán hồi quy tuyến tính, ta có thể sử dụng bình phương sai số làm hàm mất mát.

Hàm đa thức bậc cao phức tạp hơn hàm đa thức bậc thấp, vì đa thức bậc cao có nhiều tham số hơn và miền lựa chọn hàm số cũng rộng hơn. Nếu giữ nguyên tập dữ liệu huấn luyện, các hàm đa thức bậc cao hơn sẽ luôn đạt được lỗi huấn luyện thấp hơn (ít nhất là bằng) so với đa thức bậc thấp hơn. Trong thực tế, nếu mọi điểm dữ liệu có các giá trị x riêng biệt, một hàm đa thức có bậc bằng với số điểm dữ liệu đều có thể khớp một cách hoàn hảo với tập huấn luyện. Mối quan hệ giữa bậc của đa thức với hai hiện tượng dưới khớp và quá khớp được biểu diễn trong Hình 4.6.



Hình 4.6. Ảnh hưởng của Độ phức tạp Mô hình tới Dưới khớp và Quá khớp

4.2.3.2. Kích thước Tập dữ liệu

Một lưu ý quan trọng khác cần ghi nhớ là kích thước tập dữ liệu. Với một mô hình cố định, tập dữ liệu càng ít mẫu thì càng có nhiều khả năng gặp phải tình trạng quá khớp với mức độ

nghiêm trọng hơn. Khi số lượng dữ liệu tăng lên, lỗi khái quát sẽ có xu hướng giảm. Hơn nữa, trong hầu hết các trường hợp, nhiều dữ liệu không bao giờ là thừa. Trong một tác vụ với một phân phối dữ liệu cố định, ta có thể quan sát được mối quan hệ giữa độ phức tạp của mô hình và kích thước tập dữ liệu. Khi có nhiều dữ liệu, thử khớp một mô hình phức tạp hơn thường sẽ mang lợi nhiều lợi ích. Khi dữ liệu không quá nhiều, mô hình đơn giản sẽ là lựa chọn tốt hơn. Đối với nhiều tác vụ, học sâu chỉ tốt hơn các mô hình tuyến tính khi có sẵn hàng ngàn mẫu huấn luyện. Sự thành công hiện nay của học sâu phần nào dựa vào sự phong phú của các tập dữ liệu khổng lồ từ các công ty hoạt động trên internet, từ các thiết bị lưu trữ giá rẻ, các thiết bị được nối mạng và rộng hơn là việc số hóa nền kinh tế.

4.2.4. Hồi quy Đa thức

Bây giờ ta có thể khám phá một cách tương tác những khái niệm này bằng cách khớp đa thức với dữ liệu. Để bắt đầu ta sẽ nhập các gói thư viện thường dùng.

4.2.4.1. Tạo ra Tập dữ liệu

Đầu tiên ta cần dữ liệu. Cho x, ta sẽ sử dụng đa thức bậc ba ở dưới đây để tạo nhãn cho tập dữ liệu huấn luyện và tập kiểm tra:

$$y = 5 + 1.2x - 3.4\frac{x^2}{2!} + 5.6\frac{x^3}{3!} + \epsilon \text{ với } \epsilon \sim N(0, 0.1)$$

Số hạng nhiễu ϵ tuân theo phân phối chuẩn (phân phối Gauss) với giá trị trung bình bằng 0 và độ lệch chuẩn bằng 0.1. Ta sẽ tạo 100 mẫu cho mỗi tập huấn luyện và tập kiểm tra.

4.3. Suy giảm trọng số

Bởi chúng ta đã mô tả xong vấn đề quá khớp, giờ ta có thể tìm hiểu một vài kỹ thuật tiêu chuẩn trong việc điều chuẩn mô hình. Nhắc lại rằng chúng ta luôn có thể giảm thiểu hiện tượng quá khớp bằng cách thu thập thêm dữ liệu huấn luyện, nhưng trong trường hợp ngắn hạn thì giải pháp này có thể không khả thi do quá tốn kém, lãng phí thời gian, hoặc nằm ngoài khả năng của ta. Hiện tại, chúng ta có thể giả sử rằng ta đã thu thập được một lượng tối đa dữ liệu chất lượng và sẽ tập trung vào các kỹ thuật điều chuẩn.

Nhắc lại rằng trong ví dụ về việc khớp đường cong đa thức, chúng ta có thể giới hạn năng lực của mô hình bằng việc đơn thuần điều chỉnh số bậc của đa thức. Đúng như vậy, giới hạn số đặc trưng là một kỹ thuật phổ biến để tránh hiện tượng quá khớp. Tuy nhiên, việc đơn thuần loại bỏ các đặc trưng có thể hơi quá mức cần thiết. Quay lại với ví dụ về việc khớp đường cong đa thức, hãy xét chuyện gì sẽ xảy ra với đầu vào nhiều chiều. Ta mở rộng đa thức cho dữ liệu đa biến bằng việc thêm các đơn thức, hay nói đơn giản là thêm tích của lũy thừa các biến. Bậc của một đơn thức là tổng của các số mũ. Ví dụ, $x^2_1x_2$, và $x_3x^2_5$ đều là các đơn thức bậc 3.

Lưu ý rằng số lượng đơn thức bậc d tăng cực kỳ nhanh theo d. Với k biến, số lượng các đơn thức bậc d là $\binom{k-1+d}{k-1}$. Chỉ một thay đổi nhỏ về số bậc, ví dụ từ 2 lên 3 cũng sẽ tăng độ phức tạp của mô hình một cách chóng mặt. Do vậy, chúng ta cần có một công cụ tốt hơn để điều chỉnh độ phức tạp của hàm số.

4.4. Dropout

Vừa xong ở Phần 4.3, chúng tôi đã giới thiệu cách tiếp cận điển hình để điều chuẩn các mô hình thống kê bằng cách phạt giá trị chuẩn ℓ_2 của các trọng số. Theo ngôn ngữ xác suất, ta có thể giải thích kĩ thuật này bằng cách nói ta đã có một niềm tin từ trước rằng các trọng số được lấy ngẫu nhiên từ một phân phối Gauss với trung bình bằng 0. Hiểu một cách trực quan, ta có thể nói

rằng mô hình được khuyến khích trải rộng giá trị các trọng số ra trên nhiều đặc trưng thay vì quá phụ thuộc vào một vài những liên kết có khả năng không chính xác.

4.4.1. Bàn lại về Quá khớp

Khi có nhiều đặc trưng hơn số mẫu, các mô hình tuyến tính sẽ có xu hướng quá khớp. Tuy nhiên nếu có nhiều mẫu hơn số đặc trưng, nhìn chung ta có thể tin cậy mô hình tuyến tính sẽ không quá khớp. Thật không may, mô hình tuyến tính dựa trên tính ổn định này để khái quát hoá lại kèm theo một cái giá phải trả: Mô hình tuyến tính không màng tới sự tương tác giữa các đặc trưng, nếu chỉ được áp dụng một cách đơn giản. Mỗi đặc trưng sẽ được gán một giá trị trọng số hoặc là âm, hoặc là dương mà không màng tới ngữ cảnh.

Trong các tài liệu truyền thống, vấn đề cốt lõi giữa khả năng khái quát và tính linh hoạt này được gọi là đánh đổi độ chệch - phương sai (bias-variance tradeoff). Mô hình tuyến tính có độ chệch cao (vì nó chỉ có thể biểu diễn một nhóm nhỏ các hàm số) nhưng lại có phương sai thấp (vì nó cho kết quả khá tương đồng trên nhiều tập dữ liệu được lấy mẫu ngẫu nhiên).

Mạng nơ-ron sâu lại nằm ở thái cực trái ngược trên phổ độ chệch - phương sai. Khác với mô hình tuyến tính, các mạng nơ-ron không bị giới hạn ở việc chỉ được xét từng đặc trưng một cách riêng biệt. Chúng có thể học được sự tương tác giữa các nhóm đặc trưng. Chẳng hạn, chúng có thể suy ra được rằng nếu từ "Nigeria" và "Western Union" xuất hiện cùng nhau trong một email thì đó là thư rác, nhưng nếu hai từ đó xuất hiện riêng biệt thì lại không phải.

Ngay cả khi số mẫu nhiều hơn hẳn so với số đặc trưng, mạng nơ-ron sâu vẫn có thể quá khớp. Năm 2017, một nhóm các nhà nghiên cứu đã minh họa khả năng linh hoạt tột cùng của mạng nơ-ron bằng cách huấn luyện mạng nơ-ron sâu trên tập ảnh được gán nhãn ngẫu nhiên. Dù không hề có bất cứ một khuôn mẫu nào liên kết đầu vào và đầu ra, họ phát hiện rằng mạng nơ-ron được tối ưu bằng SGD vẫn có thể khớp tất cả nhãn trên tập huấn luyện một cách hoàn hảo.

Hãy cùng xem xét ý nghĩa của điều này. Nếu nhãn được gán ngẫu nhiên từ một phân phối đều với 10 lớp, sẽ không có bộ phân loại nào có thể có độ chính xác cao hơn 10% trên tập dữ liệu kiểm tra. Khoảng cách khái quát là tận 90%. Nếu mô hình của chúng ta có đủ năng lực để quá khớp tới như vậy, phải như thế nào chúng ta mới có thể trông đợi rằng mô hình sẽ không quá khớp? Nền tảng toán học đằng sau tính chất khái quát hoá hóc búa của mạng nơ-ron sâu vẫn còn là một câu hỏi mở và chúng tôi khuyến khích bạn đọc chú trọng lý thuyết đào sâu hơn vào chủ đề này. Còn bây giờ, hãy quay về bề mặt của vấn đề và chuyển sang tìm hiểu các công cụ dựa trên thực nghiệm để cải thiện khả năng khái quát của các mạng nơ-ron sâu.

4.4.2. Khả năng Kháng Nhiễu

Hãy cùng nghĩ một chút về thứ mà ta mong đợi từ một mô hình dự đoán tốt. Ta muốn mô hình hoạt động tốt khi gặp dữ liệu mà nó chưa từng thấy. Lý thuyết khái quát cổ điển cho rằng: để thu hẹp khoảng cách giữa chất lượng khi huấn luyện và chất lượng khi kiểm tra, ta nên hướng tới một mô hình đơn giản. Sự đơn giản này có thể nằm ở việc đặc trưng có số chiều thấp, chuẩn (nghịch đảo) của các tham số là một phép đo khác cho sự đơn giản. Một khái niệm hữu ích khác để biểu diễn sự đơn giản là độ mượt, tức hàm số không nên quá nhạy với những thay đổi nhỏ ở đầu vào. Ví dụ, khi phân loại ảnh, ta mong muốn rằng việc thêm một chút nhiễu ngẫu nhiên vào các điểm ảnh sẽ không ảnh hưởng nhiều tới kết quả dự đoán.

Vào năm 1995, Christopher Bishop đã chính quy hóa ý tưởng này khi ông chứng minh rằng việc huấn luyện với đầu vào chứa nhiễu tương đương với điều chuẩn Tikhonov [Bishop, 1995]. Công trình này đã chỉ rõ mối liên kết toán học giữa điều kiện hàm là mượt (nên nó cũng đơn giản) với khả năng kháng nhiễu đầu vào của hàm số.

Và rồi vào năm 2014, Srivastava et al. [Srivastava et al., 2014] đã phát triển một ý tưởng thông minh để áp dụng ý tưởng trên của Bishop cho các tầng nội bộ của mạng nơ-ron. Cụ thể, họ đề xuất việc thêm nhiễu vào mỗi tầng của mạng trước khi tính toán các tầng kế tiếp trong quá trình huấn luyện. Họ nhận ra rằng khi huấn luyện mạng đa tầng, thêm nhiễu vào dữ liệu chỉ ép buộc điều kiện mượt lên phép ánh xạ giữa đầu vào và đầu ra.

Ý tưởng này, có tên gọi là dropout, hoạt động bằng cách thêm nhiễu khi tính toán các tầng nội bộ trong lượt truyền xuôi và nó đã trở thành một kỹ thuật tiêu chuẩn để huấn luyện các mạng nơ-ron. Phương pháp này có tên gọi như vậy là bởi ta loại bỏ (drop out) một số nơ-ron trong quá trình huấn luyện. Tại mỗi vòng lặp huấn luyện, phương pháp dropout tiêu chuẩn sẽ đặt giá trị của một lượng nhất định (thường là 50%) các nút trong mỗi tầng về không, trước khi tính toán các tầng kế tiếp.

Để nói cho rõ, mối liên kết đến Bishop là của chúng tôi tự đặt ra. Đáng ngạc nhiên, bài báo gốc về dropout xây dựng cách hiểu trực giác bằng việc so sánh nó với quá trình sinh sản hữu tính. Các tác giả cho rằng hiện tượng quá khớp mạng nơ-ron là biểu hiện của việc mỗi tầng đều dựa vào một khuôn mẫu nhất định của các giá trị kích hoạt ở tầng trước đó, họ gọi trạng thái này là đồng thích nghi. Họ khẳng định rằng dropout phá bỏ sự đồng thích nghi này, tương tự như luận điểm sinh sản hữu tính phá bỏ các gen đã đồng thích nghi.

Thách thức chính bây giờ là làm thế nào để thêm nhiễu. Một cách để làm điều này là thêm nhiễu một cách không thiên lệch sao cho giá trị kỳ vọng của mỗi tầng bằng giá trị kỳ vọng của chính tầng đó trước khi được thêm nhiễu, giả sử rằng các tầng khác được giữ nguyên.

Trong nghiên cứu của Bishop, ông thêm nhiễu Gauss cho đầu vào của một mô hình tuyến tính như sau: Tại mỗi bước huấn luyện, ông đã thêm nhiễu lấy từ một phân phối có trung bình bằng không $\epsilon \sim N(0, \sigma^2)$ cho đầu vào x, thu được một điểm bị nhiễu $x' = x + \epsilon$ với kỳ vọng $E[x'] = x$.

Với điều chuẩn dropout tiêu chuẩn, ta khử độ chệch tại mỗi tầng bằng cách chuẩn hóa theo tỉ lệ các nút được giữ lại (chứ không phải các nút bị loại bỏ). Nói cách khác, dropout với xác suất dropout p được áp dụng như sau:

$$h' = \begin{cases} 0 & \text{with probability } p \\ \dfrac{h}{1-p} & \text{otherwise} \end{cases}$$

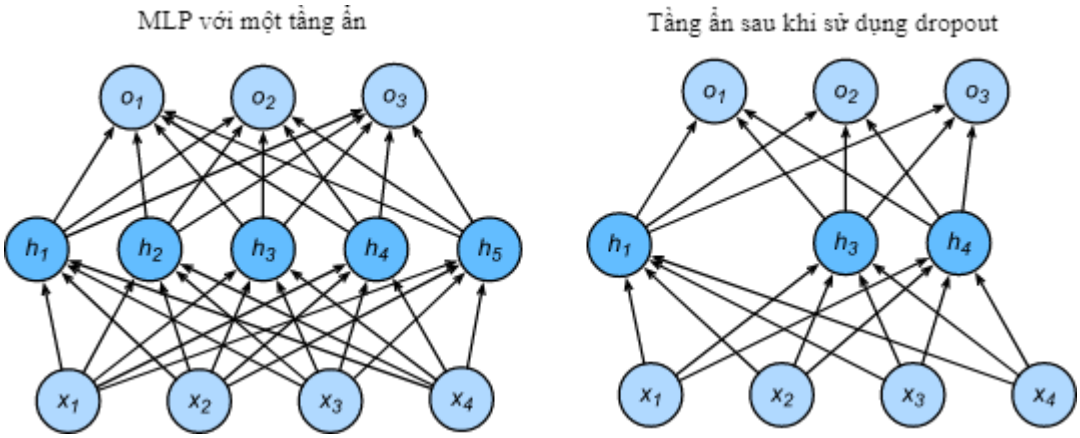
Như ta mong muốn, kỳ vọng không bị thay đổi, hay nói cách khác $E[h'] = h$. Đầu ra của các hàm kích hoạt trung gian h được thay thế bởi một biến ngẫu nhiên h' với kỳ vọng tương ứng.

4.4.3. Dropout trong Thực tiễn

Nhắc lại về mạng perceptron đa tầng với duy nhất một tầng ẩn có 5 nút ẩn. Kiến trúc mạng được biểu diễn như sau

$$\begin{aligned} \mathbf{H} &= \sigma\left(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}\right), \\ \mathbf{O} &= \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)} \end{aligned}$$

Khi chúng ta áp dụng dropout cho một tầng ẩn, tức gán mỗi nút ẩn bằng không với xác suất là p, kết quả có thể được xem như là một mạng chỉ chứa một tập con của các nơ-ron ban đầu. Trong Hình 4.7, h_2 và h_5 bị loại bỏ. Hệ quả là, việc tính toán y không còn phụ thuộc vào h_2 và h_5 nữa và gradient tương ứng của chúng cũng biến mất khi thực hiện lan truyền ngược. Theo cách này, việc tính toán tầng đầu ra không thể quá phụ thuộc vào bất kỳ một thành phần nào trong $h_1 \ldots, h_5$.



Hình 4.7. MLP trước và sau khi dropout

Thông thường, chúng ta sẽ vô hiệu hóa dropout tại thời điểm kiểm tra. Với một mô hình đã huấn luyện và một mẫu kiểm tra, ta sẽ không thực hiện loại bỏ bất kỳ nút nào (do đó cũng không cần chuẩn hóa). Tuy nhiên cũng có một vài ngoại lệ. Một vài nhà nghiên cứu sử dụng dropout tại thời điểm kiểm tra như một thủ thuật đề ước lượng độ bất định trong dự đoán của mạng nơ-ron: nếu các dự đoán giống nhau với nhiều mặt nạ dropout khác nhau, ta có thể nói rằng mạng đó đáng tin cậy hơn. Hiện tại, ta sẽ để dành phần ước lượng độ bất định này cho các chương sau.

4.4.4. Kết luận

- Ngoài phương pháp kiểm soát số chiều và độ lớn của vector trọng số, dropout cũng là một công cụ khác để tránh tình trạng quá khớp. Thông thường thì cả ba cách được sử dụng cùng nhau.

- Dropout thay thế giá trị kích hoạt h bằng một biến ngẫu nhiên h′ với giá trị kỳ vọng h và phương sai bằng xác suất dropout p.

- Dropout chỉ được sử dụng trong quá trình huấn luyện.

4.5. Lan truyền xuôi, Lan truyền ngược và Đồ thị tính toán

Cho đến lúc này, ta đã huấn luyện các mô hình với giải thuật hạ gradient ngẫu nhiên theo minibatch. Tuy nhiên, khi lập trình thuật toán, ta mới chỉ bận tâm đến các phép tính trong quá trình lan truyền xuôi qua mô hình. Khi cần tính gradient, ta mới chỉ đơn giản gọi hàm backward và mô-đun autograd sẽ lo các chi tiết tính toán.

Việc tính toán gradient tự động sẽ giúp công việc lập trình các thuật toán học sâu được đơn giản hóa đi rất nhiều. Trước đây, khi chưa có công cụ tính vi phân tự động, ngay cả khi ta chỉ thay đổi một chút các mô hình phức tạp, các đạo hàm rắc rối cũng cần phải được tính lại một cách thủ công. Điều đáng ngạc nhiên là các bài báo học thuật thường có các công thức cập nhật mô hình dài hàng trang giấy. Vậy nên dù vẫn phải tiếp tục dựa vào autograd để có thể tập trung vào những phần thú vị của học sâu, bạn vẫn nên nắm rõ thay vì chỉ hiểu một cách hời hợt cách tính gradient nếu bạn muốn tiến xa hơn.

Trong mục này, ta sẽ đi sâu vào chi tiết của lan truyền ngược (thường được gọi là backpropagation hoặc backprop). Ta sẽ sử dụng một vài công thức toán học cơ bản và đồ thị tính toán để giải thích một cách chi tiết cách thức hoạt động cũng như cách lập trình các kỹ thuật này. Và để bắt đầu, ta sẽ tập trung giải trình một perceptron đa tầng gồm ba tầng (một tầng ẩn) đi kèm với suy giảm trọng số (điều chuẩn ℓ_2).

4.5.1. Lan truyền Xuôi

Lan truyền xuôi là quá trình tính toán cũng như lưu trữ các biến trung gian (bao gồm cả đầu ra) của mạng nơ-ron theo thứ tự từ tầng đầu vào đến tầng đầu ra. Bây giờ ta sẽ thực hiện qua từng bước trong cơ chế vận hành của mạng nơ-ron sâu có một tầng ẩn. Điều này nghe có vẻ tẻ nhạt nhưng theo như cách nói dân giã, bạn phải "tập đi trước khi tập chạy".

Để đơn giản hóa vấn đề, ta giả sử mẫu đầu vào là $x \in R^d$ và tầng ẩn của ta không có hệ số điều chỉnh. Ở đây biến trung gian là:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x},$$

trong đó $W^{(1)} \in R^{h \times d}$ là tham số trọng số của tầng ẩn. Sau khi đưa biến trung gian $z \in R^h$ qua hàm kích hoạt ϕ, ta thu được vector kích hoạt ẩn với h phần tử,

$$\mathbf{h} = \phi(\mathbf{z}).$$

Biến ẩn h cũng là một biến trung gian. Giả sử tham số của tầng đầu ra chỉ gồm trọng số $W^{(2)} \in R^{q \times h}$, ta sẽ thu được một vector với q phần tử ở tầng đầu ra:

$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}.$$

Giả sử hàm mất mát là l và nhãn của mẫu là y, ta có thể tính được lượng mất mát cho một mẫu dữ liệu duy nhất,

$$L = l(\mathbf{o}, y).$$

Theo định nghĩa của điều chuẩn $\ell 2$ với siêu tham số λ, lượng điều chuẩn là:

$$s = \frac{\lambda}{2}(\|\mathbf{W}^{(1)}\|_F^2 + \|\mathbf{W}^{(2)}\|_F^2),$$

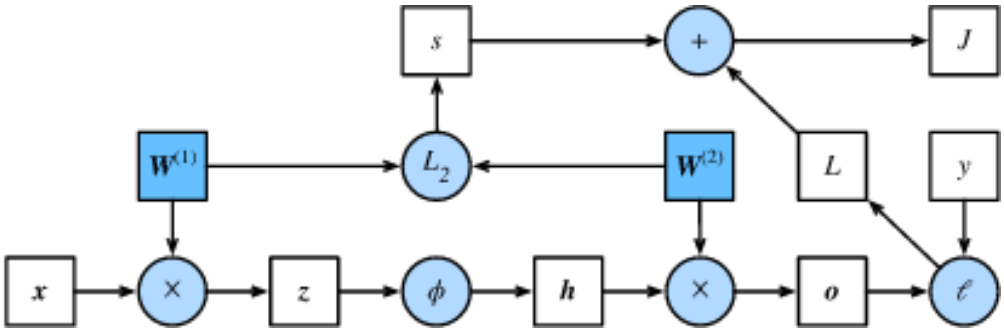
trong đó chuẩn Frobenius của ma trận chỉ đơn giản là chuẩn L_2 của vector thu được sau khi trải phẳng ma trận. Cuối cùng, hàm mất mát được điều chuẩn của mô hình trên một mẫu dữ liệu cho trước là:

$$J = L + s.$$

Ta sẽ bàn thêm về hàm mục tiêu J ở phía dưới.

4.5.2. Đồ thị Tính toán của Lan truyền Xuôi

Vẽ đồ thị tính toán giúp chúng ta hình dung được sự phụ thuộc giữa các toán tử và các biến trong quá trình tính toán. Hình 4.8 thể hiện đồ thị tương ứng với mạng nơ-ron đã miêu tả ở trên. Góc trái dưới biểu diễn đầu vào trong khi góc phải trên biểu diễn đầu ra. Lưu ý rằng hướng của các mũi tên (thể hiện luồng dữ liệu) chủ yếu là đi qua phải và hướng lên trên.



Hình 4.8. Đồ thị tính toán

4.5.3. Lan truyền Ngược

Lan truyền ngược là phương pháp tính gradient của các tham số mạng nơ-ron. Nói một cách đơn giản, phương thức này duyệt qua mạng nơ-ron theo chiều ngược lại, từ đầu ra đến đầu vào, tuân theo quy tắc dây chuyền trong giải tích.

Thuật toán lan truyền ngược lưu trữ các biến trung gian (là các đạo hàm riêng) cần thiết trong quá trình tính toán gradient theo các tham số. Giả sử chúng ta có hàm Y=f(X) và Z=g(Y)=g∘f(X), trong đó đầu vào và đầu ra X,Y,Z là các tensor có kích thước bất kỳ. Bằng cách sử dụng quy tắc dây chuyền, chúng ta có thể tính đạo hàm của Z theo X như sau:

$$\frac{\partial \mathsf{Z}}{\partial \mathsf{X}} = \text{prod}(\frac{\partial \mathsf{Z}}{\partial \mathsf{Y}}, \frac{\partial \mathsf{Y}}{\partial \mathsf{X}}).$$

Ở đây, chúng ta sử dụng toán tử prod để nhân các đối số sau khi các phép tính cần thiết như là chuyển vị và hoán đổi đã được thực hiện. Với vector, điều này khá đơn giản: nó chỉ đơn thuần là phép nhân ma trận. Với các tensor nhiều chiều thì sẽ có các phương án tương ứng phù hợp. Toán tử prod sẽ đơn giản hoá việc ký hiệu.

Các tham số của mạng nơ-ron đơn giản với một tầng ẩn là $W^{(1)}$ và $W^{(2)}$. Mục đích của lan truyền ngược là để tính gradient $\partial J/\partial W^{(1)}$ và $\partial J/\partial W^{(2)}$. Để làm được điều này, ta áp dụng quy tắc dây chuyền và lần lượt tính gradient của các biến trung gian và tham số. Các phép tính trong lan truyền ngược có thứ tự ngược lại so với các phép tính trong lan truyền xuôi, bởi ta muốn bắt đầu từ kết quả của đồ thị tính toán rồi dần đi tới các tham số. Bước đầu tiên đó là tính gradient của hàm mục tiêu J=L+s theo mất mát L và điều chuẩn s.

$$\frac{\partial J}{\partial L} = 1 \text{ và } \frac{\partial J}{\partial s} = 1.$$

Tiếp theo, ta tính gradient của hàm mục tiêu theo các biến của lớp đầu ra o, sử dụng quy tắc dây chuyền.

$$\frac{\partial J}{\partial \mathbf{o}} = \text{prod}(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \mathbf{o}}) = \frac{\partial L}{\partial \mathbf{o}} \in \mathbb{R}^q.$$

Kế tiếp, ta tính gradient của điều chuẩn theo cả hai tham số.

$$\frac{\partial s}{\partial \mathbf{W}^{(1)}} = \lambda \mathbf{W}^{(1)} \text{and} \frac{\partial s}{\partial \mathbf{W}^{(2)}} = \lambda \mathbf{W}^{(2)}.$$

Bây giờ chúng ta có thể tính gradient $\partial J/\partial W^{(2)} \in R^{q \times h}$ của các tham số mô hình gần nhất với tầng đầu ra. Áp dụng quy tắc dây chuyền, ta có:

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \text{prod}(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial \mathbf{o}}{\partial \mathbf{W}^{(2)}}) + \text{prod}(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial \mathbf{W}^{(2)}}) = \frac{\partial J}{\partial \mathbf{o}} \mathbf{h}^\top + \lambda \mathbf{W}^{(2)}.$$

Để tính được gradient theo $W^{(1)}$ ta cần tiếp tục lan truyền ngược từ tầng đầu ra đến các tầng ẩn. Gradient theo các đầu ra của tầng ẩn $\partial J/\partial h \in R^h$ được tính như sau:

$$\frac{\partial J}{\partial \mathbf{h}} = \text{prod}(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial \mathbf{o}}{\partial \mathbf{h}}) = {\mathbf{W}^{(2)}}^\top \frac{\partial J}{\partial \mathbf{o}}.$$

Vì hàm kích hoạt ϕ áp dụng cho từng phần tử, việc tính gradient $\partial J/\partial z \in R^h$ của biến trung gian z cũng yêu cầu sử dụng phép nhân theo từng phần tử, kí hiệu bởi ⊙.

$$\frac{\partial J}{\partial \mathbf{z}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{h}}, \frac{\partial \mathbf{h}}{\partial \mathbf{z}}\right) = \frac{\partial J}{\partial \mathbf{h}} \odot \phi'(\mathbf{z}).$$

Cuối cùng, ta có thể tính gradient $\partial J/\partial W^{(1)} \in R^{h \times d}$ của các tham số mô hình gần nhất với tầng đầu vào. Theo quy tắc dây chuyền, ta có

$$\frac{\partial J}{\partial \mathbf{W}^{(1)}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{z}}, \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}}\right) + \text{prod}\left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial \mathbf{W}^{(1)}}\right) = \frac{\partial J}{\partial \mathbf{z}} \mathbf{x}^\top + \lambda \mathbf{W}^{(1)}.$$

4.5.4. Huấn luyện một Mô hình

Khi huấn luyện các mạng nơ-ron, lan truyền xuôi và lan truyền ngược phụ thuộc lẫn nhau. Cụ thể với lan truyền xuôi, ta duyệt đồ thị tính toán theo hướng của các quan hệ phụ thuộc và tính tất cả các biến trên đường đi. Những biến này sau đó được sử dụng trong lan truyền ngược khi thứ tự tính toán trên đồ thị bị đảo ngược lại. Hệ quả là ta cần lưu trữ các giá trị trung gian cho đến khi lan truyền ngược hoàn tất. Đây cũng chính là một trong những lý do khiến lan truyền ngược yêu cầu nhiều bộ nhớ hơn đáng kể so với khi chỉ cần đưa ra dự đoán.

Ta tính các tensor gradient và giữ các biến trung gian lại để sử dụng trong quy tắc dây chuyền. Việc huấn luyện trên các minibatch chứa nhiều mẫu, do đó cần lưu trữ nhiều giá trị kích hoạt trung gian hơn cũng là một lý do khác.

4.5.5. Kết luận

- Lan truyền xuôi lần lượt tính và lưu trữ các biến trung gian từ tầng đầu vào đến tầng đầu ra trong đồ thị tính toán được định nghĩa bởi mạng nơ-ron.

- Lan truyền ngược lần lượt tính và lưu trữ các gradient của biến trung gian và tham số mạng nơ-ron theo chiều ngược lại.

- Khi huấn luyện các mô hình học sâu, lan truyền xuôi và lan truyền ngược phụ thuộc lẫn nhau.

- Việc huấn luyện cần nhiều bộ nhớ lưu trữ hơn đáng kể so với việc dự đoán.

CHƯƠNG 5

TÍNH TOÁN HỌC SÂU

Ngoài các tập dữ liệu khổng lồ và phần cứng mạnh mẽ, không thể không nhắc tới vai trò quan trọng của các công cụ phần mềm tốt trong sự phát triển chóng mặt của học sâu. Mở đầu với thư viện tiên phong Theano được phát hành vào năm 2007, các công cụ mã nguồn mở linh hoạt đã giúp các nhà nghiên cứu nhanh chóng thử nghiệm các mô hình bằng cách tránh việc bắt người dùng phải xây dựng lại các thành phần tiêu chuẩn nhưng vẫn cho phép việc thay đổi ở bậc thấp. Theo thời gian, các thư viện học sâu ngày càng phát triển để cung cấp tính trừu tượng cao hơn. Tương tự với việc các nhà thiết kế chất bán dẫn đi từ việc chỉ rõ các lựa chọn bóng bán dẫn đến mạch logic để viết mã nguồn, các nhà nghiên cứu mạng nơ-ron sâu đã thay đổi từ việc nghĩ về hành vi của từng nơ-ron nhân tạo đơn lẻ sang việc xem xét cả một tầng trong mạng nơ-ron. Giờ đây, họ thường thiết kế các kiến trúc mạng ở mức độ trừu tượng là các khối.

Đến nay, chúng tôi đã giới thiệu một vài khái niệm học máy cơ bản, rồi tiến tới các mô hình học sâu. Ở chương trước, ta đã lập trình từng thành phần của một perceptron đa tầng từ đầu và biết được cách tận dụng thư viện Gluon từ MXNet để xây dựng lại mô hình một cách dễ dàng hơn. Để giúp bạn có những bước tiến xa hơn mức mong đợi, chúng tôi tập trung vào việc sử dụng các thư viện và không đề cập đến những chi tiết nâng cao hơn về cách hoạt động của chúng. Trong chương này, chúng tôi sẽ vén tấm màn bí ẩn và đào sâu vào những yếu tố chính của tính toán học sâu; cụ thể là việc xây dựng mô hình, truy cập và khởi tạo tham số, thiết kế các tầng và khối tùy chỉnh, đọc và ghi mô hình lên ổ cứng và cuối cùng là tận dụng GPU nhằm đạt được tốc độ đáng kể. Những hiểu biết này sẽ giúp bạn từ một người dùng cuối (end user) trở thành một người dùng thành thạo (power user), cung cấp cho bạn các công cụ cần thiết để gặt hái lợi ích của một thư viện học sâu trưởng thành, đồng thời giữ được sự linh hoạt để lập trình những mô hình phức tạp hơn, bao gồm cả những mô hình mà bạn tự phát minh! Mặc dù chương này không giới thiệu bất cứ mô hình hay tập dữ liệu mới nào, các chương sau về mô hình nâng cao sẽ phụ thuộc rất nhiều vào những kỹ thuật sắp được nhắc đến.

5.1. Tầng và Khối

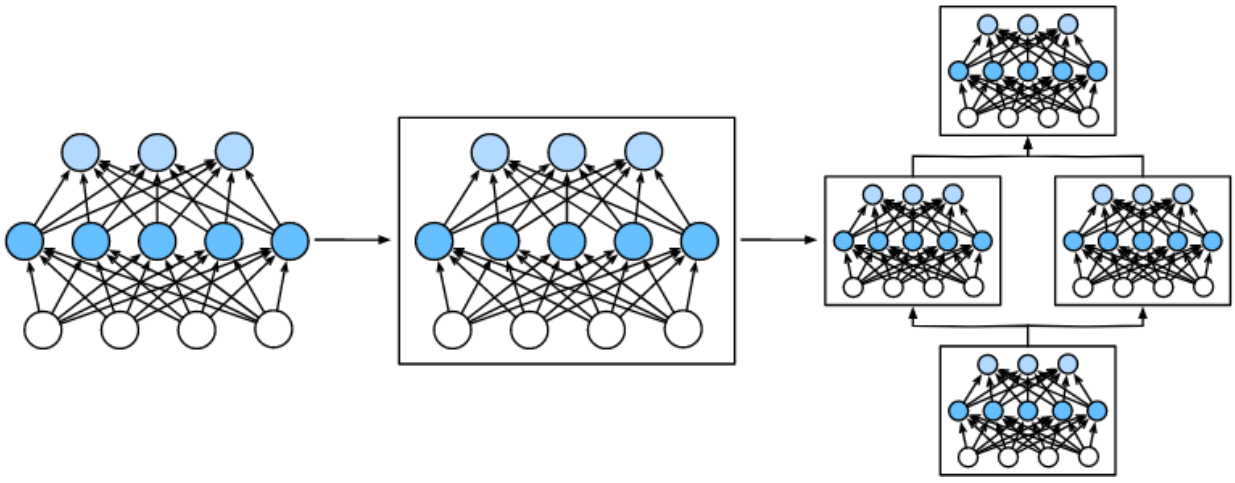
Khi lần đầu giới thiệu về các mạng nơ-ron, ta tập trung vào các mô hình tuyến tính với một đầu ra duy nhất. Như vậy toàn bộ mô hình chỉ chứa một nơ-ron. Lưu ý rằng một nơ-ron đơn lẻ (i) nhận một vài đầu vào; (ii) tạo một đầu ra (vô hướng) tương ứng; và (iii) có một tập các tham số liên quan có thể được cập nhật để tối ưu một hàm mục tiêu nào đó mà ta quan tâm. Sau đó, khi bắt đầu nghĩ về các mạng có nhiều đầu ra, ta tận dụng các phép tính vector để mô tả nguyên một tầng nơ-ron. Cũng giống như các nơ-ron riêng lẻ, các tầng (i) nhận một số đầu vào, (ii) tạo các đầu ra tương ứng, và (iii) được mô tả bằng một tập các tham số có thể điều chỉnh được. Trong hồi quy softmax, bản thân tầng duy nhất ấy chính là một mô hình. Thậm chí đối với các perceptron đa tầng, ta vẫn có thể nghĩ về chúng theo cấu trúc cơ bản này.

Điều thú vị là đối với các perceptron đa tầng, cả mô hình và các tầng cấu thành đều chia sẻ cấu trúc này. (Toàn bộ) mô hình nhận các đầu vào thô (các đặc trưng), tạo các đầu ra (các dự đoán) và sở hữu các tham số (được tập hợp từ tất cả các tầng cấu thành). Tương tự, mỗi tầng riêng lẻ cũng nhận các đầu vào (được cung cấp bởi tầng trước đó), tính toán các đầu ra (cũng chính là các đầu vào cho tầng tiếp theo), và có một tập các tham số có thể điều chỉnh thông qua việc cập nhật dựa trên tín hiệu được truyền ngược từ tầng kế tiếp.

Dù bạn có thể nghĩ rằng các nơ-ron, các tầng và các mô hình đã cung cấp đủ sự trừu tượng để bắt tay vào làm việc, hóa ra sẽ là thuận tiện hơn khi ta bàn về các thành phần lớn hơn một tầng riêng lẻ nhưng lại nhỏ hơn toàn bộ mô hình. Ví dụ, kiến trúc ResNet-152, rất phổ biến trong thị giác máy tính, sở hữu hàng trăm tầng. Nó bao gồm các khuôn mẫu nhóm tầng được lặp lại nhiều lần. Việc lập trình từng tầng của một mạng như vậy có thể trở nên tẻ nhạt. Mối quan tâm này không chỉ là trên lý thuyết — các khuôn mẫu thiết kế như vậy rất phổ biến trong thực tế. Kiến trúc ResNet được đề cập ở trên đã giành chiến thắng trong hai cuộc thi thị giác máy tính ImageNet và COCO năm 2015, trong cả bài toán nhận dạng và bài toán phát hiện [He et al., 2016a] và vẫn là một kiến trúc được tin dùng cho nhiều bài toán thị giác. Các kiến trúc tương tự, trong đó các tầng được sắp xếp thành những khuôn mẫu lặp lại, hiện đã trở nên thông dụng ở nhiều lĩnh vực khác, bao gồm cả xử lý ngôn ngữ tự nhiên và xử lý tiếng nói.

Để thiết kế các mạng phức tạp này, ta sẽ giới thiệu khái niệm khối trong mạng nơ-ron. Một khối có thể mô tả một tầng duy nhất, một mảng đa tầng hoặc toàn bộ một mô hình! Dưới góc nhìn xây dựng phần mềm, một Block (Khối) là một lớp. Bất kỳ một lớp con nào của Block đều phải định nghĩa phương thức forward để chuyển hóa đầu vào thành đầu ra và phải lưu trữ mọi tham số cần thiết. Lưu ý rằng có một vài Block sẽ không yêu cầu chứa bất kỳ tham số nào cả! Ngoài ra, một Block phải sở hữu một phương thức backward cho mục đích tính toán gradient. May mắn thay, nhờ sự trợ giúp đắc lực của gói autograd nên khi định nghĩa Block, ta chỉ cần quan tâm đến các tham số và hàm forward.

Một lợi ích khi làm việc ở mức độ trừu tượng Block đó là ta có thể kết hợp chúng, thường là theo phương pháp đệ quy, để tạo ra các thành phần lớn hơn (xem hình minh họa trong Hình 5.1).



Hình 5.1. Nhiều tầng được kết hợp để tạo thành các khối

Bằng cách định nghĩa các khối với độ phức tạp tùy ý, các mạng nơ-ron phức tạp có thể được lập trình với mã nguồn ngắn gọn một cách đáng ngạc nhiên.

5.1.5. kết luận

- Các tầng trong mạng nơ-ron là các Khối.
- Nhiều tầng có thể cấu thành một Khối.
- Nhiều Khối có thể cấu thành một Khối.
- Một Khối có thể chứa các đoạn mã nguồn.
- Các Khối đảm nhiệm nhiều tác vụ bao gồm khởi tạo tham số và lan truyền ngược.

- Việc gắn kết các tầng và khối một cách tuần tự được đảm nhiệm bởi Khối Sequential.

5.2. Quản lý Tham số

Một khi ta đã chọn được kiến trúc mạng và các giá trị siêu tham số, ta sẽ bắt đầu với vòng lặp huấn luyện với mục tiêu là tìm các giá trị tham số để cực tiểu hóa hàm mục tiêu. Sau khi huấn luyện xong, ta sẽ cần các tham số đó để đưa ra dự đoán trong tương lai. Hơn nữa, thi thoảng ta sẽ muốn trích xuất tham số để sử dụng lại trong một hoàn cảnh khác, có thể lưu trữ mô hình để thực thi trong một phần mềm khác hoặc để rút ra hiểu biết khoa học bằng việc phân tích mô hình.

Thông thường, ta có thể bỏ qua những chi tiết chuyên sâu về việc khai báo và xử lý tham số bởi Gluon sẽ đảm nhiệm công việc nặng nhọc này. Tuy nhiên, khi ta bắt đầu tiến xa hơn những kiến trúc chỉ gồm các tầng tiêu chuẩn được xếp chồng lên nhau, đôi khi ta sẽ phải tự đi sâu vào việc khai báo và xử lý tham số. Trong mục này, chúng tôi sẽ đề cập đến những việc sau:

- Truy cập các tham số để gỡ lỗi, chẩn đoán mô hình và biểu diễn trực quan.

- Khởi tạo tham số.

- Chia sẻ tham số giữa các thành phần khác nhau của mô hình.

5.3. Khởi tạo trễ

Cho tới nay, có vẻ như ta chưa phải chịu hậu quả của việc thiết lập mạng cẩu thả. Cụ thể, ta đã "giả mù" và làm những điều không trực quan sau:

- Ta định nghĩa kiến trúc mạng mà không xét đến chiều đầu vào.

- Ta thêm các tầng mà không xét đến chiều đầu ra của tầng trước đó.

- Ta thậm chí còn "khởi tạo" các tham số mà không có đầy đủ thông tin để xác định số lượng các tham số của mô hình.

5.4. Các tầng Tuỳ chỉnh

Một trong những yếu tố dẫn đến thành công của học sâu là sự đa dạng của các tầng. Những tầng này có thể được sắp xếp theo nhiều cách sáng tạo để thiết kế nên những kiến trúc phù hợp với nhiều tác vụ khác nhau. Ví dụ, các nhà nghiên cứu đã phát minh ra các tầng chuyên dụng để xử lý ảnh, chữ viết, lặp trên dữ liệu tuần tự, thực thi quy hoạch động, v.v…

5.4.1. Các tầng không có Tham số

Để bắt đầu, ta tạo một tầng tùy chỉnh (một Khối) không chứa bất kỳ tham số nào. Bước này khá quen thuộc nếu bạn còn nhớ phần giới thiệu về Block của Gluon. Lớp CenteredLayer chỉ đơn thuần trừ đi giá trị trung bình từ đầu vào của nó. Để xây dựng nó, chúng ta chỉ cần kế thừa từ lớp Block và lập trình phương thức forward.

5.4.2. Tầng có Tham số

Giờ đây ta đã biết cách định nghĩa các tầng đơn giản, hãy chuyển sang việc định nghĩa các tầng chứa tham số có thể điều chỉnh được trong quá trình huấn luyện. Để tự động hóa các công việc lặp lại, lớp Parameter và từ điển ParameterDict cung cấp một số tính năng quản trị cơ bản. Cụ thể, chúng sẽ quản lý việc truy cập, khởi tạo, chia sẻ, lưu và nạp các tham số mô hình. Bằng cách này, cùng với nhiều lợi ích khác, ta không cần phải viết lại các thủ tục tuần tự hóa (serialization) cho mỗi tầng tùy chỉnh mới.

Lớp Block chứa biến params với kiểu dữ liệu ParameterDict. Từ điển này ánh xạ các xâu kí tự biểu thị tên tham số đến các tham số mô hình (thuộc kiểu Parameter). ParameterDict cũng cung cấp hàm get giúp việc tạo tham số mới với tên và chiều cụ thể trở nên dễ dàng.

5.4.3. Kết luận

- Ta có thể thiết kế các tầng tùy chỉnh thông qua lớp Block. Điều này cho phép ta định nghĩa một cách linh hoạt các tầng có cách hoạt động khác với các tầng có sẵn trong thư viện.

- Một khi đã được định nghĩa, các tầng tùy chỉnh có thể được gọi trong những bối cảnh và kiến trúc tùy ý.

- Các khối có thể có các tham số cục bộ, được lưu trữ dưới dạng đối tượng ParameterDict trong mỗi thuộc tính params của Block.

Chương 6

MẠNG NƠ-RON TÍCH CHẬP

Trong những chương đầu tiên, chúng ta đã làm việc trên dữ liệu ảnh với mỗi mẫu là một mảng điểm ảnh 2D. Tùy vào ảnh đen trắng hay ảnh màu mà ta cần xử lý một hay nhiều giá trị số học tương ứng tại mỗi vị trí điểm ảnh. Cho đến nay, cách ta xử lý dữ liệu với cấu trúc phong phú này vẫn chưa thật sự thoả đáng. Ta chỉ đang đơn thuần loại bỏ cấu trúc không gian từ mỗi bức ảnh bằng cách chuyển chúng thành các vector và truyền chúng qua một mạng MLP (kết nối đầy đủ). Vì các mạng này là bất biến với thứ tự của các đặc trưng, ta sẽ nhận được cùng một kết quả bất kể việc chúng ta có giữ lại thứ tự cấu trúc không gian của các điểm ảnh hay hoán vị các cột của ma trận đặc trưng trước khi khớp các tham số của mạng MLP. Tốt hơn hết, ta nên tận dụng điều đã biết là các điểm ảnh kề cận thường có tương quan lẫn nhau, để xây dựng những mô hình hiệu quả hơn cho việc học từ dữ liệu ảnh.

Chương này sẽ giới thiệu về các Mạng Nơ-ron Tích chập (Convolutional Neural Network - CNN), một họ các mạng nơ-ron ưu việt được thiết kế chính xác cho mục đích trên. Các kiến trúc dựa trên CNN hiện nay xuất hiện trong mọi ngóc ngách của lĩnh vực thị giác máy tính, và đã trở thành kiến trúc chủ đạo mà hiếm ai ngày nay phát triển các ứng dụng thương mại hay tham gia một cuộc thi nào đó liên quan tới nhận dạng ảnh, phát hiện đối tượng, hay phân vùng theo ngữ cảnh mà không xây nền móng dựa trên phương pháp này.

Theo cách hiểu thông dụng, thiết kế của mạng ConvNets đã vay mượn rất nhiều ý tưởng từ ngành sinh học, lý thuyết nhóm và lượng rất nhiều những thí nghiệm nhỏ lẻ khác. Bên cạnh hiệu năng cao trên số lượng mẫu cần thiết để đạt được đủ độ chính xác, các mạng nơ-ron tích chập thường có hiệu quả tính toán hơn, bởi đòi hỏi ít tham số hơn và dễ thực thi song song trên nhiều GPU hơn các kiến trúc mạng dày đặc.

Do đó, các mạng CNN sẽ được áp dụng bất cứ khi nào có thể, và chúng đã nhanh chóng trở thành một công cụ quan trọng đáng tin cậy thậm chí với các tác vụ liên quan tới cấu trúc tuần tự một chiều, như là xử lý âm thanh, văn bản, và phân tích dữ liệu chuỗi thời gian (time series analysis), mà ở đó các mạng nơ-rơn hồi tiếp vốn thường được sử dụng. Với một số điều chỉnh khôn khéo, ta còn có thể dùng mạng CNN cho dữ liệu có cấu trúc đồ thị và hệ thống đề xuất.

Trước hết, chúng ta sẽ đi qua các phép toán cơ bản nhằm tạo nên bộ khung sườn của tất cả các mạng nơ-ron tích chập. Chúng bao gồm các tầng tích chập, các chi tiết cơ bản quan trọng như đệm và sải bước, các tầng gộp dùng để kết hợp thông tin qua các vùng không gian kề nhau, việc sử dụng đa kênh (cũng được gọi là các bộ lọc) ở mỗi tầng và một cuộc thảo luận cẩn thận về cấu trúc của các mạng hiện đại. Chúng ta sẽ kết thúc cho chương này với một ví dụ hoàn toàn hoạt động của mạng LeNet, mạng tích chập đầu tiên đã triển khai thành công và tồn tại nhiều năm trước khi có sự trỗi dậy của kỹ thuật học sâu hiện đại. Ở chương kế tiếp, chúng ta sẽ đắm mình vào việc xây dựng hoàn chỉnh một số kiến trúc CNN tương đối gần đây và khá phổ biến. Thiết kế của chúng chứa hầu hết những kỹ thuật mà ngày nay hay được sử dụng.

6.1. Từ Tầng Kết nối Dày đặc đến phép Tích chập

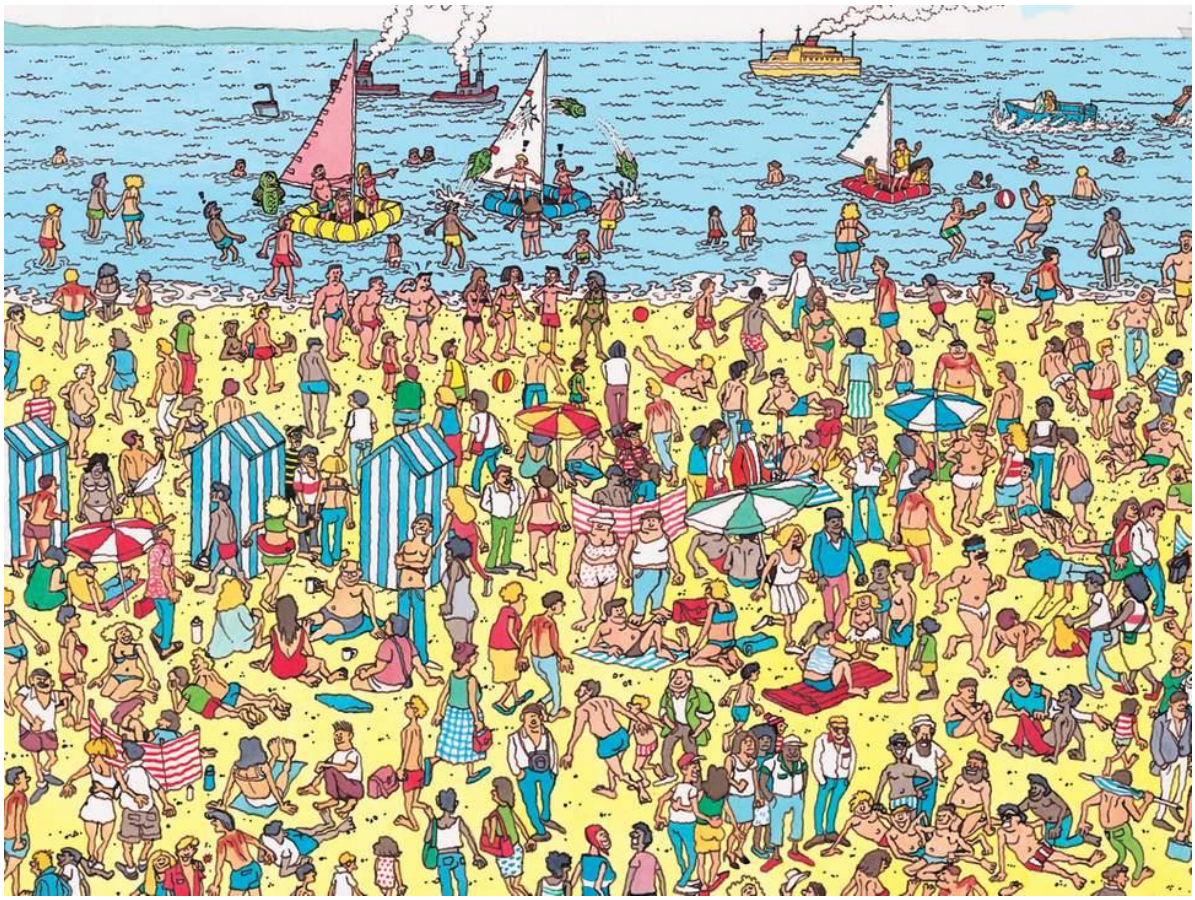
Đến nay, các mô hình mà ta đã thảo luận là các lựa chọn phù hợp nếu dữ liệu mà ta đang xử lý có dạng bảng với các hàng tương ứng với các mẫu, còn các cột tương ứng với các đặc trưng. Với dữ liệu có dạng như vậy, ta có thể dự đoán rằng khuôn mẫu mà ta đang tìm kiếm có thể yêu cầu việc mô hình hóa sự tương tác giữa các đặc trưng, nhưng ta không giả định trước rằng những đặc trưng nào liên quan tới nhau và mối quan hệ của chúng.

Đôi khi ta thực sự không có bất kỳ kiến thức nào để định hướng việc thiết kế các kiến trúc được sắp xếp khéo léo hơn. Trong những trường hợp này, một perceptron đa tầng thường là giải pháp tốt nhất. Tuy nhiên, một khi ta bắt đầu xử lý dữ liệu tri giác đa chiều, các mạng không có cấu trúc này có thể sẽ trở nên quá cồng kềnh.

Hãy quay trở lại với ví dụ phân biệt chó và mèo quen thuộc. Giả sử ta đã thực hiện việc thu thập dữ liệu một cách kỹ lưỡng và thu được một bộ ảnh được gán nhãn chất lượng cao với độ phân giải 1 triệu điểm ảnh. Điều này có nghĩa là đầu vào của mạng sẽ có 1 triệu chiều. Ngay cả việc giảm mạnh xuống còn 1000 chiều ẩn sẽ cần tới một tầng dày đặc (kết nối đầy đủ) có 10^9 tham số. Trừ khi ta có một tập dữ liệu cực lớn (có thể là hàng tỷ ảnh?), một số lượng lớn GPU, chuyên môn cao trong việc tối ưu hóa phân tán và sức kiên nhẫn phi thường, việc học các tham số của mạng này có thể là điều bất khả thi.

Độc giả kỹ tính có thể phản đối lập luận này trên cơ sở độ phân giải 1 triệu điểm ảnh có thể là không cần thiết. Tuy nhiên, ngay cả khi chỉ sử dụng 100.000 điểm ảnh, ta đã đánh giá quá thấp số lượng các nút ẩn cần thiết để tìm các biểu diễn ẩn tốt của các ảnh. Việc học một bộ phân loại nhị phân với rất nhiều tham số có thể sẽ cần tới một tập dữ liệu khổng lồ, có lẽ tương đương với số lượng chó và mèo trên hành tinh này. Tuy nhiên, việc cả con người và máy tính đều có thể phân biệt mèo với chó khá tốt dường như mâu thuẫn với các kết luận trên. Đó là bởi vì các ảnh thể hiện cấu trúc phong phú, thường được khai thác bởi con người và các mô hình học máy theo các cách giống nhau.

6.1.1. Tính Bất biến



Hình 6.1. Một ảnh trong Walker Books

Hãy tưởng tượng rằng ta muốn nhận diện một vật thể trong ảnh. Có vẻ sẽ hợp lý nếu cho rằng bất cứ phương pháp nào ta sử dụng đều không nên quá quan tâm đến vị trí chính xác của vật thể trong ảnh. Lý tưởng nhất, ta có thể học một hệ thống có khả năng tận dụng được kiến thức này bằng một cách nào đó. Lợn thường không bay và máy bay thường không bơi. Tuy nhiên, ta vẫn có thể nhận ra một con lợn đang bay nếu nó xuất hiện. Ý tưởng này được thể hiện rõ rệt trong trò chơi trẻ em 'Đi tìm Waldo', một ví dụ được miêu tả trong Hình 6.1. Trò chơi này bao gồm một số cảnh hỗn loạn với nhiều hoạt động đan xen và Waldo xuất hiện ở đâu đó trong mỗi cảnh (thường ẩn nấp ở một số vị trí khó ngờ tới). Nhiệm vụ của người chơi là xác định vị trí của anh ta. Mặc dù Waldo có trang phục khá nổi bật, việc này có thể vẫn rất khó khăn do có quá nhiều yếu tố gây nhiễu.

Quay lại với ảnh, những trực giác mà ta đã thảo luận có thể được cụ thể hóa hơn nữa để thu được một vài nguyên tắc chính trong việc xây dựng mạng nơ-ron cho thị giác máy tính:

1. Ở một khía cạnh nào đó, các hệ thống thị giác nên phản ứng tương tự với cùng một vật thể bất kể vật thể đó xuất hiện ở đâu trong ảnh (tính bất biến tịnh tiến).
2. Ở khía cạnh khác, các hệ thống thị giác nên tập trung vào các khu vực cục bộ và không quan tâm đến bất kỳ thứ gì khác ở xa hơn trong ảnh (tính cục bộ).

Hãy cùng xem cách biểu diễn những điều trên bằng ngôn ngữ toán học.

6.1.2. Ràng buộc Perceptron Đa tầng

Trong phần này, ta coi hình ảnh và các tầng ẩn là các mảng hai chiều. Để bắt đầu, hãy tưởng tượng một perceptron đa tầng sẽ như thế nào với đầu vào là ảnh kích thước $h \times w$ (biểu diễn dưới dạng ma trận trong toán học và mảng hai chiều khi lập trình), và với các biểu diễn ẩn cũng là các ma trận / mảng hai chiều kích thước $h \times w$. Đặt $x[i, j]$ và $h[i, j]$ lần lượt là điểm ảnh tại vị trí (i, j) của ảnh và biểu diễn ẩn. Để mỗi nút ẩn trong tổng số $h \times w$ nút nhận dữ liệu từ tất cả $h \times w$ đầu vào, ta sẽ chuyển từ việc biểu diễn các tham số bằng ma trận trọng số (như đã thực hiện với perceptron đa tầng trước đây) sang sử dụng các tensor trọng số bốn chiều.
Ta có thể biểu diễn tầng kết nối đầy đủ bằng công thức toán sau:

$$h[i,j] = u[i,j] + \sum_{k,l} W[i,j,k,l] \cdot x[k,l] = u[i,j] + \sum_{a,b} V[i,j,a,b] \cdot x[i+a, j+b].$$

Việc chuyển từ W sang V hoàn toàn chỉ có mục đích thẩm mĩ (tại thời điểm này) bởi có một sự tương ứng một-một giữa các hệ số trong cả hai tensor. Ta chỉ đơn thuần đặt lại các chỉ số dưới (k,l) với k=i+a và l=j+b. Nói cách khác, V[i,j,a,b]=W[i,j,i+a,j+b]. Các chỉ số a,b chạy trên toàn bộ hình ảnh, có thể mang cả giá trị dương và âm. Với bất kỳ vị trí (i,j) nào ở tầng ẩn, giá trị biểu diễn ẩn h[i,j] được tính bằng tổng trọng số của các điểm ảnh nằm xung quanh vị trí (i,j) của x, với trọng số là V[i,j,a,b].

Bây giờ hãy sử dụng nguyên tắc đầu tiên mà ta đã thiết lập ở trên: tính bất biến tịnh tiến. Nguyên tắc này ngụ ý rằng một sự dịch chuyển ở đầu vào x cũng sẽ tạo ra sự dịch chuyển ở biểu diễn ẩn h. Điều này chỉ có thể xảy ra nếu V và u không phụ thuộc vào (i,j), tức V[i,j,a,b]=V[a,b] và u là một hằng số. Vì vậy, ta có thể đơn giản hóa định nghĩa của h.

$$h[i,j] = u + \sum_{a,b} V[a,b] \cdot x[i+a, j+b]$$

Đây là một phép tích chập! Ta đang đánh trọng số cho các điểm ảnh (i+a,j+b) trong vùng lân cận của (i,j) bằng các hệ số V[a,b] để thu được giá trị h[i,j]. Lưu ý rằng V[a,b] cần ít hệ số hơn hẳn so với V[i,j,a,b]. Với đầu vào là hình ảnh 1 megapixel (với tối đa 1 triệu hệ số cho mỗi vị trí), lượng tham số của V[a,b] giảm đi 1 triệu vì không còn phụ thuộc vào vị trí trong ảnh. Ta đã có được tiến triển đáng kể!

Bây giờ hãy sử dụng nguyên tắc thứ hai—tính cục bộ. Như trình bày ở trên, giả sử rằng ta không cần thông tin tại các vị trí quá xa (i,j) để đánh giá những gì đang diễn ra tại h[i,j]. Điều này có nghĩa là ở các miền giá trị $|a|,|b|>\Delta$, ta có thể đặt V[a,b]=0. Tương tự, ta có thể đơn giản hoá h[i,j] như sau

$$h[i,j] = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} V[a,b] \cdot x[i+a, j+b]$$

Một cách ngắn gọn, đây chính là biểu diễn toán học của tầng tích chập. Khi vùng cục bộ xung quanh vị trí đang xét (còn được gọi là vùng tiếp nhận) nhỏ, sự khác biệt so với mạng kết nối đầy đủ có thể rất lớn. Trước đây ta có thể phải cần hàng tỷ tham số để biểu diễn một tầng duy nhất trong mạng xử lý ảnh, hiện giờ chỉ cần vài trăm. Cái giá phải trả là các đặc trưng sẽ trở nên bất biến tịnh tiến và các tầng chỉ có thể nhận thông tin cục bộ. Toàn bộ quá trình học dựa trên việc áp đặt các thiên kiến quy nạp (inductive bias). Khi các thiên kiến đó phù hợp với thực tế, ta sẽ có được các mô hình hoạt động hiệu quả với ít mẫu và khái quát tốt cho dữ liệu chưa gặp. Nhưng tất nhiên, nếu những thiên kiến đó không phù hợp với thực tế, ví dụ như nếu các ảnh không có tính bất biến tịnh tiến, các mô hình có thể sẽ không khái quát tốt.

6.1.3. Phép Tích chập

Hãy cùng xem qua lý do tại sao toán tử trên được gọi là tích chập. Trong toán học, phép tích chập giữa hai hàm số $f,g:R^d \rightarrow R$ được định nghĩa như sau

$$[f \circledast g](x) = \int_{R^d} f(z)g(x-z)dz$$

Trong phép toán này, ta đo lường sự chồng chéo giữa f và g khi g được dịch chuyển một khoảng x và "bị lật lại". Đối với các đối tượng rời rạc, phép tích phân trở thành phép lấy tổng. Chẳng hạn, đối với các vector được định nghĩa trên ℓ_2, là tập các vector vô hạn chiều có tổng bình phương hội tụ, với chỉ số chạy trên Z, ta có phép tích chập sau:

$$[f \circledast g](i) = \sum_{a} f(a)g(i-a)$$

Đối với mảng hai chiều, ta có một tổng tương ứng với các chỉ số (i,j) cho f và (i−a,j−b) cho g. Tổng này nhìn gần giống với định nghĩa tầng tích chập ở trên, nhưng với một khác biệt lớn. Thay vì (i+a,j+b), ta lại sử dụng hiệu. Tuy nhiên, lưu ý rằng sự khác biệt này không phải vấn đề lớn vì ta luôn có thể chuyển về ký hiệu của phép tích chập bằng cách sử dụng $V^{\sim}[a,b]=V[-a,-b]$ để có $h=x \circledast V^{\sim}$. Cũng lưu ý rằng định nghĩa ban đầu thực ra là của phép toán tương quan chéo. Ta sẽ quay trở lại phép toán này trong phần tiếp theo.

6.1.4. Xem lại ví dụ về Waldo

Hãy cùng xem việc xây dựng một bộ phát hiện Waldo cải tiến sẽ trông như thế nào. Tầng tích chập chọn các cửa sổ có kích thước cho sẵn và đánh trọng số cường độ dựa theo mặt nạ V, như được minh họa trong Hình 6.2. Ta hy vọng rằng ở đâu có "tính Waldo" cao nhất, các tầng kích hoạt ẩn cũng sẽ có cao điểm ở đó.

Hình 6.2. Tìm Waldo

Chỉ có một vấn đề với cách tiếp cận này là cho đến nay ta đã vô tư bỏ qua việc hình ảnh bao gồm 3 kênh màu: đỏ, xanh lá cây và xanh dương. Trong thực tế, hình ảnh không hẳn là các đối tượng hai chiều mà là một tensor bậc ba, ví dụ tensor với kích thước 1024×1024×3 điểm ảnh. Chỉ có hai trong số các trục này chứa mối quan hệ về mặt không gian, trong khi trục thứ ba có thể được coi như là một biểu diễn đa chiều cho từng vị trí điểm ảnh.

Do đó, ta phải truy cập x dưới dạng x[i,j,k]. Mặt nạ tích chập phải thích ứng cho phù hợp. Thay vì V[a,b] bây giờ ta có V[a,b,c].

Hơn nữa, tương tự như việc đầu vào là các tensor bậc ba, việc xây dựng các biểu diễn ẩn là các tensor bậc ba tương ứng hoá ra cũng là một ý tưởng hay. Nói cách khác, thay vì chỉ có một biểu diễn 1D tương ứng với từng vị trí không gian, ta muốn có một biểu diễn ẩn đa chiều tương ứng với từng vị trí không gian. Ta có thể coi các biểu diễn ẩn như được cấu thành từ các lưới hai chiều xếp chồng lên nhau. Đôi khi chúng được gọi là kênh (channel) hoặc ánh xạ đặc trưng (feature map). Theo trực giác, bạn có thể tưởng tượng rằng ở các tầng thấp hơn, một số kênh tập trung vào việc nhận diện cạnh trong khi các kênh khác đảm nhiệm việc nhận diện kết cấu, v.v. Để hỗ trợ đa kênh ở cả đầu vào và kích hoạt ẩn, ta có thể thêm tọa độ thứ tư vào V:V[a,b,c,d]. Từ mọi điều trên, ta có:

$$\mathrm{h}[\mathrm{i},\mathrm{j},\mathrm{k}] = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{c} [\mathrm{a},\mathrm{b},\mathrm{c},\mathrm{k}] \cdot \mathrm{x}[\mathrm{i}+\mathrm{a},\mathrm{j}+\mathrm{b},\mathrm{c}]$$

Đây là định nghĩa của một tầng mạng nơ-ron tích chập. Vẫn còn nhiều phép toán mà ta cần phải giải quyết. Chẳng hạn, ta cần tìm ra cách kết hợp tất cả các giá trị kích hoạt thành một đầu ra duy nhất (ví dụ đầu ra cho: có Waldo trong ảnh không). Ta cũng cần quyết định cách tính toán mọi thứ một cách hiệu quả, cách kết hợp các tầng với nhau và liệu có nên sử dụng thật nhiều tầng hẹp hay chỉ một vài tầng rộng. Tất cả những điều này sẽ được giải quyết trong phần còn lại của chương.

6.1.5. Kết luận

- Tính bất biến tịnh tiến của hình ảnh ngụ ý rằng tất cả các mảng nhỏ trong một tấm ảnh đều được xử lý theo cùng một cách.

- Tính cục bộ có nghĩa là chỉ một vùng lân cận nhỏ các điểm ảnh sẽ được sử dụng cho việc tính toán.

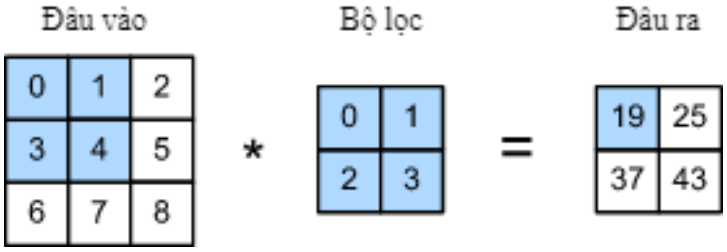
- Các kênh ở đầu vào và đầu ra cho phép việc phân tích các đặc trưng trở nên ý nghĩa hơn.

6.2. Phép Tích chập cho Ảnh

Giờ chúng ta đã hiểu cách các tầng tích chập hoạt động trên lý thuyết, hãy xem chúng hoạt động trong thực tế như thế nào. Dựa vào ý tưởng mạng nơ-ron tích chập là kiến trúc hiệu quả để khám phá cấu trúc của dữ liệu ảnh, chúng tôi vẫn sẽ sử dụng loại dữ liệu này khi lấy ví dụ.

6.2.1. Toán tử Tương quan Chéo

Như ta đã biết, tầng tích chập là cái tên có phần không chính xác, vì phép toán mà chúng biểu diễn là phép tương quan chéo (cross correlation). Trong một tầng tích chập, một mảng đầu vào và một mảng hạt nhân tương quan được kết hợp để tạo ra mảng đầu ra bằng phép toán tương quan chéo. Hãy tạm thời bỏ qua chiều kênh và xem phép toán này hoạt động như thế nào với dữ liệu và biểu diễn ẩn hai chiều. Trong Hình 6.3, đầu vào là một mảng hai chiều với chiều dài 3 và chiều rộng 3. Ta kí hiệu kích thước của mảng là 3×3 hoặc (3, 3). Chiều dài và chiều rộng của hạt nhân đều là 2. Chú ý rằng trong cộng đồng nghiên cứu học sâu, mảng này còn có thể được gọi là hạt nhân tích chập, bộ lọc hay đơn thuần là trọng số của tầng. Kích thước của cửa sổ hạt nhân là chiều dài và chiều rộng của hạt nhân (ở đây là 2×2).



Hình 6.3. Phép tương quan chéo hai chiều. Các phần được tô màu là phần tử đầu tiên của đầu ra cùng với các phần tử của mảng đầu vào và mảng hạt nhân được sử dụng trong phép toán: $0×0+1×1+3×2+4×3=19$

Trong phép tương quan chéo hai chiều, ta bắt đầu với cửa sổ tích chập đặt tại vị trí góc trên bên trái của mảng đầu vào và di chuyển cửa sổ này từ trái sang phải và từ trên xuống dưới. Khi cửa sổ tích chập được đẩy tới một vị trí nhất định, mảng con đầu vào nằm trong cửa sổ đó và mảng hạt nhân được nhân theo từng phần tử, rồi sau đó ta lấy tổng các phần tử trong mảng kết quả để có được một giá trị số vô hướng duy nhất. Giá trị này được ghi vào mảng đầu ra tại vị trí tương ứng. Ở đây, mảng đầu ra có chiều dài 2 và chiều rộng 2, với bốn phần tử được tính bằng phép tương quan chéo hai chiều:

$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$

$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$

$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$

$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$

Lưu ý rằng theo mỗi trục, kích thước đầu ra nhỏ hơn một chút so với đầu vào. Bởi vì hạt nhân có chiều dài và chiều rộng lớn hơn một, ta chỉ có thể tính độ tương quan chéo cho những vị trí mà ở đó hạt nhân nằm hoàn toàn bên trong ảnh, kích thước đầu ra được tính bằng cách lấy đầu vào H×W trừ kích thước của bộ lọc tích chập h×w bằng (H−h+1)×(W−w+1). Điều này xảy ra vì ta cần đủ không gian để 'dịch chuyển' hạt nhân tích chập qua tấm hình (sau này ta sẽ xem làm thế nào để có thể giữ nguyên kích thước bằng cách đệm các số không vào xung quanh biên của hình ảnh sao cho có đủ không gian để dịch chuyển hạt nhân). Kế tiếp, ta lập trình quá trình ở trên trong hàm corr2d. Hàm này nhận mảng đầu vào X với mảng hạt nhân K và trả về mảng đầu ra Y.

Ta có thể xây dựng mảng đầu vào X và mảng hạt nhân K như hình trên để kiểm tra lại kết quả của cách lập trình phép toán tương quan chéo hai chiều vừa rồi.

6.2.2. Tầng Tích chập

Tầng tích chập thực hiện phép toán tương quan chéo giữa đầu vào và hạt nhân, sau đó cộng thêm một hệ số điều chỉnh để có được đầu ra. Hai tham số của tầng tích chập là hạt nhân và hệ số điều chỉnh. Khi huấn luyện mô hình chứa các tầng tích chập, ta thường khởi tạo hạt nhân ngẫu nhiên, giống như cách ta làm với tầng kết nối đầy đủ.

Bây giờ ta đã sẵn sàng lập trình một tầng tích chập hai chiều dựa vào hàm corr2d ta vừa định nghĩa ở trên. Trong hàm khởi tạo __init__, ta khai báo hai tham số của mô hình weight và bias. Hàm tính lượt truyền xuôi forward gọi hàm corr2d và cộng thêm hệ số điều chỉnh. Cũng giống cách gọi phép tương quan chéo h×w, ta cũng gọi các tầng tích chập là phép tích chập h×w.

6.2.3. Phát hiện Biên của Vật thể trong Ảnh

Hãy quan sát một ứng dụng đơn giản của tầng tích chập: phát hiện đường biên của một vật thể trong một bức ảnh bằng cách xác định vị trí các điểm ảnh thay đổi. Đầu tiên, ta dựng một 'bức ảnh' có kích thước là 6×8 điểm ảnh. Bốn cột ở giữa có màu đen (giá trị 0) và các cột còn lại có màu trắng (giá trị 1).

Sau đó, ta tạo một hạt nhân K có chiều cao bằng 1 và chiều rộng bằng 2. Khi thực hiện phép tương quan chéo với đầu vào, nếu hai phần tử cạnh nhau theo chiều ngang có giá trị giống nhau thì đầu ra sẽ bằng 0, còn lại đầu ra sẽ khác không.

Ta đã sẵn sàng thực hiện phép tương quan chéo với các đối số X (đầu vào) và K (hạt nhân). Bạn có thể thấy rằng các vị trí biên trắng đổi thành đen có giá trị 1, còn các vị trí biên đen đổi thành trắng có giá trị -1. Các vị trí còn lại của đầu ra có giá trị 0.

Bây giờ hãy áp dụng hạt nhân này cho chuyển vị của ma trận điểm ảnh. Như kỳ vọng, giá trị tương quan chéo bằng không. Hạt nhân K chỉ có thể phát hiện biên dọc.

6.2.4. Học một Bộ lọc

Việc thiết kế bộ phát hiện biên bằng sai phân hữu hạn [1, -1] thì khá gọn gàng nếu ta biết chính xác đây là những gì cần làm. Tuy nhiên, khi xét tới các bộ lọc lớn hơn và các tầng tích chập liên tiếp, việc chỉ định chính xác mỗi bộ lọc cần làm gì một cách thủ công là bất khả thi.

Bây giờ ta hãy xem liệu có thể học một bộ lọc có khả năng tạo ra Y từX chỉ từ các cặp (đầu vào, đầu ra) hay không. Đầu tiên chúng ta xây dựng một tầng tích chập và khởi tạo một mảng ngẫu nhiên làm bộ lọc. Tiếp theo, trong mỗi lần lặp, ta sẽ sử dụng bình phương sai số để so sánh Y và đầu ra của tầng tích chập, sau đó tính toán gradient để cập nhật trọng số. Để đơn giản, trong tầng tích chập này, ta sẽ bỏ qua hệ số điều chỉnh.

Trước đây ta đã tự xây dựng lớp Conv2D. Tuy nhiên, do ta sử dụng các phép gán một phần tử, Gluon sẽ gặp một số khó khăn khi tính gradient. Thay vào đó, ta sử dụng lớp Conv2D có sẵn của Gluon như sau.

Có thể thấy sai số đã giảm xuống còn khá nhỏ sau 10 lần lặp. Bây giờ hãy xem mảng bộ lọc đã học được.

Thật vậy, mảng bộ lọc học được rất gần với mảng bộ lọc K mà ta tự định nghĩa trước đó.

6.2.5. Tương quan Chéo và Tích chập

Hãy nhớ lại kiến thức của phần trước về mối liên hệ giữa phép tương quan chéo và tích chập. Trong hình trên, ta dễ dàng nhận thấy điều này. Đơn giản chỉ cần lật bộ lọc từ góc dưới cùng bên trái lên góc trên cùng bên phải. Trong trường hợp này, chỉ số trong phép lấy tổng được đảo ngược, nhưng ta vẫn thu được kết quả tương tự. Để thống nhất với các thuật ngữ tiêu chuẩn trong tài liệu học sâu, ta sẽ tiếp tục đề cập đến phép tương quan chéo như là phép tích chập, mặc dù đúng ra chúng hơi khác nhau một chút.

6.2.6. Kết luận

- Về cốt lõi, phần tính toán của tầng tích chập hai chiều là phép tương quan chéo hai chiều. Ở dạng đơn giản nhất, phép tương quan chéo thao tác trên dữ liệu đầu vào hai chiều và bộ lọc, sau đó cộng thêm hệ số điều chỉnh.

- Chúng ta có thể thiết kế bộ lọc để phát hiện các biên trong ảnh.

- Chúng ta có thể học các tham số của bộ lọc từ dữ liệu.

6.3. Đệm và Sải Bước

Trong ví dụ trước, đầu vào có cả chiều dài và chiều rộng cùng bằng 3, cửa sổ hạt nhân tích chập có cả chiều dài và chiều rộng cùng bằng 2, nên ta thu được biểu diễn đầu ra có kích thước 2×2. Nói chung, giả sử kích thước của đầu vào là $n_h \times n_w$ và kích thước của cửa sổ hạt nhân tích chập là $k_h \times k_w$, kích thước của đầu ra sẽ là:

$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$

Do đó, kích thước của đầu ra tầng tích chập được xác định bởi kích thước đầu vào và kích thước cửa sổ hạt nhân tích chập.

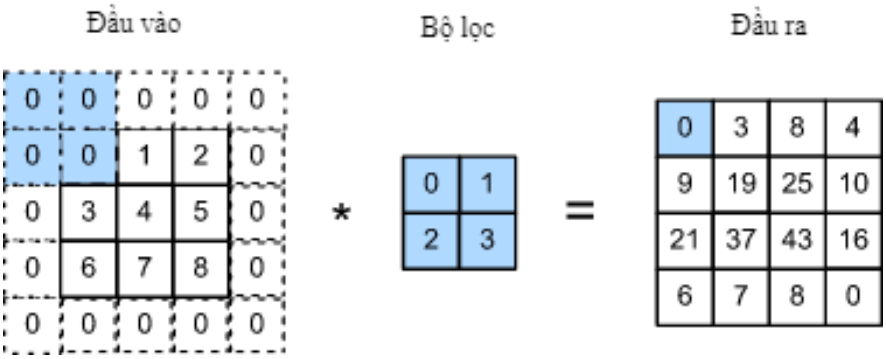
Trong vài trường hợp, ta sẽ kết hợp thêm các kỹ thuật khác cũng có ảnh hưởng tới kích thước của đầu ra, như thêm phần đệm và phép tích chập sải bước. Lưu ý rằng vì các hạt nhân thường có chiều rộng và chiều cao lớn hơn 1 nên sau khi áp dụng nhiều phép tích chập liên tiếp, đầu ra thường có kích thước nhỏ hơn đáng kể so với đầu vào. Nếu ta bắt đầu với một ảnh có 240×240 điểm ảnh và áp dụng 10 tầng tích chập có kích thước 5×5 thì kích thước ảnh này sẽ giảm xuống 200×200 điểm ảnh, 30% của ảnh sẽ bị cắt bỏ và mọi thông tin có ích trên viền của ảnh gốc sẽ bị xóa sạch. Đệm là công cụ phổ biến nhất để xử lý vấn đề này.

Trong những trường hợp khác, ta có thể muốn giảm đáng kể kích thước ảnh, ví dụ như khi độ phân giải của đầu vào quá cao. Phép tích chập sải bước (Strided convolution) là một kỹ thuật phổ biến có thể giúp ích trong trường hợp này.

6.3.1. Đệm

Như mô tả ở trên, một vấn đề rắc rối khi áp dụng các tầng tích chập là việc chúng ta có thể mất một số điểm ảnh trên biên của ảnh. Vì chúng ta thường sử dụng các hạt nhân nhỏ, với một

phép tích chập ta có thể chỉ mất một ít điểm ảnh, tuy nhiên sự mất mát này có thể tích lũy dần khi ta thực hiện qua nhiều tầng tích chập liên tiếp. Một giải pháp đơn giản cho vấn đề này là chèn thêm các điểm ảnh xung quanh đường biên trên bức ảnh đầu vào, nhờ đó làm tăng kích thước sử dụng của bức ảnh. Thông thường, chúng ta thiết lập các giá trị của các điểm ảnh thêm vào là 0. Trong Hình 6.4, ta đệm một đầu vào 3×3, làm tăng kích thước lên thành 5×5. Đầu ra tương ứng sẽ tăng lên thành một ma trận 4×4.



Hình 6.4. Phép tương quan chéo với sải bước 3 theo chiều dài và và 2 theo chiều rộng. Phần tô đậm là các phần tử đầu ra, các phần tử đầu vào và bộ lọc được sử dụng để tính các đầu ra này:

$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8, 0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8, \; 0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$

Nhìn chung, khi sải bước theo chiều cao là s_h và sải bước theo chiều rộng là s_w, kích thước đầu ra là:

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

Nếu đặt $p_h=k_h-1$ và $p_w=k_w-1$, kích thước đầu ra sẽ được thu gọn thành $\lfloor (n_h+s_h-1)/s_h \rfloor \times \lfloor (n_w+s_w-1)/s_w \rfloor$. Hơn nữa, nếu chiều cao và chiều rộng của đầu vào chia hết cho sải bước theo chiều cao và chiều rộng tương ứng thì kích thước đầu ra sẽ là $(n_h/s_h) \times (n_w/s_w)$.

Để đơn giản hóa vấn đề, khi phần đệm theo chiều cao và chiều rộng của đầu vào lần lượt là p_h và p_w, chúng ta sẽ kí hiệu phần đệm là (p_h,p_w). Ở trường hợp đặc biệt khi $p_h=p_w=p$, ta kí hiệu phần đệm là p. Khi sải bước trên chiều cao và chiều rộng lần lượt là s_h và s_w, chúng ta kí hiệu sải bước là (s_h,s_w). Ở trường hợp đặc biệt khi $s_h=s_w=s$, ta kí hiệu sải bước là s. Mặc định, phần đệm là 0 và sải bước là 1. Trên thực tế, ít khi chúng ta sử dụng các giá trị khác nhau cho sải bước hoặc phần đệm, tức ta thường đặt $p_h=p_w$ và $s_h=s_w$.

6.3.3. Kết luận

- Phần đệm có thể tăng chiều cao vào chiều rộng của đầu ra. Nó thường được sử dụng để đầu ra có cùng kích thước với đầu vào.

- Sải bước có thể giảm độ phân giải của đầu ra, ví dụ giảm chiều cao và chiều rộng của đầu ra xuống 1/n chiều cao và chiều rộng của đầu vào (n là một số nguyên lớn hơn 1).

- Đệm và sải bước có thể được dùng để điều chỉnh kích thước chiều của dữ liệu một cách hiệu quả.

6.4. Đa kênh Đầu vào và Đầu ra

Mặc dù chúng ta đã mô tả mỗi tấm ảnh được tạo nên bởi nhiều kênh (channel) (cụ thể, ảnh màu sử dụng hệ màu RGB tiêu chuẩn với các kênh riêng biệt thể hiện lượng màu đỏ, xanh lá và

xanh dương), nhưng cho đến lúc này, ta vẫn đơn giản hóa tất cả các ví dụ tính toán với chỉ một kênh đầu vào và một kênh đầu ra. Điều đó đã cho phép chúng ta coi các đầu vào, các bộ lọc tích chập và các đầu ra như các mảng hai chiều.

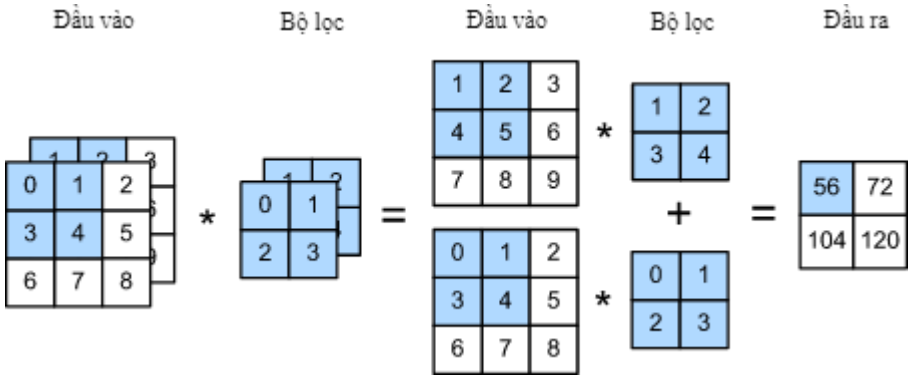
Khi chúng ta thêm các kênh vào hỗn hợp ấy, đầu vào cùng với các lớp biểu diễn ẩn của ta trở thành các mảng ba chiều. Chẳng hạn, mỗi ảnh RGB đầu vào có dạng $3\times h\times w$. Ta xem trục này là chiều kênh, có kích thước là 3. Trong phần này, ta sẽ quan sát sâu hơn vào các bộ lọc tích chập với đầu vào và đầu ra đa kênh.

6.4.1. Đa kênh Đầu vào

Khi dữ liệu đầu vào có nhiều kênh, ta cần xây dựng một bộ lọc tích chập với cùng số kênh đầu vào như dữ liệu nhập, để nó có thể thực hiện tính tương quan chéo với dữ liệu này. Giả sử số kênh dữ liệu đầu vào là c_i, ta sẽ cần số kênh đầu vào của bộ lọc tích chập là c_i. Nếu kích thước cửa sổ của bộ lọc tích chập là $k_h\times k_w$, thì khi $c_i=1$, ta có thể xem bộ lọc tích chập này đơn giản là một mảng hai chiều có kích thước $k_h\times k_w$.

Tuy nhiên, khi $c_i>1$, chúng ta cần một bộ lọc chứa mảng có kích thước $k_h\times k_w$ cho mỗi kênh của đầu vào. Gộp c_i mảng này lại ta được một bộ lọc tích chập kích thước $c_i\times k_h\times k_w$. Vì đầu vào và bộ lọc đều có c_i kênh, ta có thể thực hiện phép tương quan chéo trên từng cặp mảng hai chiều của đầu vào và bộ lọc cho mỗi kênh, rồi cộng kết quả của c_i kênh lại để tạo ra một mảng hai chiều. Đây là kết quả của phép tương quan chéo hai chiều giữa dữ liệu đầu vào đa kênh và kênh bộ lọc tích chập đa đầu vào.

Hình 6.5 minh hoạ một ví dụ về phép tương quan chéo hai chiều với hai kênh đầu vào. Phần tô đậm là phần tử đầu ra đầu tiên cùng các phần tử của mảng đầu vào và bộ lọc được sử dụng trong phép tính đó: $(1\times 1+2\times 2+4\times 3+5\times 4)+(0\times 0+1\times 1+3\times 2+4\times 3)=56$.



Hình 6.5. Phép tính tương quan chéo với hai kênh đầu vào. Phần tô đậm là phần tử đầu ra đầu tiên cùng các phần tử của mảng đầu vào và bộ lọc được sử dụng trong phép tính đó:

$$(1\times 1+2\times 2+4\times 3+5\times 4)+(0\times 0+1\times 1+3\times 2+4\times 3)=56$$

6.4.2. Đa kênh Đầu ra

Cho đến nay, bất kể số lượng kênh đầu vào là bao nhiêu thì ta vẫn luôn kết thúc với chỉ một kênh đầu ra. Tuy nhiên, như đã thảo luận trước đây, hóa ra việc có nhiều kênh ở mỗi tầng là rất cần thiết. Trong các kiến trúc mạng nơ-ron phổ biến nhất, ta thường tăng kích thước chiều kênh khi tiến sâu hơn trong mạng, đồng thời giảm độ phân giải không gian để đánh đổi với chiều kênh sâu hơn này. Theo trực giác, ta có thể xem mỗi kênh tương ứng với một tập các đặc trưng khác nhau. Nhưng thực tế phức tạp hơn một chút so với cách diễn giải theo trực giác này vì các biểu diễn không được học độc lập mà được tối ưu hóa để có ích khi kết hợp với nhau. Vì vậy, có thể việc phát hiện biên sẽ được học bởi một vài kênh thay vì chỉ một kênh duy nhất.

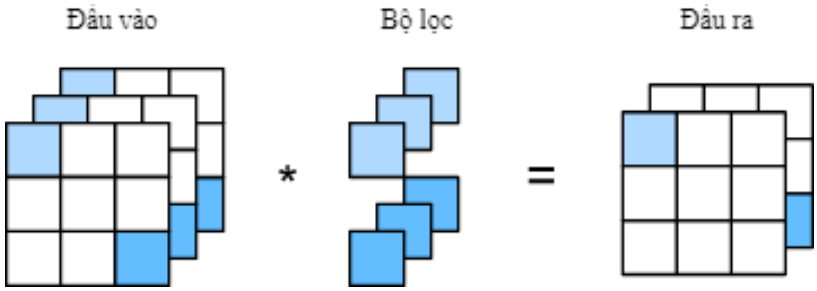
Đặt c_i và c_o lần lượt là số lượng kênh đầu vào và đầu ra, k_h và k_w lần lượt là chiều cao và chiều rộng của bộ lọc. Để có được một đầu ra với nhiều kênh, ta có thể tạo một mảng bộ lọc có kích thước $c_i \times k_h \times k_w$ cho mỗi kênh đầu ra. Ta nối chúng lại dựa trên chiều kênh đầu ra đã biết, sao cho kích thước của bộ lọc tích chập là $c_o \times c_i \times k_h \times k_w$. Trong các phép tính tương quan chéo, kết quả trên mỗi kênh đầu ra được tính từ bộ lọc tích chập tương ứng với kênh đầu ra đó và lấy đầu vào từ tất cả các kênh trong mảng đầu vào.

6.4.3. Tầng Tích chập 1×1

Thoạt nhìn, một phép tích chập 1×1, tức $k_h=k_w=1$, dường như không có nhiều ý nghĩa. Suy cho cùng, một phép tích chập là để tính toán tương quan giữa các điểm ảnh liền kề. Nhưng rõ ràng một phép tích chập 1×1 lại không làm như vậy. Mặc dù vậy, chúng là các phép tính phổ biến đôi khi được sử dụng khi thiết kế các mạng sâu phức tạp. Ta sẽ xem kỹ cách hoạt động của chúng.

Do cửa sổ có kích thước tối thiểu nên so với các tầng tích chập lớn hơn, phép tích chập 1×1 mất đi khả năng nhận dạng các khuôn mẫu chứa các tương tác giữa các phần tử liền kề theo chiều cao và chiều rộng. Phép tích chập 1×1 chỉ xảy ra trên chiều kênh.

Hình 6.6 biểu diễn phép tính tương quan chéo sử dụng bộ lọc tích chập 1×1 với 3 kênh đầu vào và 2 kênh đầu ra. Lưu ý rằng đầu vào và đầu ra có cùng chiều cao và chiều rộng. Mỗi phần tử trong đầu ra là một tổ hợp tuyến tính của các phần tử ở cùng một vị trí trong ảnh đầu vào. Bạn có thể xem tầng tích chập 1×1 như một tầng kết nối đầy đủ được áp dụng lên mỗi vị trí điểm ảnh đơn lẻ để chuyển đổi c_i giá trị đầu vào thành c_o giá trị đầu ra tương ứng. Bởi vì đây vẫn là một tầng tích chập nên các trọng số sẽ được chia sẻ giữa các vị trí điểm ảnh. Do đó, tầng tích chập 1×1 cần tới $c_o \times c_i$ trọng số (cộng thêm các hệ số điều chỉnh).



Hình 6.6. Phép tính tương quan chéo sử dụng bộ lọc tích chập 1×1 với 3 kênh đầu vào và 2 kênh đầu ra. Các đầu vào và các đầu ra có cùng chiều cao và chiều rộng.

6.4.4. Kết luận

- Ta có thể sử dụng nhiều kênh để mở rộng các tham số mô hình của tầng tích chập.

- Tầng tích chập 1×1 khi được áp dụng lên từng điểm ảnh tương đương với tầng kết nối đầy đủ giữa các kênh.

- Tầng tích chập 1×1 thường được sử dụng để điều chỉnh số lượng kênh giữa các tầng của mạng và để kiểm soát độ phức tạp của mô hình.

6.5. Gộp (Pooling)

Khi xử lý ảnh, ta thường muốn giảm dần độ phân giải không gian của các biểu diễn ẩn, tổng hợp thông tin lại để khi càng đi sâu vào mạng, vùng tiếp nhận (ở đầu vào) ảnh hưởng đến mỗi nút ẩn càng lớn.

Nhiệm vụ cuối cùng thường là trả lời một câu hỏi nào đó về toàn bộ tấm ảnh, ví dụ như: trong ảnh có mèo không? Vậy nên các nút của tầng cuối cùng thường cần phải chịu ảnh hưởng của toàn bộ đầu vào. Bằng cách dần gộp thông tin lại để tạo ra các ánh xạ đặc trưng thưa dần, ta sẽ học được một biểu diễn toàn cục, trong khi vẫn có thể giữ nguyên toàn bộ lợi thế đến từ các tầng tích chập xử lý trung gian.

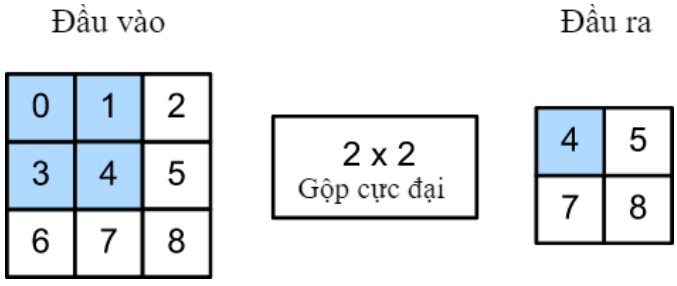
Hơn nữa, khi phát hiện các đặc trưng cấp thấp như cạnh (được thảo luận tại Section 6.2), ta thường muốn cách biểu diễn này bất biến với phép tịnh tiến trong một chừng mực nào đó. Ví dụ, nếu ta lấy ảnh X với một ranh giới rõ rệt giữa màu đen và màu trắng và dịch chuyển toàn bộ tấm ảnh sang phải một điểm ảnh, tức Z[i, j] = X[i, j+1] thì đầu ra cho ảnh mới Z có thể sẽ khác đi rất nhiều. Đường biên đó và các giá trị kích hoạt sẽ đều dịch chuyển sang một điểm ảnh. Trong thực tế, các vật thể hiếm khi xuất hiện chính xác ở cùng một vị trí. Thậm chí với một chân máy ảnh và một vật thể tĩnh, chuyển động của màn trập vẫn có thể làm rung máy ảnh và dịch chuyển tất cả đi một vài điểm ảnh (các máy ảnh cao cấp được trang bị những tính năng đặc biệt nhằm khắc phục vấn đề này).

Trong mục này, chúng tôi sẽ giới thiệu về các tầng gộp, với hai chức năng là giảm độ nhạy cảm của các tầng tích chập đối với vị trí và giảm kích thước của các biểu diễn.

6.5.1. Gộp cực đại và Gộp trung bình

Giống như các tầng tích chập, các toán tử gộp bao gồm một cửa sổ có kích thước cố định được trượt trên tất cả các vùng đầu vào với giá trị sải bước nhất định, tính toán một giá trị đầu ra duy nhất tại mỗi vị trí mà cửa sổ (đôi lúc được gọi là cửa sổ gộp) trượt qua. Tuy nhiên, không giống như phép toán tương quan chéo giữa đầu vào và hạt nhân ở tầng tích chập, tầng gộp không chứa bất kỳ tham số nào (ở đây không có "bộ lọc"). Thay vào đó, các toán tử gộp được định sẵn. Chúng thường tính giá trị cực đại hoặc trung bình của các phần tử trong cửa sổ gộp. Các phép tính này lần lượt được gọi là là gộp cực đại (max pooling) và gộp trung bình (average pooling).

Trong cả hai trường hợp, giống như với toán tử tương quan chéo, ta có thể xem như cửa sổ gộp bắt đầu từ phía trên bên trái của mảng đầu vào và trượt qua mảng này từ trái sang phải và từ trên xuống dưới. Ở mỗi vị trí mà cửa sổ gộp dừng, nó sẽ tính giá trị cực đại hoặc giá trị trung bình của mảng con nằm trong cửa sổ (tùy thuộc vào phép gộp được sử dụng).



Hình 6.7. Gộp cực đại với cửa sổ có kích thước 2×2. Các phần tô đậm thể hiện phần tử đầu ra đầu tiên và phần tử đầu vào được dùng để tính toán: max(0,1,3,4)=4

Mảng đầu ra ở Hình 6.7 phía trên có chiều cao là 2 và chiều rộng là 2. Bốn phần tử của nó được là các giá trị cực đại của hàm max:

$$\begin{aligned} max(0,1,3,4) = 4, \\ max(1,2,4,5) = 5, \\ max(3,4,6,7) = 7, \\ max(4,5,7,8) = 8. \end{aligned}$$

Một tầng gộp với cửa sổ gộp có kích thước p×q được gọi là một tầng gộp p×q. Phép gộp sẽ được gọi là phép gộp p×q.

Hãy cùng quay trở lại với ví dụ nhận diện biên của vật thể được đề cập ở đầu mục. Bây giờ, chúng ta sẽ sử dụng kết quả của tầng tích chập làm giá trị đầu vào cho tầng gộp cực đại 2×2. Đặt giá trị đầu vào của tầng tích chập là X và kết quả của tầng gộp là Y. Dù giá trị của X[i, j] và X[i, j+1] hay giá trị của X[i, j+1] và X[i, j+2] có khác nhau hay không, tất cả giá trị trả về của tầng gộp sẽ là Y[i, j]=1. Nói cách khác, khi sử dụng tầng gộp cực đại 2×2, ta vẫn có thể phát hiện ra khuôn mẫu được nhận diện bởi tầng tích chập nếu nó bị chuyển dịch không nhiều hơn một phần tử theo chiều cao và chiều rộng.

Trong đoạn mã bên dưới, ta lập trình lượt truyền xuôi của tầng gộp trong hàm pool2d. Hàm này khá giống với hàm corr2d trong Section 6.2. Tuy nhiên, hàm này không có bộ lọc nên kết quả đầu ra hoặc là giá trị lớn nhất, hoặc là giá trị trung bình tương ứng của mỗi vùng đầu vào.

6.5.2. Đệm và Sải bước

Cũng giống như các tầng tính chập, các tầng gộp cũng có thể thay đổi kích thước đầu ra. Và cũng như trước, chúng ta có thể thay đổi cách thức hoạt động của tầng gộp để đạt được kích thước đầu ra như mong muốn bằng cách thêm đệm vào đầu vào và điều chỉnh sải bước. Chúng ta có thể minh hoạ cách sử dụng đệm và sải bước trong các tầng gộp thông qua tầng gộp cực đại hai chiều MaxPool2D được cung cấp trong mô-đun nn của thư viện MXNet Gluon. Đầu tiên, chúng ta tạo ra dữ liệu đầu vào kích thước (1, 1, 4, 4), trong đó hai chiều đầu tiên lần lượt là kích thước batch và số kênh.

6.5.3. Với đầu vào Đa Kênh

Khi xử lý dữ liệu đầu vào đa kênh, tầng gộp sẽ áp dụng lên từng kênh một cách riêng biệt thay vì cộng từng phần tử tương ứng của các kênh lại với nhau như tầng tích chập. Điều này có nghĩa là số lượng kênh đầu ra của tầng gộp sẽ giống số lượng kênh đầu vào. Dưới đây, chúng ta sẽ ghép 2 mảng X và X+1 theo chiều kênh để tạo ra đầu vào 2 kênh.

6.5.4. Kết luận

- Với các phần tử đầu vào nằm trong cửa sổ gộp, tầng gộp cực đại sẽ cho đầu ra là giá trị lớn nhất trong số các phần tử đó và tầng gộp trung bình sẽ cho đầu ra là giá trị trung bình của các phần tử.

- Một trong những chức năng chủ yếu của tầng gộp là giảm thiểu sự ảnh hưởng quá mức của vị trí tới tầng tích chập.

- Chúng ta có thể chỉ rõ giá trị của đệm và sải bước cho tầng gộp.

- Tầng gộp cực đại kết hợp với sải bước lớn hơn 1 có thể dùng để giảm độ phân giải.

- Số lượng kênh đầu ra của tầng gộp sẽ bằng số lượng kênh đầu vào tầng gộp đó.

6.6. Mạng Nơ-ron Tích chập (LeNet)

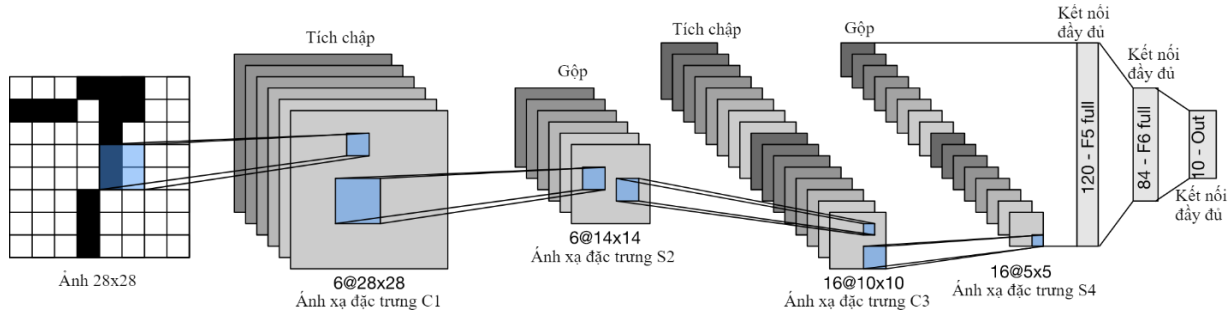
Bây giờ ta đã sẵn sàng kết hợp tất cả các công cụ lại với nhau để triển khai mạng nơ-ron tích chập hoàn chỉnh đầu tiên. Lần đầu làm việc với dữ liệu ảnh, ta đã áp dụng một perceptron đa tầng cho ảnh quần áo trong bộ dữ liệu Fashion-MNIST. Mỗi ảnh trong Fashion-MNIST là một ma trận hai chiều có kích thước 28×28. Để tương thích với đầu vào dạng vector một chiều với độ dài cố định của các perceptron đa tầng, đầu tiên ta trải phẳng từng hình ảnh và thu được các vector có chiều dài 784, trước khi xử lý chúng với một chuỗi các tầng kết nối đầy đủ.

Bây giờ đã có các tầng tích chập, ta có thể giữ nguyên ảnh đầu vào ở dạng không gian hai chiều như ảnh gốc và xử lý chúng với một chuỗi các tầng tích chập liên tiếp. Hơn nữa, vì ta đang sử dụng các tầng tích chập, số lượng tham số cần thiết sẽ giảm đi đáng kể.

Trong phần này, chúng tôi sẽ giới thiệu một trong những mạng nơ-ron tích chập được công bố đầu tiên. Ưu điểm của mạng tích chập được minh hoạ lần đầu bởi Yann Lecun (lúc đó đang nghiên cứu tại AT&T Bell Labs) với ứng dụng nhận dạng các số viết tay trong ảnh-LeNet5. Vào những năm 90, các thí nghiệm của các nhà nghiên cứu với LeNet đã đưa ra bằng chứng thuyết phục đầu tiên về tính khả thi của việc huấn luyện mạng nơ-ron tích chập bằng lan truyền ngược. Mô hình của họ đã đạt được kết quả rất tốt (chỉ có Máy Vector Hỗ trợ — SVM tại thời điểm đó là có thể sánh bằng) và đã được đưa vào sử dụng để nhận diện các chữ số khi xử lý tiền gửi trong máy ATM. Một số máy ATM vẫn chạy các đoạn mã mà Yann và đồng nghiệp Leon Bottou đã viết vào những năm 1990!

6.6.1. LeNet

Một cách đơn giản, ta có thể xem LeNet gồm hai phần: (i) một khối các tầng tích chập; và (ii) một khối các tầng kết nối đầy đủ. Trước khi đi vào các chi tiết cụ thể, hãy quan sát tổng thể mô hình trong Hình 6.8.



Hình 6.8. Dòng dữ liệu trong LeNet 5. Đầu vào là một chữ số viết tay, đầu ra là một xác suất đối với 10 kết quả khả thi.

Các đơn vị cơ bản trong khối tích chập là một tầng tích chập và một lớp gộp trung bình theo sau (lưu ý rằng gộp cực đại hoạt động tốt hơn, nhưng nó chưa được phát minh vào những năm 90). Tầng tích chập được sử dụng để nhận dạng các mẫu không gian trong ảnh, chẳng hạn như các đường cạnh và các bộ phận của vật thể, lớp gộp trung bình phía sau được dùng để giảm số chiều. Khối tầng tích chập tạo nên từ việc xếp chồng các khối nhỏ gồm hai đơn vị cơ bản này. Mỗi tầng tích chập sử dụng hạt nhân có kích thước 5×5 và xử lý mỗi đầu ra với một hàm kích hoạt sigmoid (nhấn mạnh rằng ReLU hiện được biết là hoạt động đáng tin cậy hơn, nhưng chưa được phát minh vào thời điểm đó). Tầng tích chập đầu tiên có 6 kênh đầu ra và tầng tích chập thứ hai tăng độ sâu kênh hơn nữa lên 16.

Tuy nhiên, cùng với sự gia tăng số lượng kênh này, chiều cao và chiều rộng lại giảm đáng kể. Do đó, việc tăng số lượng kênh đầu ra làm cho kích thước tham số của hai tầng tích chập tương tự nhau. Hai lớp gộp trung bình có kích thước 2×2 và sải bước bằng 2 (điều này có nghĩa là chúng không chồng chéo). Nói cách khác, lớp gộp giảm kích thước của các biểu diễn còn một phần tư kích thước trước khi gộp.

Đầu ra của khối tích chập có kích thước được xác định bằng (kích thước batch, kênh, chiều cao, chiều rộng). Trước khi chuyển đầu ra của khối tích chập sang khối kết nối đầy đủ, ta phải trải phẳng từng mẫu trong minibatch. Nói cách khác, ta biến đổi đầu vào 4D thành đầu vào 2D tương thích với các tầng kết nối đầy đủ: nhắc lại, chiều thứ nhất là chỉ số các mẫu trong minibatch và

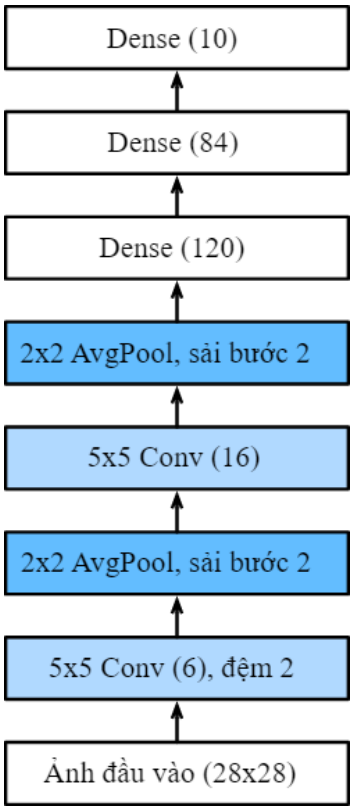
chiều thứ hai là biểu diễn vector phẳng của mỗi mẫu. Khối tầng kết nối đầy đủ của LeNet có ba tầng kết nối đầy đủ, với số lượng đầu ra lần lượt là 120, 84 và 10. Bởi vì ta đang thực hiện bài toán phân loại, tầng đầu ra 10 chiều tương ứng với số lượng các lớp đầu ra khả thi (10 chữ số từ 0 đến 9).

Để thực sự hiểu những gì diễn ra bên trong LeNet có thể đòi hỏi một chút nỗ lực, tuy nhiên bạn có thể thấy bên dưới đây việc lập trình Lenet bằng thư viện học sâu hiện đại rất đơn giản. Một lần nữa, ta sẽ dựa vào lớp Sequential.

So với mạng ban đầu, ta đã thay thế kích hoạt Gauss ở tầng cuối cùng bằng một tầng kết nối đầy đủ thông thường mà thường dễ huấn luyện hơn đáng kể. Ngoại trừ điểm đó, mạng này giống với định nghĩa gốc của LeNet5.

Tiếp theo, ta hãy xem một ví dụ dưới đây. Như trong Hình 6.9, ta đưa vào mạng một mẫu đơn kênh kích thước 28×28 và thực hiện một lượt truyền xuôi qua các tầng và in kích thước đầu ra ở mỗi tầng để hiểu rõ những gì đang xảy ra bên trong.

Xin hãy chú ý rằng, chiều cao và chiều rộng của biểu diễn sau mỗi tầng trong toàn bộ khối tích chập sẽ giảm theo chiều sâu của mạng (so với chiều cao và chiều rộng của biểu diễn ở tầng trước). Tầng tích chập đầu tiên sử dụng một hạt nhân với chiều cao và chiều rộng là 5 rồi đệm thêm 2 đơn vị điểm ảnh để giữ nguyên kích thước đầu vào. Trong khi đó, tầng tích chập thứ hai cũng dùng cùng một hạt nhân với kích thước là 5×5 mà không có sử dụng giá trị đệm thêm vào, dẫn đến việc chiều cao và chiều rộng giảm đi 4 đơn vị điểm ảnh. Ngoài ra, mỗi tầng gộp sẽ làm giảm đi một nửa chiều cao và chiều rộng của đặc trưng ánh xạ đầu vào. Tuy nhiên, khi chúng ta đi theo chiều sâu của mạng, số kênh sẽ tăng lần lượt theo từng tầng. Từ 1 kênh của dữ liệu đầu vào lên tới 6 kênh sau tầng tích chập thứ nhất và 16 kênh sau tầng tích chập thứ hai. Sau đó,giảm số chiều lần lượt qua từng tầng kết nối đầy đủ đến khi trả về một đầu ra có kích thước bằng số lượng lớp của hình ảnh.



Hình 6.9. Kí hiệu vắn tắt cho mô hình LeNet5

6.6.2. Thu thập Dữ liệu và Huấn luyện

Sau khi xây dựng xong mô hình, chúng ta sẽ thực hiện một số thử nghiệm để xem chất lượng của mô hình LeNet. Tập dữ liệu Fashion-MNIST sẽ được dùng trong ví dụ này. Việc phân loại tập Fashion-MNIST sẽ khó hơn so với tập MNIST gốc mặc dù chúng đều chứa các ảnh có cùng kích thước 28×28.

Dù mạng tích chập có thể có số lượng tham số không lớn, chúng vẫn tiêu tốn nhiều tài nguyên tính toán hơn so với perceptron sâu đa tầng. Vì vậy, nếu có sẵn GPU, thì đây là thời điểm thích hợp để dùng nó nhằm tăng tốc quá trình huấn luyện.

Để đánh giá mô hình, chúng ta cần điều chỉnh một chút hàm evaluate_accuracy. Vì toàn bộ tập dữ liệu đang nằm trên CPU, ta cần sao chép nó lên GPU trước khi thực hiện tính toán với mô hình. Việc này được thực hiện thông qua việc gọi hàm as_in_ctx.

Chúng ta cũng cần phải cập nhật hàm huấn luyện để mô hình có thể chạy được trên GPU. Không giống hàm train_epoch_ch3, giờ chúng ta cần chuyển từng batch dữ liệu tới ngữ cảnh được chỉ định (hy vọng là GPU thay vì CPU) trước khi thực hiện lượt truyền xuôi và lượt truyền ngược.

Hàm huấn luyện train_ch6 khá giống với hàm huấn luyện train_ch3. Để đơn giản khi làm việc với mạng nơ-ron có tới hàng chục tầng, hàm train_ch6 chỉ hỗ trợ các mô hình được xây dựng bằng thư viện Gluon. Để khởi tạo bộ tham số của mô hình trên thiết bị đã được chỉ định bởi ctx, ta sẽ sử dụng bộ khởi tạo Xavier. Ta vẫn sử dụng hàm mất mát entropy chéo và thuật toán huấn luyện là phương pháp hạ gradient ngẫu nhiên theo minibatch. Với mỗi epoch tốn khoảng hàng chục giây để chạy, ta sẽ vẽ đường biểu diễn giá trị mất mát huấn luyện với nhiều giá trị chi tiết hơn.

6.6.3. Kết luận

- Mạng nơ-ron tích chập (gọi tắt là ConvNet) là một mạng sử dụng các tầng tích chập.

- Trong ConvNet, ta xen kẽ các phép tích chập, các hàm phi tuyến và các phép gộp.

- Độ phân giải được giảm xuống trước khi tạo một đầu ra thông qua một (hoặc nhiều) tầng kết nối dày đặc.

- LeNet là mạng ConvNet đầu tiên được triển khai thành công.

6.7. Mạng Nơ-ron Tích chập Hiện đại

Trong mục này, từng phần sẽ trình bày một kiến trúc mạng nơ-ron quan trọng mà tại một thời điểm nào đó (có thể cả trong hiện tại) là mô hình nền tảng được sử dụng trong một lượng lớn các nghiên cứu và dự án. Mỗi kiến trúc mạng này thống trị trong một khoảng thời gian, nhiều mạng đã về nhất hoặc nhì tại ImageNet, một cuộc thi được xem là thước đo sự phát triển của học có giám sát trong thị giác máy tính kể từ năm 2010.

Những mô hình này gồm có: AlexNet, mạng quy mô lớn đầu tiên được triển khai để đánh bại các phương pháp thị giác máy tính truyền thống trong một thử thách quy mô lớn; VGG, mạng tận dụng một số lượng các khối được lặp đi lặp lại; Mạng trong Mạng (Network in Network - NiN), một kiến trúc nhằm di chuyển toàn bộ mạng nơ-ron theo từng điểm ảnh ở đầu vào; GoogLeNet sử dụng các phép nối song song giữa các mạng; Mạng phần dư (residual networks - ResNet) là kiến trúc mạng được sử dụng phổ biến nhất hiện nay; và cuối cùng là Mạng tích chập kết nối dày đặc (densely connected network - DenseNet), dù yêu cầu khối lượng tính toán lớn nhưng thường được sử dụng làm mốc chuẩn trong thời gian gần đây.

6.7.1. Mạng Nơ-ron Tích chập Sâu (AlexNet)

Mặc dù đã trở nên nổi tiếng trong cộng đồng thị giác máy tính và học máy sau khi LeNet được giới thiệu, mạng nơ-ron tích chập chưa lập tức thống trị lĩnh vực này. Dẫu LeNet đã đạt được kết quả tốt trên những tập dữ liệu nhỏ, chất lượng và tính khả thi của việc huấn luyện mạng tích chập trên một tập dữ liệu lớn và sát thực tế hơn vẫn là một câu hỏi. Trên thực tế, hầu hết trong khoảng thời gian từ đầu những năm 1990 cho tới năm 2012 với những thành tựu mang tính bước ngoặt, mạng nơ-ron tích chập thường không sánh bằng những phương pháp học máy khác, như Máy Vector hỗ trợ - SVM.

Với thị giác máy tính, phép so sánh này dường như không công bằng. Nguyên nhân là do giá trị đầu vào của mạng tích chập chỉ bao gồm giá trị điểm ảnh thô hoặc đã xử lý thô (như định tâm ảnh (centering)), và kinh nghiệm cho thấy những giá trị thô này không bao giờ nên dùng trực tiếp trong các mô hình truyền thống. Thay vào đó, các hệ thống thị giác máy tính cổ điển dùng những pipeline trích xuất đặc trưng một cách thủ công. Thay vì được học, các đặc trưng được tạo thủ công. Hầu hết những tiến triển trong ngành đều đến từ các ý tưởng thông minh hơn trong tạo đặc trưng và ít để ý hơn tới thuật toán học.

Mặc dù cũng đã có các thiết bị phần cứng tăng tốc độ thực thi mạng nơ-ron vào đầu những năm 1990, chúng vẫn chưa đủ mạnh để triển khai những mạng nơ-ron nhiều kênh, nhiều tầng với số lượng tham số lớn. Ngoài ra, những tập dữ liệu vẫn còn tương đối nhỏ. Thêm vào đó, những thủ thuật chính để huấn luyện mạng nơ-ron bao gồm khởi tạo tham số dựa trên thực nghiệm, các biến thể tốt hơn của hạ gradient ngẫu nhiên, hàm kích hoạt không ép (non-squashing activation functions), và thiếu các kỹ thuật điều chuẩn hiệu quả.

Vì vậy, thay vì huấn luyện các hệ thống đầu-cuối (từ điểm ảnh đến phân loại), các pipeline cổ điển sẽ thực hiện các bước sau:

Thu thập tập dữ liệu đáng chú ý. Trong những ngày đầu, các tập dữ liệu này đòi hỏi các cảm biến đắt tiền (ảnh có 1 triệu điểm ảnh đã được coi là tối tân nhất vào thời điểm đó).

Tiền xử lý tập dữ liệu với các đặc trưng được tạo thủ công dựa trên các kiến thức quang học, hình học, các công cụ phân tích khác và thi thoảng dựa trên các khám phá tình cờ của các nghiên cứu sinh.

Đưa dữ liệu qua một bộ trích chọn đặc trưng tiêu chuẩn như SIFT, hoặc SURF, hay bất kỳ pipeline được tinh chỉnh thủ công nào.

Dùng các kết quả biểu diễn để huấn luyện một bộ phân loại ưa thích, có thể là một mô hình tuyến tính hoặc phương pháp hạt nhân.

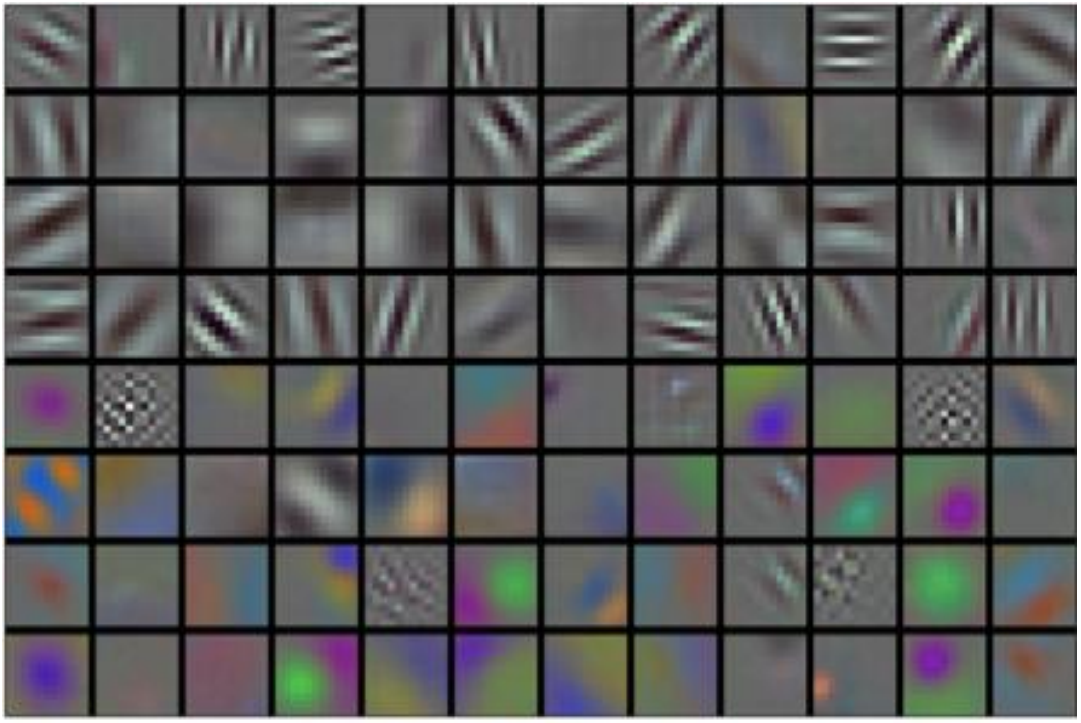
Khi tiếp xúc với những nhà nghiên cứu học máy, bạn sẽ thấy họ tin rằng học máy không những quan trọng mà còn "đẹp" nữa. Bởi lẽ có nhiều lý thuyết tinh vi được đưa ra để chứng minh các tính chất của nhiều bộ phân loại. Và cứ như vậy, lĩnh vực học máy ngày một lớn mạnh, nghiêm ngặt, và hữu dụng hơn bao giờ hết. Tuy nhiên, nếu có dịp thảo luận với một nhà nghiên cứu thị giác máy tính, thì có thể ta lại được nghe một câu chuyện rất khác. Họ sẽ nói rằng sự thật trần trụi trong nhận dạng ảnh là "đặc trưng mới mang tính quyết định tới chất lượng chứ không phải thuật toán học". Những nhà nghiên cứu thị giác máy tính thời đó có lý do để tin rằng chỉ cần một tập dữ liệu hơi lớn hơn, sạch hơn hoặc một pipeline trích xuất đặc trưng tốt hơn một chút sẽ có ảnh hưởng lớn hơn bất kỳ thuật toán học nào.

6.7.1.1. Học Biểu diễn Đặc trưng

Nói một cách khác, tại thời điểm đó phần lớn các nhà nghiên cứu tin rằng phần quan trọng nhất của pipeline là sự biểu diễn. Và cho tới năm 2012 việc biểu diễn vẫn được tính toán một cách máy móc. Trong thực tế, thiết kế và xây dựng một tập các hàm đặc trưng mới, cải thiện kết quả, và viết ra phương pháp thực hiện từng là một phần quan trọng của các bài báo nghiên cứu. SIFT, SURF, HOG, Bags of visual words và các bộ trích chọn đặc trưng tương tự đã chiếm ưu thế vượt trội.

Một nhóm các nhà nghiên cứu bao gồm Yann LeCun, Geoff Hinton, Yoshua Bengio, Andrew Ng, Shun-ichi Amari, và Juergen Schmidhuber, lại có những kế hoạch khác. Họ tin rằng đặc trưng cũng có thể được học. Hơn nữa, họ cũng cho rằng để có được độ phức tạp hợp lý, các đặc trưng nên được phân thành thứ lớp với nhiều tầng học cùng nhau, mỗi tầng có các tham số có thể được huấn luyện. Trong trường hợp ảnh, các tầng thấp nhất có thể dùng để phát hiện biên, màu sắc và đường nét. Thật vậy, [Krizhevsky et al., 2012] giới thiệu một biến thể mới của mạng nơ-ron tích chập đã đạt được hiệu năng xuất sắc trong cuộc thi ImageNet.

Một điều thú vị là ở các tầng thấp nhất của mạng, mô hình đã học được cách trích xuất đặc trưng giống như các bộ lọc truyền thống. Hình 6.10 được tái tạo từ bài báo khoa học trên mô tả các đặc trưng cấp thấp của hình ảnh.



Hình 6.10. Các bộ lọc hình ảnh học được ở tầng đầu tiên của mô hình AlexNet

Các tầng cao hơn của mạng sẽ dựa vào các biểu diễn này để thể hiện các cấu trúc lớn hơn như mắt, mũi, ngọn cỏ, v.v. Thậm chí các tầng cao hơn nữa có thể đại diện cho nguyên một vật thể như con người, máy bay, chó hoặc là đĩa ném. Sau cùng, tầng trạng thái ẩn cuối sẽ học cách biểu diễn cô đọng của toàn bộ hình ảnh để tổng hợp lại nội dung sao cho dữ liệu thuộc các lớp khác nhau có thể được dễ dàng phân biệt.

Mặc dù bước đột phá của các mạng tích chập nhiều tầng xuất hiện vào năm 2012, một nhóm nòng cốt các nhà nghiên cứu đã theo đuổi ý tưởng này, tìm cách học các biểu diễn phân tầng của dữ liệu hình ảnh trong nhiều năm. Có hai yếu tố chính dẫn tới bước đột phá lớn vào năm 2012.

6.7.1.1.1. Yếu tố bị Thiếu - Dữ liệu

Mô hình học sâu với nhiều tầng đòi hỏi phải có một lượng dữ liệu lớn để đạt hiệu quả vượt trội so với các phương pháp truyền thống dựa trên tối ưu lồi (ví dụ: phương pháp tuyến tính và phương pháp nhân). Tuy nhiên, do khả năng lưu trữ của máy tính còn hạn chế, các bộ cảm biến khá đắt đỏ, và ngân sách dành cho việc nghiên cứu tương đối bị thắt chặt vào những năm 1990, phần lớn các nghiên cứu đều dựa trên những bộ dữ liệu nhỏ. Có rất nhiều bài báo nghiên cứu khoa học giải quyết các vấn đề dựa trên bộ dữ liệu tổng hợp UCI, nhiều bộ dữ liệu trong số đó chỉ chứa khoảng vài trăm hoặc (một vài) ngàn hình ảnh được chụp trong điều kiện không tự nhiên với độ phân giải thấp.

Năm 2009, tập dữ liệu ImageNet được ban hành, thách thức các nhà nghiên cứu huấn luyện những mô hình với 1 triệu hình ảnh, trong đó có 1.000 ảnh cho mỗi 1.000 lớp đối tượng khác nhau. Các nhà nghiên cứu giới thiệu tập dữ liệu này, dẫn đầu bởi Fei-Fei Li, đã tận dụng công cụ Tìm kiếm Hình ảnh của Google để lọc sơ bộ các tập dữ liệu hình ảnh lớn cho mỗi lớp và sử dụng dịch vụ cộng đồng (crowdsourcing) Amazon Mechanical Turk để xác thực nhãn cho từng ảnh. Đây là quy mô lớn chưa từng có từ trước đến nay. Cuộc thi đi liền với tập dữ liệu này được đặt tên là ImageNet Challenge và đã thúc đẩy sự phát triển của nghiên cứu thị giác máy tính và học máy, thách thức các nhà nghiên cứu tìm ra mô hình tốt nhất ở quy mô lớn hơn bao giờ hết trong toàn giới học thuật.

6.7.1.1.2. Yếu tố bị Thiếu - Phần cứng

Các mô hình học sâu đòi hỏi rất nhiều chu kỳ tính toán. Quá trình huấn luyện có thể cần hàng trăm epoch, với mỗi vòng lặp yêu cầu đưa dữ liệu qua nhiều tầng nơi các phép toán đại số tuyến tính cồng kềnh được thực thi. Đây là một trong những lý do chính tại sao vào những năm 90 tới đầu những năm 2000, các thuật toán đơn giản dựa trên những mục tiêu tối ưu lồi hiệu quả lại được ưa chuộng hơn.

Bộ xử lý đồ hoạ (GPU) đóng vai trò thay đổi hoàn toàn cuộc chơi khi làm cho việc học sâu trở nên khả thi. Những vi xử lý này đã được phát triển một thời gian dài để tăng tốc độ xử lý đồ họa dành cho các trò chơi máy tính. Cụ thể, chúng được tối ưu hoá cho các phép nhân ma trận - vector 4x4 thông lượng cao, cần thiết cho nhiều tác vụ đồ hoạ. May mắn thay, phép toán này rất giống với phép toán sử dụng trong các tầng tích chập. Trong khoảng thời gian này, hai công ty NVIDIA và ATI đã bắt đầu tối ưu GPU cho mục đích tính toán tổng quát, thậm chí còn tiếp thị chúng dưới dạng GPU đa dụng (General Purpose GPUs - GPGPU).

Để hình dung rõ hơn, hãy cùng xem lại các nhân của bộ vi xử lý CPU hiện đại. Mỗi nhân thì khá mạnh khi chạy ở tần số xung nhịp cao với bộ nhớ đệm lớn (lên đến vài MB ở bộ nhớ đệm L3). Mỗi nhân phù hợp với việc thực hiện hàng loạt các loại chỉ dẫn khác nhau, với các bộ dự báo rẽ nhánh, một pipeline sâu và những tính năng phụ trợ khác cho phép nó có khả năng chạy một lượng lớn các chương trình khác nhau. Tuy nhiên, sức mạnh rõ rệt này cũng có điểm yếu: sản xuất các nhân đa dụng rất đắt đỏ. Chúng đòi hỏi nhiều diện tích cho vi xử lý, cùng cấu trúc hỗ trợ phức tạp (giao diện bộ nhớ, logic bộ nhớ đệm giữa các nhân, kết nối tốc độ cao, v.v.), và chúng tương đối tệ ở bất kỳ tác vụ đơn lẻ nào. Những máy tính xách tay ngày nay chỉ có tới 4 nhân, và thậm chí những máy chủ cao cấp hiếm khi vượt quá 64 nhân, đơn giản bởi vì chúng không hiệu quả về chi phí.

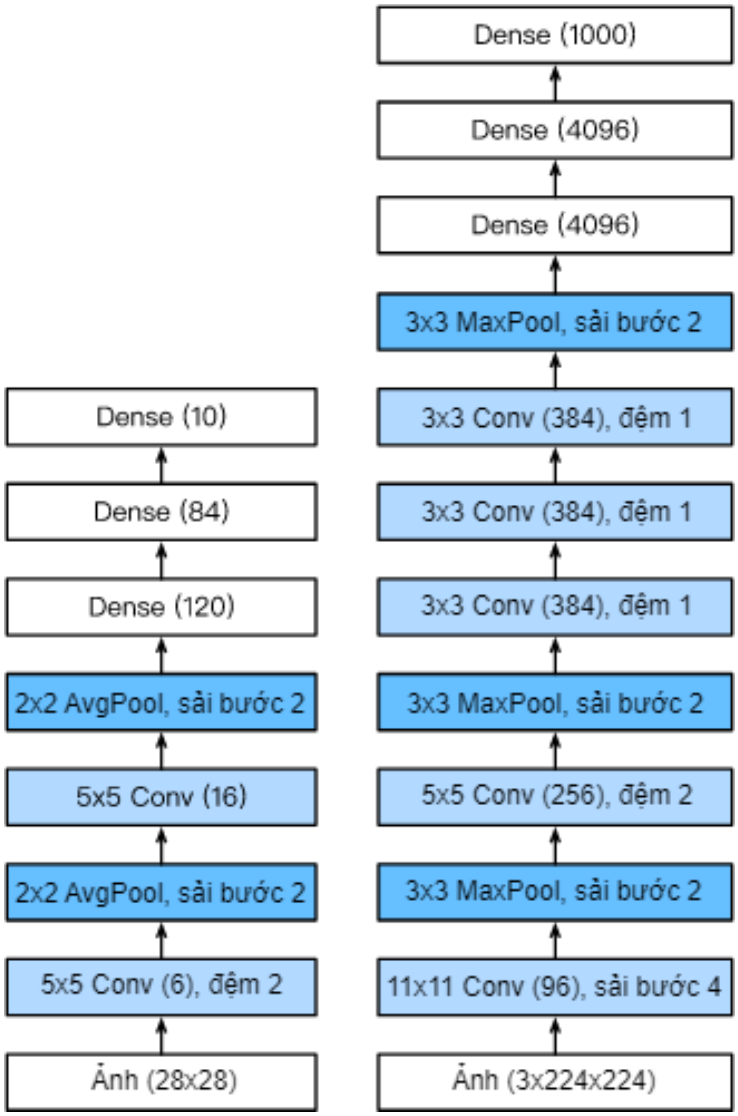
Để so sánh, GPU bao gồm 100-1000 các phần tử xử lý nhỏ (về chi tiết có khác nhau đôi chút giữa NVIDIA, ATI, ARM và các nhà sản xuất khác), thường được gộp thành các nhóm lớn hơn (NVIDIA gọi các nhóm này là luồng (warp)). Mặc dù mỗi nhân thì tương đối yếu, đôi khi thậm chí chạy ở tần số xung nhịp dưới 1GHZ, nhưng số lượng của những nhân này giúp cho GPUs

có tốc độ nhanh hơn rất nhiều so với CPUs. Chẳng hạn, thế hệ Volta mới nhất của NVIDIA có thể thực hiện tới 120 nghìn tỷ phép toán dấu phẩy động (TFlop) với mỗi vi xử lý cho những lệnh chuyên biệt (và lên tới 24 TFlop cho các lệnh đa dụng), trong khi hiệu năng của CPU trong việc thực hiện tính toán số thực dấu phẩy động cho đến nay không vượt quá 1 TFlop. Lý do khá đơn giản: thứ nhất, mức độ tiêu thụ năng lượng có xu hướng tăng theo hàm bậc hai so với tần số xung nhịp. Do đó, với cùng lượng năng lượng để một nhân CPU chạy nhanh gấp 4 lần tốc độ hiện tại (mức tăng thường gặp), chúng ta có thể thay bằng 16 nhân GPU với tốc độ mỗi nhân giảm còn 1/4, cũng sẽ cho kết quả là 16 x 1/4 = 4 lần tốc độ hiện tại. Hơn nữa, các nhân của GPU thì đơn giản hơn nhiều (trên thực tế, trong một khoảng thời gian dài những nhân này thậm chí không thể thực thi được các mã lệnh đa dụng), điều này giúp chúng tiết kiệm năng lượng hơn. Cuối cùng, nhiều phép tính trong quá trình học sâu đòi hỏi bộ nhớ băng thông cao. Và một lần nữa, GPUs tỏa sáng khi độ rộng đường bus của nó lớn hơn ít nhất 10 lần so với nhiều loại CPUs.

Quay trở lại năm 2012. Một bước đột phá lớn khi Alex Krizhevsky và Ilya Sutskever đã xây dựng thành công một mạng nơ-ron tích chập sâu có thể chạy trên phần cứng GPU. Họ nhận ra rằng nút thắt cổ chai khi tính toán trong CNN (phép nhân tích chập và ma trận) có thể được xử lý song song trên phần cứng. Sử dụng hai card đồ họa NVIDIA GTX 580s với 3GB bộ nhớ, họ đã xây dựng các phép toán tích chập nhanh. Phần mã nguồn cuda-convnet được xem là ngòi nổ cho sự phát triển vượt bậc của học sâu ngày nay.

6.7.1.2. Mạng AlexNet

Mạng AlexNet được giới thiệu vào năm 2012, được đặt theo tên của Alex Krizhevsky, tác giả thứ nhất của bài báo đột phá trong phân loại ImageNet [Krizhevsky et al., 2012]. Mạng AlexNet bao gồm 8 tầng mạng nơ-ron tích chập, đã chiến thắng cuộc thi ImageNet Large Scale Visual Recognition Challenge năm 2012 với cách biệt không tưởng. AlexNet lần đầu tiên đã chứng minh được rằng các đặc trưng thu được bởi việc học có thể vượt qua các đặc trưng được thiết kế thủ công, phá vỡ định kiến trước đây trong nghiên cứu thị giác máy tính. Cấu trúc mạng AlexNet và LeNet rất giống nhau, như Hình 6.11 đã minh họa. Lưu ý rằng chúng tôi cung cấp một phiên bản AlexNet được sắp xếp hợp lý, loại bỏ một số điểm thiết kế có từ năm 2012 với mục đích làm cho mô hình phù hợp với hai GPU dung lượng nhỏ mà bây giờ đã không còn cần thiết.



Hình 6.11. LeNet (trái) và AlexNet (phải)

Các triết lý thiết kế của AlexNet và LeNet rất giống nhau, nhưng cũng có những khác biệt đáng kể. Đầu tiên, AlexNet sâu hơn nhiều so với LeNet5. AlexNet có tám tầng gồm: năm tầng tích chập, hai tầng ẩn kết nối đầy đủ, và một tầng đầu ra kết nối đầy đủ. Thứ hai, AlexNet sử dụng ReLU thay vì sigmoid làm hàm kích hoạt. Chúng tôi sẽ đi sâu hơn vào chi tiết ở dưới đây.

6.7.1.2.1. Kiến trúc

Trong tầng thứ nhất của AlexNet, kích thước cửa sổ tích chập là 11×11. Vì hầu hết các ảnh trong ImageNet đều có chiều cao và chiều rộng lớn gấp hơn mười lần so với các ảnh trong MNIST, các vật thể trong dữ liệu ImageNet thường có xu hướng chiếm nhiều điểm ảnh hơn. Do đó, ta cần sử dụng một cửa sổ tích chập lớn hơn để xác định được các vật thể này. Kích thước cửa sổ tích chập trong tầng thứ hai được giảm xuống còn 5×5 và sau đó là 3×3. Ngoài ra, theo sau các tầng chập thứ nhất, thứ hai và thứ năm là các tầng gộp cực đại với kích thước cửa sổ là 3×3 và sải bước bằng 2. Hơn nữa, số lượng các kênh tích chập trong AlexNet nhiều hơn gấp mười lần so với LeNet.

Sau tầng tích chập cuối cùng là hai tầng kết nối đầy đủ với 4096 đầu ra. Hai tầng này tạo ra tới gần 1 GB các tham số mô hình. Do các GPU thế hệ trước bị giới hạn về bộ nhớ, phiên bản gốc của AlexNet sử dụng thiết kế luồng dữ liệu kép cho hai GPU, trong đó mỗi GPU chỉ phải chịu trách nhiệm lưu trữ và tính toán cho một nửa mô hình. May mắn thay, hiện nay các GPU có bộ

nhớ tương đối dồi dào, vì vậy ta hiếm khi cần phải chia nhỏ mô hình trên các GPU (phiên bản mô hình AlexNet của ta khác với bài báo ban đầu ở khía cạnh này).

6.7.1.2.2. Các hàm Kích hoạt

AlexNet đã thay hàm kích hoạt sigmoid bằng hàm kích hoạt ReLU đơn giản hơn. Một mặt là giảm việc tính toán, bởi ReLu không có phép lũy thừa như trong hàm kích hoạt sigmoid. Mặt khác, hàm kích hoạt ReLU giúp cho việc huấn luyện mô hình trở nên dễ dàng hơn khi sử dụng các phương thức khởi tạo tham số khác nhau. Điều này là do khi đầu ra của hàm kích hoạt sigmoid rất gần với 0 hoặc 1 thì gradient sẽ gần như bằng 0, vì vậy khiến cho lan truyền ngược không thể tiếp tục cập nhật một số tham số mô hình. Ngược lại, gradient của hàm kích hoạt ReLU trong khoảng dương luôn bằng 1. Do đó, nếu các tham số mô hình không được khởi tạo đúng cách thì hàm sigmoid có thể có gradient gần bằng 0 trong khoảng dương, dẫn đến việc mô hình không được huấn luyện một cách hiệu quả.

6.7.1.2.3. Kiểm soát năng lực mô hình và Tiền xử lý

AlexNet kiểm soát năng lực của tầng kết nối đầy đủ bằng cách áp dụng dropout (Section 4.6), trong khi LeNet chỉ sử dụng suy giảm trọng số. Để tăng cường dữ liệu thì trong quá trình huấn luyện, AlexNet đã bổ sung rất nhiều kỹ thuật tăng cường hình ảnh chẳng hạn như lật, cắt hay thay đổi màu sắc. Điều này giúp cho mô hình trở nên mạnh mẽ hơn, cùng với đó kích thước dữ liệu lớn hơn giúp làm giảm hiện tượng quá khớp một cách hiệu quả.

6.7.1.3. Đọc Tập dữ liệu

Mặc dù AlexNet sử dụng ImageNet trong bài báo nêu trên, ở đây ta sẽ sử dụng Fashion-MNIST vì ngay cả với một GPU hiện đại thì việc huấn luyện một mô hình trên ImageNet có thể mất nhiều giờ hoặc nhiều ngày để hội tụ. Một trong những vấn đề khi áp dụng AlexNet trực tiếp trên Fashion-MNIST là các ảnh trong tập dữ liệu này có độ phân giải thấp hơn (28×28 điểm ảnh) so với các ảnh trong ImageNet. Để có thể tiến hành được thử nghiệm, ta sẽ nâng kích thước ảnh lên 224×224 (nói chung đây không phải là một giải pháp thông minh, nhưng ta cần làm việc này để có thể sử dụng kiến trúc gốc của AlexNet). Việc thay đổi kích thước có thể được thực hiện thông qua đối số resize trong hàm load_data_fashion_mnist.

6.7.1.4. Huấn luyện

Bây giờ, ta có thể bắt đầu quá trình huấn luyện AlexNet. So với LeNet, thay đổi chính ở đây là việc sử dụng tốc độ học nhỏ hơn và quá trình huấn luyện chậm hơn nhiều do tính chất sâu và rộng hơn của mạng, đồng thời do độ phân giải hình ảnh cao hơn và việc tính toán các phép tích chập tốn kém hơn.

6.7.1.5. Kết luận

- AlexNet có cấu trúc tương tự như LeNet, nhưng sử dụng nhiều tầng tích chập hơn với không gian tham số lớn hơn để khớp tập dữ liệu ImageNet với kích thước lớn.

- Ngày nay AlexNet đã bị vượt qua bởi các kiến trúc hiệu quả hơn nhiều nhưng nó là một bước quan trọng để đi từ các mạng nông đến các mạng sâu được sử dụng ngày nay.

- Mặc dù có vẻ như chỉ tốn thêm một vài dòng trong mã nguồn của AlexNet so với LeNet, nhưng cộng đồng học thuật đã phải mất nhiều năm để đón nhận sự thay đổi khái niệm này và tận dụng những kết quả thực nghiệm tuyệt vời của nó. Một phần cũng do sự thiếu thốn của các công cụ tính toán hiệu quả.

- Dropout, ReLU và tiền xử lý là những bước quan trọng khác để đạt được kết quả xuất sắc trong các bài toán thị giác máy tính.

6.7.2. Mạng sử dụng Khối (VGG)

Mặc dù AlexNet đã chứng minh rằng các mạng nơ-ron tích chập có thể đạt được kết quả tốt, nó lại không cung cấp một khuôn mẫu chung để định hướng nghiên cứu sau này trong việc thiết kế các mạng mới. Trong các phần tiếp theo, chúng tôi sẽ giới thiệu một số khái niệm dựa trên thực nghiệm được sử dụng rộng rãi khi thiết kế mạng học sâu.

Sự phát triển trong lĩnh vực này có nét tương đồng tiến triển trong ngành thiết kế vi xử lý, chuyển từ việc sắp đặt các bóng bán dẫn, đến các phần tử logic và các khối logic.

Tương tự như vậy, việc thiết kế kiến trúc các mạng nơ-ron đã phát triển theo hướng ngày một trừu tượng hơn. Điển hình là việc các nhà nghiên cứu đã thay đổi suy nghĩ từ quy mô các nơ-ron riêng lẻ sang các tầng, và giờ đây là các khối chứa các tầng lặp lại theo khuôn mẫu.

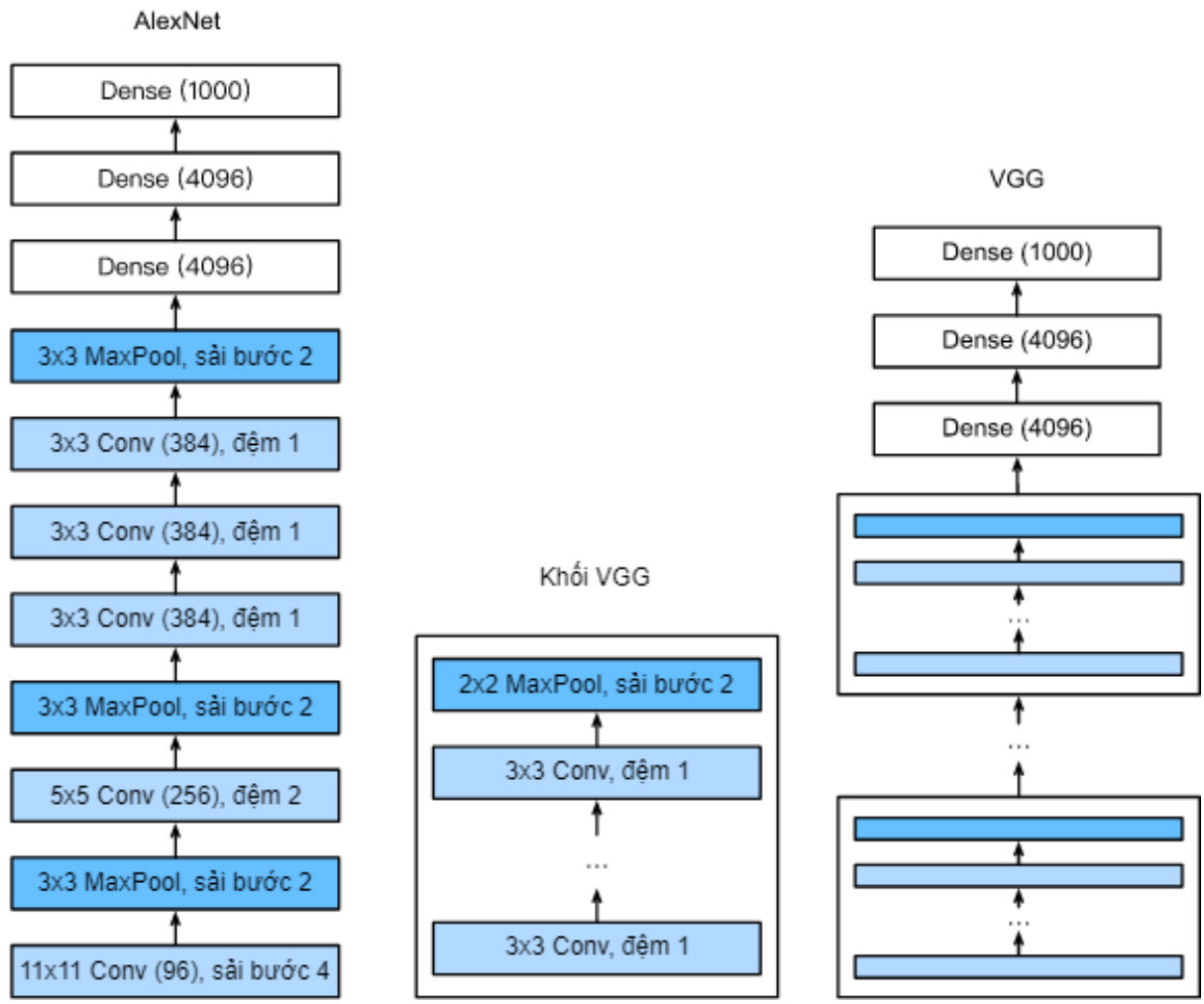
Ý tưởng sử dụng các khối lần đầu xuất hiện trong mạng VGG, được đặt theo tên của nhóm VGG thuộc Đại học Oxford. Sử dụng bất kỳ các framework học sâu hiện đại nào với vòng lặp và chương trình con để xây dựng các cấu trúc lặp lại này là tương đối dễ dàng.

6.7.2.1. Khối VGG

Khối cơ bản của mạng tích chập cổ điển là một chuỗi các tầng sau đây: (i) một tầng tích chập (với phần đệm để duy trì độ phân giải), (ii) một tầng phi tuyến như ReLU, (iii) một tầng gộp như tầng gộp cực đại. Một khối VGG gồm một chuỗi các tầng tích chập, tiếp nối bởi một tầng gộp cực đại để giảm chiều không gian. Trong bài báo gốc của VGG [Simonyan & Zisserman, 2014], tác giả sử dụng tích chập với các hạt nhân 3×3 và tầng gộp cực đại 2×2 với sải bước bằng 2 (giảm một nửa độ phân giải sau mỗi khối). Trong mã nguồn dưới đây, ta định nghĩa một hàm tên vgg_block để tạo một khối VGG. Hàm này nhận hai đối số num_convs và num_channels tương ứng lần lượt với số tầng tích chập và số kênh đầu ra.

6.7.2.2. Mạng VGG

Giống như AlexNet và LeNet, mạng VGG có thể được phân chia thành hai phần: phần đầu tiên bao gồm chủ yếu các tầng tích chập và tầng gộp, còn phần thứ hai bao gồm các tầng kết nối đầy đủ. Phần tích chập của mạng gồm các mô-đun vgg_block kết nối liên tiếp với nhau. Trong Hình 6.12, biến conv_arch bao gồm một danh sách các tuples (một tuple cho mỗi khối), trong đó mỗi tuple chứa hai giá trị: số lượng tầng tích chập và số kênh đầu ra, cũng chính là những đối số cần thiết để gọi hàm vgg_block. Mô-đun kết nối đầy đủ giống hệt với mô-đun tương ứng trong AlexNet.



Hình 6.12. Thiết kế mạng từ các khối cơ bản

Mạng VGG gốc có 5 khối tích chập, trong đó hai khối đầu tiên bao gồm một tầng tích chập ở mỗi khối, ba khối còn lại chứa hai tầng tích chập ở mỗi khối. Khối đầu tiên có 64 kênh đầu ra, mỗi khối tiếp theo nhân đôi số kênh đầu ra cho tới khi đạt giá trị 512. Vì mạng này sử dụng 8 tầng tích chập và 3 tầng kết nối đầy đủ nên nó thường được gọi là VGG-11.

6.7.2.3. Huấn luyện Mô hình

Vì VGG-11 thực hiện nhiều tính toán hơn AlexNet, ta sẽ xây dựng một mạng với số kênh nhỏ hơn. Như vậy vẫn là quá đủ để huấn luyện trên bộ dữ liệu Fashion-MNIST. Ngoại trừ việc sử dụng tốc độ học lớn hơn một chút, quy trình huấn luyện mô hình này tương tự như của AlexNet trong phần trước.

6.7.2.4. Kết luận

- Mạng VGG-11 được xây dựng bằng cách tái sử dụng các khối tích chập. Các mô hình VGG khác nhau có thể được định nghĩa bằng cách thay đổi số lượng các tầng tích chập và số kênh đầu ra ở mỗi khối.

- Việc sử dụng các khối giúp ta định nghĩa mạng bằng các đoạn mã nguồn ngắn gọn và thiết kế các mạng phức tạp một cách hiệu quả hơn.

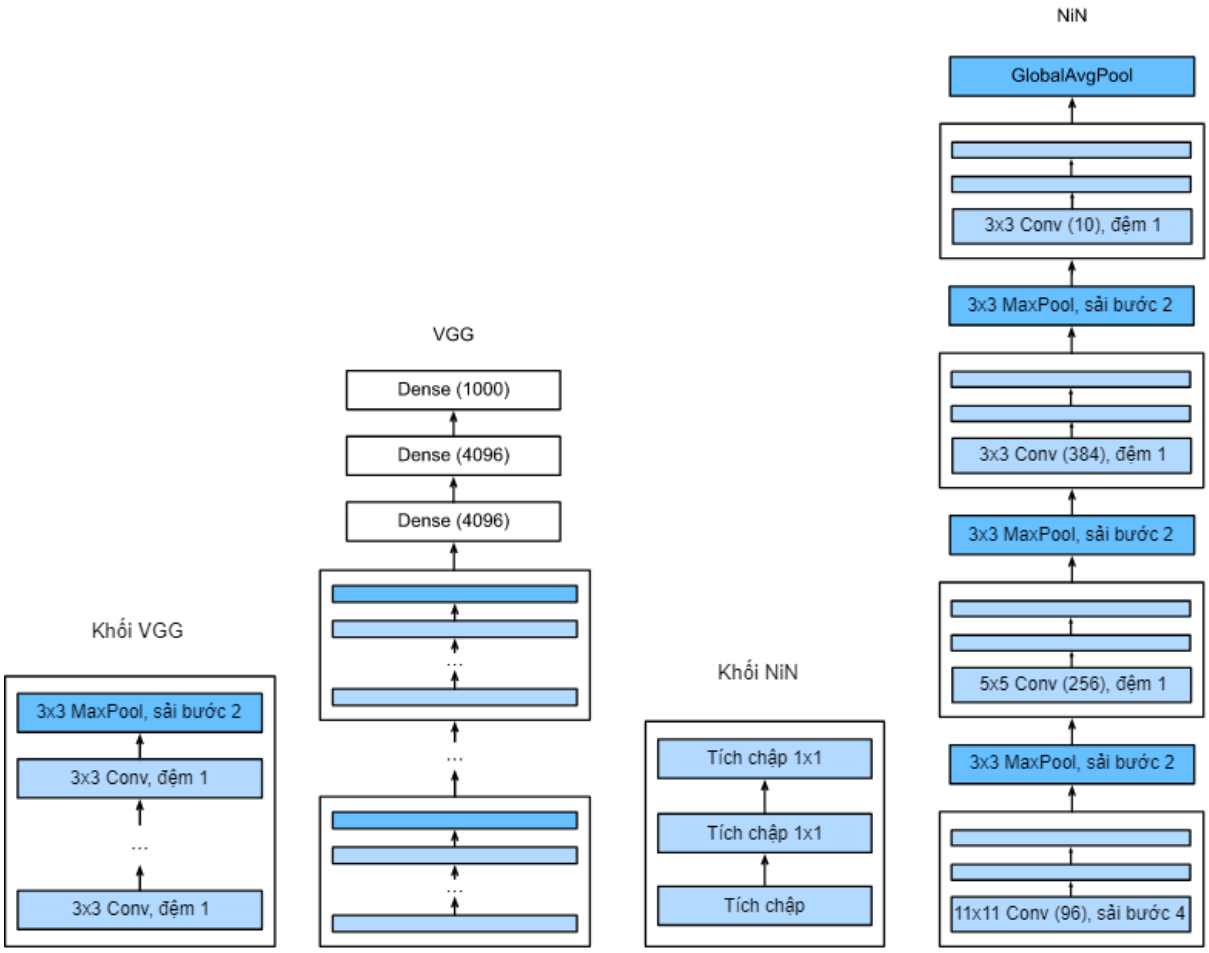
- Thử nghiệm nhiều kiến trúc khác nhau, Simonyan và Zisserman đã phát hiện rằng mạng có cửa sổ tích chập hẹp (như 3×3) và nhiều tầng cho hiệu quả cao hơn mạng có cửa sổ tích chập rộng nhưng ít tầng.

6.7.3. Mạng trong Mạng (Network in Network - NiN)

LeNet, AlexNet và VGG đều có chung một khuôn mẫu thiết kế: trích xuất các đặc trưng khai thác cấu trúc không gian thông qua một chuỗi các phép tích chập và các tầng gộp, sau đó hậu xử lý các biểu diễn thông qua các tầng kết nối đầy đủ. Những cải tiến so với LeNet của AlexNet và VGG chủ yếu nằm ở việc mở rộng và tăng chiều sâu hai mô-đun này. Một lựa chọn khác là ta có thể sử dụng các tầng kết nối đầy đủ ngay từ giai đoạn trước. Tuy nhiên, việc tùy tiện sử dụng các tầng kết nối dày đặc có thể làm mất đi cấu trúc không gian của biểu diễn. Dùng các khối của Mạng trong Mạng (Network in Network - NiN) là một giải pháp thay thế khác. Ý tưởng này được đề xuất trong [Lin et al., 2013] dựa trên một thay đổi rất đơn giản — sử dụng MLP trên các kênh cho từng điểm ảnh riêng biệt.

6.7.3.1. Khối NiN

Hãy nhớ lại rằng đầu vào và đầu ra của các tầng tích chập là các mảng bốn chiều với các trục tương ứng với batch, kênh, chiều cao và chiều rộng. Đầu vào và đầu ra của các tầng kết nối đầy đủ thường là các mảng hai chiều tương ứng với batch và các đặc trưng. Ý tưởng chính của NiN là áp dụng một tầng kết nối đầy đủ tại mỗi vị trí điểm ảnh (theo chiều cao và chiều rộng). Nếu trọng số tại mỗi vị trí không gian được chia sẻ với nhau, ta có thể coi đây là một tầng chập 1×1 (như được mô tả trong Section 6.4) hoặc như một tầng kết nối đầy đủ được áp dụng độc lập trên từng vị trí điểm ảnh. Nói theo một cách khác, ta có thể coi từng phần tử trong chiều không gian (chiều cao và chiều rộng) là tương đương với một mẫu và mỗi kênh tương đương với một đặc trưng. Hình 6.13 minh họa sự khác biệt chính về cấu trúc giữa NiN và AlexNet, VGG cũng như các mạng khác.



Hình 6.13. Hình bên trái biểu diễn cấu trúc mạng của AlexNet và VGG, và hình bênh phải biểu diễn cấu trúc mạng của NiN

Khối NiN bao gồm một tầng tích chập theo sau bởi hai tầng tích chập 1×1 hoạt động như các tầng kết nối đầy đủ trên điểm ảnh và sau đó là hàm kích hoạt ReLU. Kích thước cửa sổ tích chập của tầng thứ nhất thường được định nghĩa bởi người dùng. Kích thước cửa sổ tích chập ở các tầng tiếp theo được cố định bằng 1×1.

6.7.3.2. Mô hình NiN

Cấu trúc mạng NiN gốc được đề xuất ngay sau và rõ ràng lấy cảm hứng từ mạng Alexnet. NiN sử dụng các tầng tích chập có kích thước cửa sổ 11×11, 5×5, 3×3, và số lượng các kênh đầu ra tương ứng giống với AlexNet. Mỗi khối NiN theo sau bởi một tầng gộp cực đại với sải bước 2 và kích thước cửa sổ 3×3.

Một điểm khác biệt đáng chú ý so với AlexNet là NiN tránh hoàn toàn việc sử dụng các kết nối dày đặc. Thay vào đó, mạng này sử dụng các khối NiN với số kênh đầu ra bằng với số lớp nhãn, theo sau bởi một tầng gộp trung bình toàn cục, tạo ra một vector logit. Một lợi thế của thiết kế NiN là giảm được các tham số cần thiết của mô hình một cách đáng kể. Tuy nhiên, trong thực tế, cách thiết kế này đôi lúc đòi hỏi tăng thời gian huấn luyện mô hình.

6.7.3.3. Thu thập Dữ liệu và Huấn luyện

Như thường lệ, ta sẽ sử dụng Fashion-MNIST để huấn luyện mô hình. Quá trình huấn luyện NiN cũng tương tự như AlexNet và VGG, nhưng thường sử dụng tốc độ học lớn hơn.

6.7.3.4. Kết luận

- NiN sử dụng các khối được cấu thành từ một tầng tích chập thông thường và nhiều tầng tích chập 1×1. Kỹ thuật này có thể dùng trong các khối tích chập để tăng tính phi tuyến trên điểm ảnh.

- NiN loại bỏ các tầng kết nối đầy đủ và thay thế chúng bằng phép gộp trung bình toàn cục (nghĩa là tính trung bình cộng từ tất cả các vị trí) sau khi giảm số lượng kênh xuống bằng với số lượng đầu ra mong muốn (ví dụ: 10 kênh cho Fashion-MNIST).

- Việc bỏ đi các các tầng dày đặc giúp làm giảm hiện tượng quá khớp. NiN có số lượng tham số ít hơn đáng kể.

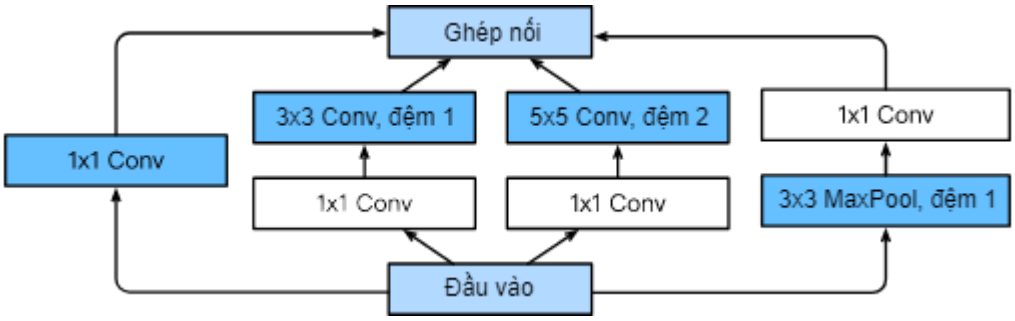
- Thiết kế của NiN đã ảnh hưởng đến thiết kế của nhiều mạng nơ-ron tích chập sau này.

6.7.4. Mạng nối song song (GoogLeNet)

Vào năm 2014, bài báo khoa học [Szegedy et al., 2015] đã giành chiến thắng ở cuộc thi ImageNet, bằng việc đề xuất một cấu trúc kết hợp những điểm mạnh của mô hình NiN và mô hình chứa các khối lặp lại. Bài báo này tập trung giải quyết câu hỏi: kích thước nào của bộ lọc tích chập là tốt nhất. Suy cho cùng, các mạng phổ biến trước đây chọn kích thước bộ lọc từ nhỏ như 1×1 tới lớn như 11×11. Một góc nhìn sâu sắc trong bài báo này là đôi khi việc kết hợp các bộ lọc có kích thước khác nhau có thể sẽ hiệu quả. Trong phần này, chúng tôi sẽ giới thiệu mô hình GoogLeNet, bằng việc trình bày một phiên bản đơn giản hơn một chút so với phiên bản gốc—bỏ qua một số tính năng đặc biệt trước đây được thêm vào nhằm ổn định quá trình huấn luyện nhưng hiện nay không cần thiết nữa do đã có các thuật toán huấn luyện tốt hơn.

6.7.4.1. Khối Inception

Khối tích chập cơ bản trong mô hình GoogLeNet được gọi là Inception, nhiều khả năng được đặt tên dựa theo câu nói "Chúng ta cần đi sâu hơn" ("We Need To Go Deeper") trong bộ phim Inception, sau này đã tạo ra một trào lưu lan rộng trên internet.



Hình 6.14. Cấu trúc của khối Inception

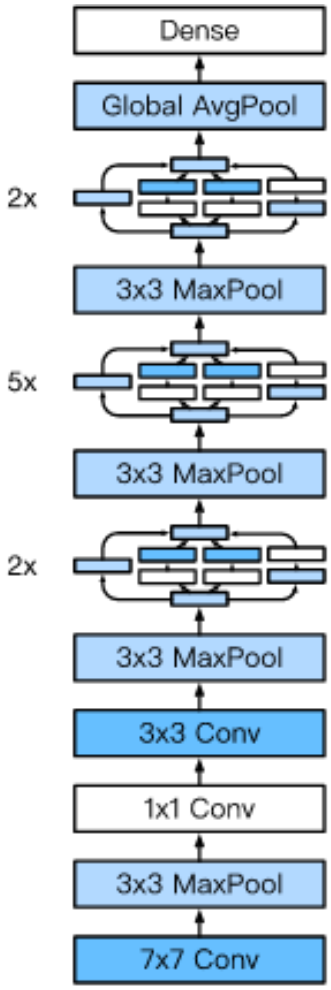
Như mô tả ở hình trên, khối inception bao gồm bốn nhánh song song với nhau. Ba nhánh đầu sử dụng các tầng tích chập với kích thước cửa sổ trượt lần lượt là 1×1, 3×3, và 5×5 để trích xuất thông tin từ các vùng không gian có kích thước khác nhau. Hai nhánh giữa thực hiện phép tích chập 1×1 trên dữ liệu đầu vào để giảm số kênh đầu vào, từ đó giảm độ phức tạp của mô hình. Nhánh thứ tư sử dụng một tầng gộp cực đại kích thước 3×3, theo sau là một tầng tích chập 1×1 để thay đổi số lượng kênh. Cả bốn nhánh sử dụng phần đệm phù hợp để đầu vào và đầu ra của khối có cùng chiều cao và chiều rộng. Cuối cùng, các đầu ra của mỗi nhánh sẽ được nối lại theo chiều kênh để tạo thành đầu ra của cả khối. Các tham số thường được tinh chỉnh của khối Inception là số lượng kênh đầu ra mỗi tầng.

Để hiểu trực quan tại sao mạng này hoạt động tốt, hãy cùng tìm hiểu sự kết hợp của các bộ lọc. Chúng khám phá hình ảnh trên các vùng có kích thước khác nhau. Tức là những chi tiết ở

những mức độ khác nhau sẽ được nhận diện một cách hiệu quả bằng các bộ lọc khác nhau. Đồng thời, chúng ta có thể phân bổ số lượng tham số khác nhau cho những vùng có phạm vi khác nhau (ví dụ: nhiều tham số hơn cho vùng phạm vi nhỏ nhưng không bỏ qua hoàn toàn vùng phạm vi lớn).

6.7.4.2. Mô hình GoogLeNet

Như trình bày ở Hình 6.15, mô hình GoogLeNet sử dụng tổng cộng 9 khối inception và tầng gộp trung bình toàn cục xếp chồng lên nhau. Phép gộp cực đại giữa các khối inception có tác dụng làm giảm kích thước chiều. Phần đầu tiên của GoogleNet giống AlexNet và LeNet, có các khối xếp chồng lên nhau kế thừa từ thiết kế của VGG và phép gộp trung bình toàn cục giúp tránh phải sử dụng nhiều tầng kết nối đầy đủ liên tiếp ở cuối. Cấu trúc của mô hình được mô tả như dưới đây.



Hình 6.15. Mô hình GoogLeNet đầy đủ

6.7.4.3. Thu thập Dữ liệu và Huấn luyện

Vẫn như trước, chúng ta sử dụng tập dữ liệu Fashion-MNIST để huấn luyện mô hình. Chúng ta chuyển đổi độ phân giải hình ảnh thành 96×96 điểm ảnh trước khi bắt đầu quá trình huấn luyện.

6.7.4.4. Kết luận

- Khối Inception tương đương với một mạng con với bốn nhánh. Nó trích xuất thông tin một cách song song thông qua các tầng tích chập với kích thước cửa sổ khác nhau và các tầng gộp cực đại.

- Phép tích chập 1×1 giảm số kênh ở mức độ điểm ảnh. Phép gộp cực đại giảm độ phân giải.

- Trong GoogLeNet, nhiều khối Inception với thiết kế khéo léo được nối với các tầng khác theo chuỗi. Tỷ lệ số kênh trong khối Inception được xác định dựa vào nhiều kết quả thử nghiệm trên tập dữ liệu ImageNet.

- Mô hình GoogLeNet, cũng như các phiên bản kế tiếp của nó, là một trong những mô hình hiệu quả nhất trên ImageNet, với độ chính xác tương tự trên tập kiểm tra nhưng độ phức tạp tính toán lại thấp hơn.

6.7.5. Chuẩn hoá theo batch

Huấn luyện mạng nơ-ron sâu không hề đơn giản, để chúng hội tụ trong khoảng thời gian chấp nhận được là một câu hỏi khá hóc búa. Trong phần này, chúng ta giới thiệu chuẩn hóa theo batch (Batch Normalization - BN) [Ioffe & Szegedy, 2015], một kỹ thuật phổ biến và hiệu quả nhằm tăng tốc độ hội tụ của mạng học sâu một cách ổn định. Cùng với các khối phần dư (residual block)— BN giúp dễ dàng hơn trong việc huấn luyện mạng học sâu với hơn 100 tầng.

6.7.5.1. Huấn luyện mạng học sâu

Để thấy mục đích của việc chuẩn hóa theo batch, hãy cùng xem xét lại một vài vấn đề phát sinh trên thực tế khi huấn luyện các mô hình học máy và đặc biệt là mạng nơ-ron.

Những lựa chọn tiền xử lý dữ liệu khác nhau thường tạo nên sự khác biệt rất lớn trong kết quả cuối cùng. Hãy nhớ lại việc áp dụng perceptron đa tầng để dự đoán giá nhà. Việc đầu tiên khi làm việc với dữ liệu thực tế là chuẩn hóa các đặc trưng đầu vào để chúng có giá trị trung bình bằng không và phương sai bằng một. Thông thường, việc chuẩn hóa này hoạt động tốt với các bộ tối ưu vì giá trị các tham số tiên nghiệm có cùng một khoảng tỷ lệ.

Khi huấn luyện các mạng thường gặp như Perceptron đa tầng hay CNN, các giá trị kích hoạt ở các tầng trung gian có thể nhận các giá trị với mức độ biến thiên lớn- dọc theo các tầng từ đầu vào đến đầu ra, qua các nút ở cùng một tầng, và theo thời gian do việc cập nhật giá trị tham số. Những nhà phát minh kỹ thuật chuẩn hoá theo batch cho rằng sự thay đổi trong phân phối của những giá trị kích hoạt có thể cản trở sự hội tụ của mạng. Dễ thấy rằng nếu một tầng có các giá trị kích hoạt lớn gấp 100 lần so với các tầng khác, thì cần phải có các điều chỉnh bổ trợ trong tốc độ học.

Mạng nhiều tầng có độ phức tạp cao và dễ gặp vấn đề quá khớp. Điều này cũng đồng nghĩa rằng kỹ thuật điều chuẩn càng trở nên quan trọng.

Chuẩn hoá theo batch được áp dụng cho từng tầng riêng lẻ (hoặc có thể cho tất cả các tầng) và hoạt động như sau: Trong mỗi vòng lặp huấn luyện, tại mỗi tầng, đầu tiên tính giá trị kích hoạt như thường lệ. Sau đó chuẩn hóa những giá trị kích hoạt của mỗi nút bằng việc trừ đi giá trị trung bình và chia cho độ lệch chuẩn. Cả hai đại lượng này được ước tính dựa trên số liệu thống kê của minibatch hiện tại. Chính vì chuẩn hóa dựa trên các số liệu thống kê của batch nên kỹ thuật này có tên gọi chuẩn hoá theo batch.

Lưu ý rằng, khi áp dụng BN với những minibatch có kích thước 1, mô hình sẽ không học được gì. Vì sau khi trừ đi giá trị trung bình, mỗi nút ẩn sẽ nhận giá trị 0! Dễ dàng suy luận ra là BN chỉ hoạt động hiệu quả và ổn định với kích thước minibatch đủ lớn. Cần ghi nhớ rằng, khi áp dụng BN là lựa chọn kích thước minibatch quan trọng hơn so với trường hợp không áp dụng BN.

BN chuyển đổi những giá trị kích hoạt tại tầng x nhất định theo công thức sau:

$$\mathrm{BN}(\mathbf{x}) = \boldsymbol{\gamma} \odot \frac{\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}}}{\hat{\boldsymbol{\sigma}}_{\mathcal{B}}} + \boldsymbol{\beta}.$$

Ở đây, μ^ là giá trị trung bình và σ^ là độ lệch chuẩn của các mẫu trong minibatch. Sau khi áp dụng BN, những giá trị kích hoạt của minibatch có giá trị trung bình bằng không và phương sai đơn vị. Vì việc lựa chọn phương sai đơn vị (so với một giá trị đặc biệt khác) là tuỳ ý, nên chúng ta thường thêm vào từng cặp tham số tương ứng là hệ số tỷ lệ γ và độ chệch β. Do đó, độ lớn giá trị kích hoạt ở những tầng trung gian không thể phân kỳ trong quá trình huấn luyện vì BN chủ động chuẩn hoá chúng theo giá trị trung bình và phương sai cho trước (thông qua μ và σ). Qua trực giác và thực nghiệm, dùng BN có thể cho phép chọn tốc độ học nhanh hơn.

Ký hiệu một minibatch là B, chúng ta tính $\hat{\mu}_B$ và $\hat{\sigma}_B$ theo công thức sau:

$$\hat{\boldsymbol{\mu}}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|}\sum_{\mathbf{x}\in\mathcal{B}} \mathbf{x},$$

$$\hat{\boldsymbol{\sigma}}_{\mathcal{B}}^2 = \frac{1}{|\mathcal{B}|}\sum_{\mathbf{x}\in\mathcal{B}} (\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}})^2 + \epsilon.$$

Lưu ý rằng chúng ta thêm hằng số rất nhỏ $\epsilon>0$ vào biểu thức tính phương sai để đảm bảo tránh phép chia cho 0 khi chuẩn hoá, ngay cả khi giá trị ước lượng phương sai thực nghiệm bằng không. Các ước lượng $\hat{\mu}_B$ và $\hat{\sigma}_B$ giúp đương đầu với vấn đề khi cần mở rộng số tầng của mạng (mạng học sâu hơn) bằng việc sử dụng nhiễu khi tính giá trị trung bình và phương sai. Bạn có thể nghĩ rằng nhiễu sẽ là vấn đề đáng ngại. Nhưng thực ra, nhiễu lại đem đến lợi ích.

Và đây là chủ đề thường xuất hiện trong học sâu. Vì những lý do vẫn chưa được giải thích rõ bằng lý thuyết, nhiều nguồn nhiễu khác nhau trong việc tối ưu hoá thường dẫn đến huấn luyện nhanh hơn và giảm quá khớp. Trong khi những nhà lý thuyết học máy truyền thống có thể bị vướng mắc ở việc định rõ điểm này, những thay đổi do nhiễu dường như hoạt động giống một dạng điều chuẩn. Trong một số nghiên cứu sơ bộ, [Teye et al., 2018] và [Luo et al., 2018] đã lần lượt chỉ ra các thuộc tính của BN liên quan tới tiên nghiệm Bayesian (Bayesian prior) và các lượng phạt (penalty). Cụ thể, nghiên cứu này làm sáng tỏ lý do BN hoạt động tốt nhất với các minibatch có kích cỡ vừa phải, trong khoảng 50 - 100.

Cố định một mô hình đã được huấn luyện, bạn có thể nghĩ rằng chúng ta nên sử dụng toàn bộ tập dữ liệu để ước tính giá trị trung bình và phương sai. Và đúng là như vậy. Bởi lẽ khi huấn luyện xong, tại sao ta lại muốn cùng một hình ảnh lại được phân loại khác nhau phụ thuộc vào batch chứa hình ảnh này? Trong quá trình huấn luyện, những tính toán chính xác như vậy không khả thi vì giá trị kích hoạt cho tất cả các điểm dữ liệu thay đổi mỗi khi cập nhật mô hình. Tuy nhiên, một khi mô hình đã được huấn luyện xong, chúng ta có thể tính được giá trị trung bình và phương sai của mỗi tầng dựa trên toàn bộ tập dữ liệu. Thực ra đây là tiêu chuẩn thực hành cho các mô hình sử dụng chuẩn hóa theo batch và do đó các tầng BN của MXNet hoạt động khác nhau giữa chế độ huấn luyện (chuẩn hoá bằng số liệu thống kê của minibatch) và chế độ dự đoán (chuẩn hoá bằng số liệu thống kê của toàn bộ tập dữ liệu)

Bây giờ chúng ta đã sẵn sàng để xem chuẩn hoá theo batch hoạt động thế nào trong thực tế.

6.7.5.2. Tầng chuẩn hoá theo batch

Thực hiện việc chuẩn hóa theo batch cho tầng kết nối đầy đủ và tầng tích chập hơi khác nhau một chút. Chúng ta sẽ thảo luận cả hai trường hợp trên. Nhớ rằng một khác biệt lớn của BN so với những tầng khác là vì BN cần số liệu thống kê trên toàn minibatch, chúng ta không thể bỏ qua kích thước batch như đã làm với các tầng khác.

6.7.5.2.1. Tầng kết nối đầy đủ

Khi áp dụng BN cho tầng kết nối đầy đủ, ta thường chèn BN sau bước biến đổi affine và trước hàm kích hoạt phi tuyến. Kí hiệu đầu vào của tầng là x, hàm biến đổi tuyến tính là $f_\theta(\cdot)$ (với trọng số là θ), hàm kích hoạt là $\phi(\cdot)$, và phép tính BN là $\text{BN}_{\beta,\gamma}$ với tham số β và γ, chúng ta sẽ biểu diễn việc tính toán tầng kết nối đầy đủ h khi chèn lớp BN vào như sau:

$$\mathbf{h} = \phi(\text{BN}(\mathbf{W}\mathbf{x} + \mathbf{b})).$$

Nhắc lại rằng giá trị trung bình và phương sai sẽ được tính toán trên chính minibatch B mà sẽ được biến đổi. Cũng cần lưu ý rằng hệ số tỷ lệ γ và độ chệch β là những tham số cần được học cùng với bộ tham số quen thuộc θ.

6.7.5.2.2. Tầng tích chập

Tương tự với tầng tích chập, chúng ta áp dụng BN sau phép tích chập và trước hàm kích hoạt phi tuyến. Khi áp dụng phép tích chập cho đầu ra nhiều kênh, chúng ta cần thực hiện chuẩn hóa theo batch cho mỗi đầu ra của những kênh này, và mỗi kênh sẽ có riêng cho nó các tham số tỉ lệ và độ chệch, cả hai đều là các số vô hướng. Giả sử các minibatch có kích thước m, đầu ra cho mỗi kênh của phép tích chập có chiều cao p và chiều rộng q. Với tầng tích chập, ta sẽ thực hiện mỗi phép chuẩn hoá theo batch trên m·p·q phần tử trên từng kênh đầu ra cùng lúc. Vì thế trên từng kênh, ta sử dụng giá trị trên tất cả các vị trí không gian để tính trung bình μ^ và phương sai σ^ và sau đó dùng hai giá trị này để chuẩn hóa các giá trị tại mỗi vị trí không gian trên kênh đó.

6.7.5.2.3. Chuẩn hoá theo Batch trong Quá trình Dự đoán

Như đã đề cập trước đó, BN thường hoạt động khác nhau trong chế độ huấn luyện và chế độ dự đoán. Thứ nhất, nhiễu trong μ và σ phát sinh từ việc chúng được xấp xỉ trên các minibatch không còn là nhiễu được mong muốn một khi ta đã huấn luyện xong mô hình. Thứ hai, trong nhiều trường hợp sẽ là xa xỉ khi tính toán các con số thống kê sau mỗi lần chuẩn hoá theo batch, ví dụ, khi cần áp dụng mô hình để đưa ra một kết quả dự đoán mỗi lần.

Thông thường, sau khi huấn luyện, chúng ta sử dụng toàn bộ tập dữ liệu để tính toán các con số thống kê của các giá trị kích hoạt và sau đó cố định chúng tại thời điểm dự đoán. Do đó, BN hoạt động khác nhau trong quá trình huấn luyện và kiểm tra. Lưu ý rằng dropout cũng có tính chất này.

6.7.5.6. Tranh luận

Theo trực giác, chuẩn hóa theo batch được cho là làm cảnh quan tối ưu (optimization landscape) mượt mà hơn. Tuy nhiên, cần cẩn thận phân biệt giữa suy đoán theo trực giác và lời giải thích thực sự cho các hiện tượng quan sát thấy khi huấn luyện các mô hình học sâu. Hãy nhớ lại rằng ngay từ đầu ta thậm chí không rõ tại sao các mạng nơ-ron sâu đơn giản hơn (như Perceptron đa tầng và CNN truyền thống) lại có thể khái quát tốt như vậy. Ngay cả với dropout

và điều chuẩn L2, chúng vẫn linh hoạt đến mức khả năng khái quát hóa trên dữ liệu chưa nhìn thấy của chúng không thể giải thích được bằng các điều kiện bảo đảm sự khái quát hóa trong lý thuyết học truyền thống.

Trong bài báo gốc khi đề xuất phương pháp chuẩn hóa theo batch, các tác giả ngoài việc giới thiệu một công cụ mạnh mẽ và hữu ích đã đưa ra lời giải thích lý do BN hoạt động tốt: bằng cách giảm sự dịch chuyển hiệp biến nội bộ - internal covariate shift. Có thể hiểu ý các tác giả về sự dịch chuyển hiệp biến nội bộ giống với cách giải thích ở trên-rằng phân phối của giá trị kích hoạt thay đổi trong quá trình huấn luyện. Tuy nhiên, có hai vấn đề với cách giải thích này: (1) Sự dịch chuyển phân phối này rất khác so với sự dịch chuyển hiệp biến, việc đặt tên như vậy có sự nhầm lẫn. (2) Cách giải thích này vẫn chưa đủ cụ thể và chặt chẽ, vẫn để ngỏ câu hỏi: chính xác thì tại sao kỹ thuật này hoạt động? Xuyên suốt cuốn sách này, chúng tôi hướng đến việc truyền đạt những kinh nghiệm thực tế để xây dựng các mạng nơ-ron sâu. Tuy nhiên, chúng tôi tin rằng cần phân biệt rõ những kinh nghiệm dựa trên trực giác này với những bằng chứng khoa học rõ ràng. Cuối cùng, khi đã thành thạo tài liệu này và bắt đầu viết các nghiên cứu của riêng mình, bạn cần phân biệt rõ ràng giữa khẳng định và linh cảm.

Nối tiếp thành công của BN, cách giải thích của kỹ thuật này thông qua khái niệm sự dịch chuyển hiệp biến nội bộ liên tục xuất hiện trong các tranh luận, các tài liệu kỹ thuật và trên các diễn đàn về cách trình bày nghiên cứu học máy. Trong một bài phát biểu đáng nhớ được đưa ra khi nhận giải thưởng Test of Time Award tại hội nghị NeurIPS 2017, Ali Rahimi đã sử dụng sự dịch chuyển hiệp biến nội bộ như một tiêu điểm trong một cuộc tranh luận so sánh thực hành học sâu hiện đại với thuật giả kim. Sau đó, cách giải thích này đã được xem xét lại một cách chi tiết trong một bài báo về các xu hướng đáng lo ngại trong học máy [Lipton & Steinhardt, 2018]. Trong các tài liệu kỹ thuật, các tác giả khác ([Santurkar et al., 2018]) đã đề xuất các giải thích thay thế cho sự thành công của BN, dù phần nào đó trái ngược với cách giải thích trong bài báo gốc.

Chúng tôi lưu ý rằng sự dịch chuyển hiệp biến nội bộ không đáng bị chỉ trích, có hàng ngàn lập luận mơ hồ được đưa ra mỗi năm trong nhiều tài liệu kỹ thuật về học máy. Việc nó trở thành tâm điểm của những cuộc tranh luận rất có thể là do sự phổ biến của nó trong cộng đồng học máy. Chuẩn hóa theo batch là một phương pháp quan trọng, được áp dụng trong gần như tất cả các bộ phân loại hình ảnh đã được triển khai, mang lại hàng chục ngàn trích dẫn cho bài báo giới thiệu kĩ thuật này.

6.7.5.7. Kết luận

- Trong quá trình huấn luyện mô hình, chuẩn hóa theo batch liên tục điều chỉnh đầu ra trung gian của mạng nơ-ron theo giá trị trung bình và độ lệch chuẩn của minibatch, giúp các giá trị này ổn định hơn.

- Chuẩn hóa theo batch có chút khác biệt khi áp dụng cho tầng kết nối đầy đủ và tầng tích chập.

- Giống như tầng dropout, tầng chuẩn hóa theo batch sẽ tính ra kết quả khác nhau trong chế độ huấn luyện và chế độ dự đoán.

- Chuẩn hóa theo batch có nhiều tác dụng phụ có lợi, chủ yếu là về điều chuẩn. Tuy nhiên, cách giải thích ban đầu về việc giảm sự dịch chuyển hiệp biến dường như không hợp lý.

CHƯƠNG 7

MẠNG NƠ-RON HỒI TIẾP

Cho đến nay, chúng ta đã gặp hai loại dữ liệu: các vector tổng quát và hình ảnh. Với dữ liệu hình ảnh, ta đã thiết kế các tầng chuyên biệt nhằm tận dụng tính chính quy (regularity property) của hình ảnh. Nói cách khác, nếu ta hoán vị các điểm ảnh trong một ảnh, ta sẽ thu được một bức ảnh trông giống như các khuôn mẫu kiểm tra (test pattern) hay thấy trong truyền hình analog, và rất khó để suy luận về nội dung của chúng.

Quan trọng hơn là cho đến thời điểm này, chúng ta đã ngầm định rằng dữ liệu được sinh ra từ những phân phối độc lập và giống hệt nhau (independently and identically distributed - i.i.d.). Thật không may, điều này lại không đúng với hầu hết các loại dữ liệu. Ví dụ, các từ trong đoạn văn này được viết theo một trình tự nhất định mà nếu bị hoán vị đi một cách ngẫu nhiên thì sẽ rất khó để giải mã ý nghĩa của chúng. Tương tự với các khung hình trong video, tín hiệu âm thanh trong một cuộc hội thoại hoặc hành vi duyệt web, tất cả đều có cấu trúc tuần tự. Do đó, hoàn toàn hợp lý khi ta giả định rằng các mô hình chuyên biệt cho những kiểu dữ liệu này sẽ giúp việc mô tả dữ liệu và giải quyết các bài toán ước lượng được tốt hơn.

Một vấn đề nữa phát sinh khi chúng ta không chỉ nhận một chuỗi làm đầu vào mà còn muốn dự đoán những phần tử tiếp theo của chuỗi. Ví dụ, bài toán có thể là dự đoán phần tử tiếp theo trong dãy 2, 4, 6, 8, 10, … Tác vụ này khá phổ biến trong phân tích chuỗi thời gian: để dự đoán thị trường chứng khoán, đường cong biểu hiện tình trạng sốt của bệnh nhân, hoặc gia tốc cần thiết cho một chiếc xe đua. Một lần nữa, chúng ta muốn xây dựng các mô hình có thể xử lý ổn thỏa kiểu dữ liệu trên.

Tóm lại, trong khi các mạng nơ-ron tích chập có thể xử lý hiệu quả thông tin trên chiều không gian, thì các mạng nơ-ron hồi tiếp được thiết kế để xử lý thông tin tuần tự tốt hơn. Các mạng này sử dụng các biến trạng thái để lưu trữ thông tin trong quá khứ, sau đó dựa vào chúng và các đầu vào hiện tại để xác định các đầu ra hiện tại.

Ở chương này, đa phần những ví dụ đề cập đến các mạng hồi tiếp đều dựa trên dữ liệu văn bản. Vì vậy, chúng ta sẽ cùng đào sâu tìm hiểu những mô hình ngôn ngữ. Sau khi tìm hiểu về dữ liệu chuỗi, ta sẽ thảo luận các khái niệm cơ bản của mô hình ngôn ngữ để làm bàn đạp cho việc thiết kế các mạng nơ-ron hồi tiếp. Cuối cùng, ta sẽ tiến hành mô tả phương pháp tính toán gradient trong các mạng nơ-ron hồi tiếp để từ đó hiểu rõ hơn các vấn đề có thể gặp phải trong quá trình huấn luyện.

7.1. Mô hình chuỗi

Hãy tưởng tượng rằng bạn đang xem phim trên Netflix. Là một người dùng Netflix tốt, bạn quyết định đánh giá từng bộ phim một cách cẩn thận. Xét cho cùng, bạn muốn xem thêm nhiều bộ phim hay phải không? Nhưng hóa ra, mọi thứ không hề đơn giản như vậy. Đánh giá của mỗi người về một bộ phim có thể thay đổi đáng kể theo thời gian. Trên thực tế, các nhà tâm lý học thậm chí còn đặt tên cho một số hiệu ứng:

- Hiệu ứng mỏ neo: dựa trên ý kiến của người khác. Ví dụ, xếp hạng của một bộ phim sẽ tăng lên sau khi nó thắng giải Oscar, mặc dù đoàn làm phim này không có bất kỳ tác động nào về mặt quảng bá đến bộ phim. Hiệu ứng này kéo dài trong vòng một vài tháng cho đến khi giải thưởng bị lãng quên. [Wu et al., 2017] chỉ ra rằng hiệu ứng này tăng chỉ số xếp hạng thêm hơn nửa điểm.

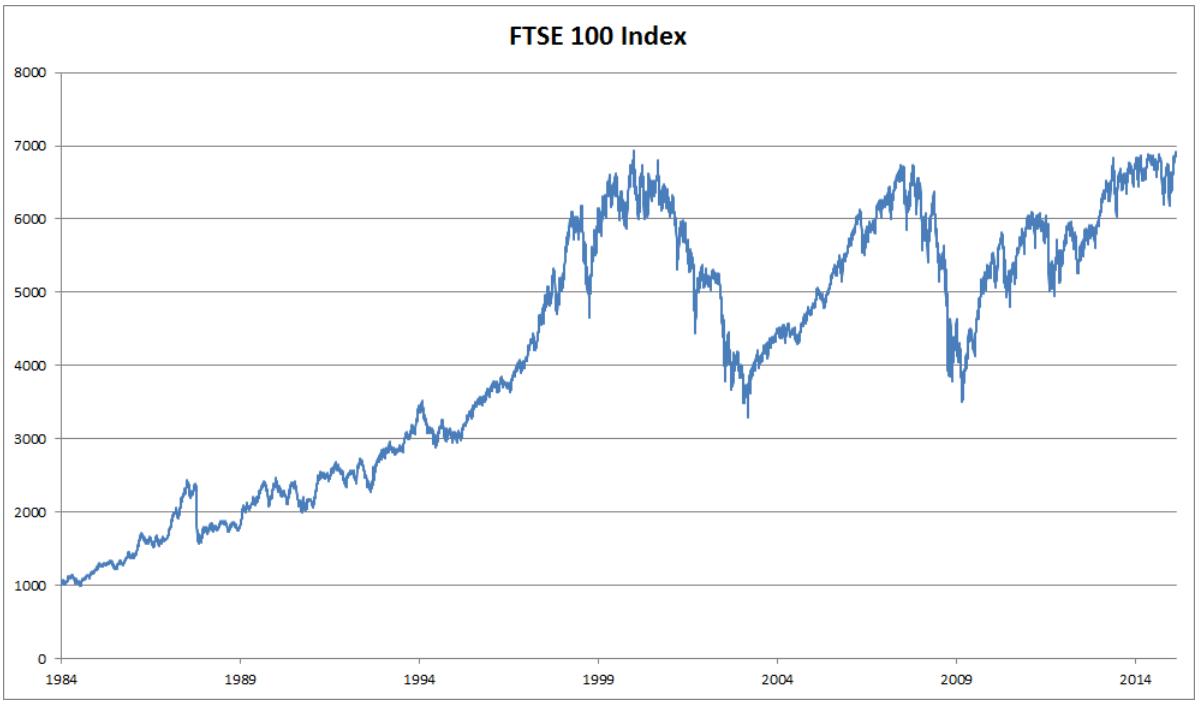
- Hiệu ứng vòng xoáy khoái lạc: con người nhanh chóng thích nghi để chấp nhận một tình huống tốt hơn (hoặc xấu đi) như một điều bình thường mới. Chẳng hạn, sau khi xem nhiều bộ phim hay, sự kỳ vọng rằng bộ phim tiếp theo sẽ hay tương đương hoặc thậm chí phải hay hơn trở nên khá cao, do đó ngay cả một bộ phim trung bình cũng có thể bị coi là một bộ phim tồi.
- Tính thời vụ: rất ít khán giả thích xem một bộ phim về ông già Noel vào tháng 8.
- Trong một số trường hợp, các bộ phim trở nên không được ưa chuộng do những hành động sai trái của các đạo diễn hoặc diễn viên tham gia vào quá trình sản xuất phim.
- Một số phim trở thành "phim cult" vì chúng gần như tệ đến mức phát cười. Plan 9 from Outer Space và Troll 2 là hai ví dụ nổi tiếng.

Tóm lại, thứ bậc xếp hạng không hề cố định. Sử dụng các động lực dựa trên thời gian đã giúp [Koren, 2009] đề xuất phim chính xác hơn. Tuy nhiên, vấn đề không chỉ là về phim ảnh.

- Nhiều người dùng có thói quen rất đặc biệt liên quan tới thời gian mở ứng dụng. Chẳng hạn, học sinh sử dụng các ứng dụng mạng xã hội nhiều hơn hẳn sau giờ học. Các ứng dụng giao dịch chứng khoán được sử dụng nhiều khi thị trường mở cửa.
- Việc dự đoán giá cổ phiếu ngày mai khó hơn nhiều so với việc dự đoán giá cổ phiếu bị bỏ lỡ ngày hôm qua, mặc dù cả hai đều là bài toán ước tính một con số. Rốt cuộc, nhìn lại quá khứ dễ hơn nhiều so với dự đoán tương lai. Trong thống kê, bài toán đầu tiên được gọi là ngoại suy và bài toán sau được gọi là nội suy.
- Âm nhạc, giọng nói, văn bản, phim ảnh, bước đi, v.v … đều có tính chất tuần tự. Nếu chúng ta hoán vị chúng, chúng sẽ không còn nhiều ý nghĩa. Dòng tiêu đề chó cắn người ít gây ngạc nhiên hơn nhiều so với người cắn chó, mặc dù các từ giống hệt nhau.
- Các trận động đất có mối tương quan mạnh mẽ, tức sau một trận động đất lớn, rất có thể sẽ có một số dư chấn nhỏ hơn và xác suất xảy ra dư chấn cao hơn nhiều so với trường hợp trận động đất lớn không xảy ra trước đó. Trên thực tế, các trận động đất có mối tương quan về mặt không-thời gian, tức các dư chấn thường xảy ra trong một khoảng thời gian ngắn và ở gần nhau.
- Con người tương tác với nhau một cách tuần tự, điều này có thể được thấy trong các cuộc tranh cãi trên Twitter, các điệu nhảy và các cuộc tranh luận.

7.1.1. Các công cụ thống kê

Tóm lại, ta cần các công cụ thống kê và các kiến trúc mạng nơ-ron sâu mới để xử lý dữ liệu chuỗi. Để đơn giản hóa mọi việc, ta sẽ sử dụng giá cổ phiếu để làm ví dụ.



Hình 7.1. Giá cổ phiếu FTSE 100 trong vòng 30 năm

Ta sẽ gọi giá cổ phiếu là $x_t \geq 0$, tức tại thời điểm $t \in N$ ta thấy giá cổ phiếu bằng x_t. Để có thể kiếm lời trên thị trường chứng khoán vào ngày t, một nhà giao dịch sẽ muốn dự đoán x_t thông qua

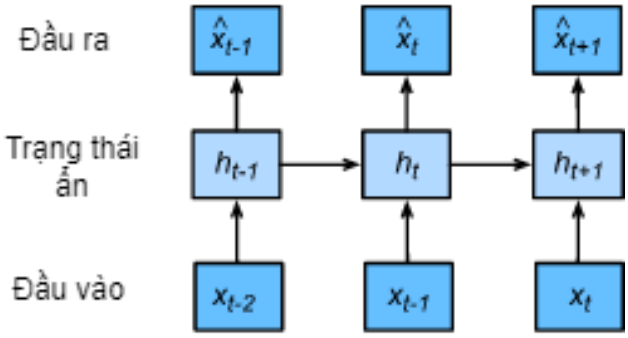
$$x_t \sim P(x_t \mid x_{t-1}, \ldots, x_1).$$

78.1.1.1. Mô hình Tự hồi quy

Để dự đoán giá cổ phiếu, các nhà giao dịch có thể sử dụng một mô hình hồi quy. Chỉ có một vấn đề lớn ở đây, đó là số lượng đầu vào, $x_{t-1},\ldots,x_1$ thay đổi tùy thuộc vào t. Cụ thể, số lượng đầu vào sẽ tăng cùng với lượng dữ liệu thu được và ta sẽ cần một phép tính xấp xỉ để làm cho giải pháp này khả thi về mặt tính toán. Phần lớn nội dung tiếp theo trong chương này sẽ xoay quanh việc làm thế nào để ước lượng $p(x_t|x_{t-1},\ldots,x_1)$ một cách hiệu quả. Nói ngắn gọn, ta có hai chiến lược:

Giả sử rằng việc sử dụng một chuỗi có thể rất dài $x_{t-1},\ldots,x_1$ là không thực sự cần thiết. Trong trường hợp này, ta có thể hài lòng với một khoảng thời gian τ và chỉ sử dụng các quan sát $x_{t-1},\ldots,x_{t-\tau}$. Lợi ích trước mắt là bây giờ số lượng đối số luôn bằng nhau, ít nhất là với $t>\tau$. Điều này sẽ cho phép ta huấn luyện một mạng sâu như được đề cập ở bên trên. Các mô hình như vậy được gọi là các mô hình tự hồi quy (autoregressive), vì chúng tự thực hiện hồi quy trên chính mình.

Một chiến lược khác, được minh họa trong Hình 7.2, là giữ một giá trị h_t để tóm tắt các quan sát trong quá khứ, đồng thời cập nhật h_t bên cạnh việc dự đoán $\hat{x}_t$. Kết quả là mô hình sẽ ước tính x_t với $\hat{x}_t = p(x_t|x_{t-1},h_t)$ và cập nhật $h_t = g(h_{t-1},x_{t-1})$. Do ht không bao giờ được quan sát nên các mô hình này còn được gọi là các mô hình tự hồi quy tiềm ẩn (latent autoregressive model). LSTM và GRU là hai ví dụ cho kiểu mô hình này.



Hình 7.2. Một mô hình tự hồi quy tiềm ẩn.

Cả hai trường hợp đều đặt ra câu hỏi về cách tạo ra dữ liệu huấn luyện. Người ta thường sử dụng các quan sát từ quá khứ cho đến hiện tại để dự đoán các quan sát xảy ra trong tương lai. Rõ ràng chúng ta không thể trông đợi thời gian sẽ đứng yên. Tuy nhiên, một giả định phổ biến là: tuy các giá trị cụ thể của x_t có thể thay đổi, ít ra động lực của chuỗi thời gian sẽ không đổi. Điều này khá hợp lý, vì nếu động lực thay đổi thì ta sẽ không thể dự đoán được nó bằng cách sử dụng dữ liệu mà ta đang có. Các nhà thống kê gọi các động lực không thay đổi này là cố định (stationary). Dù có làm gì đi chăng nữa, chúng ta vẫn sẽ tìm được ước lượng của toàn bộ chuỗi thời gian thông qua

$$P(x_1, \dots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{t-1}, \dots, x_1).$$

Lưu ý rằng các xem xét trên vẫn đúng trong trường hợp chúng ta làm việc với các đối tượng rời rạc, chẳng hạn như từ ngữ thay vì số. Sự khác biệt duy nhất trong trường hợp này là chúng ta cần sử dụng một bộ phân loại thay vì một bộ hồi quy để ước lượng $P(x_t \mid x_{t-1}, \dots, x_1)$.

7.1.1.2. Mô hình Markov

Nhắc lại phép xấp xỉ trong một mô hình tự hồi quy, chúng ta chỉ sử dụng $x_{t-1}, \dots, x_{t-\tau}$ thay vì $x_{t-1}, \dots, x_1$ để ước lượng x_t. Bất cứ khi nào phép xấp xỉ này là chính xác, chúng ta nói rằng chuỗi thỏa mãn điều kiện Markov. Cụ thể, nếu τ=1, chúng ta có mô hình Markov bậc một và $P(x)$ như sau

$$P(x_1, \dots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{t-1}) \text{ trong đó } P(x_1 \mid x_0) = P(x_1).$$

Các mô hình như trên rất hữu dụng bất cứ khi nào x_t chỉ là các giá trị rời rạc, vì trong trường hợp này, quy hoạch động có thể được sử dụng để tính toán chính xác các giá trị theo chuỗi. Ví dụ, chúng ta có thể tính toán $p(x_{t+1} \mid x_{t-1})$ một cách hiệu quả bằng cách chỉ sử dụng các quan sát trong một khoảng thời gian ngắn tại quá khứ:

$$\begin{aligned} p(x_{t+1} \mid x_{t-1}) &= \frac{\sum_{x_t} p(x_{t+1}, x_t, x_{t-1})}{P(x_{t-1})} \\ &= \frac{\sum_{x_t} p(x_{t+1} \mid x_t, x_{t-1}) P(x_t, x_{t-1})}{P(x_{t-1})} \\ &= \sum_{x_t} p(x_{t+1} \mid x_t) P(x_t \mid x_{t-1}) \end{aligned}$$

Các công cụ trên được sử dụng rất phổ biến trong các thuật toán điều khiển và học tăng cường.

7.1.1.3. Quan hệ Nhân quả

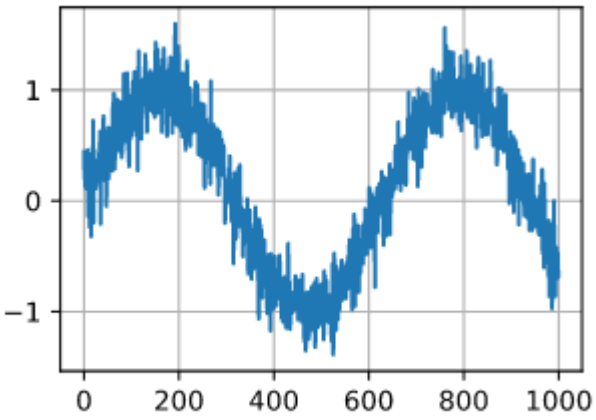
Về nguyên tắc, không có gì sai khi trải (unfolding) $p(x_1, \dots, x_T)$ theo thứ tự ngược lại. Bằng cách đặt điều kiện như vậy, chúng ta luôn có thể viết chúng như sau

$$p(x_1, \dots, x_T) = \prod_{t=T}^{1} p(x_t \mid x_{t+1}, \dots, x_T).$$

Trên thực tế, nếu có một mô hình Markov, chúng ta cũng có thể thu được một phân phối xác suất có điều kiện ngược. Tuy nhiên trong nhiều trường hợp vẫn tồn tại một trật tự tự nhiên cho dữ liệu, cụ thể đó là chiều thuận theo thời gian. Rõ ràng là các sự kiện trong tương lai không thể ảnh hưởng đến quá khứ. Do đó, nếu thay đổi x_t thì ta có thể ảnh hưởng đến những gì xảy ra tại x_{t+1} trong tương lai, nhưng lại không thể ảnh hưởng tới quá khứ theo chiều ngược lại. Nếu chúng ta thay đổi x_t, phân phối trên các sự kiện trong quá khứ sẽ không thay đổi. Do đó, việc giải thích ($x_{t+1} \mid x_t$) sẽ đơn giản hơn là $Px_t \mid x_{t+1}$). Ví dụ: [Hoyer et al., 2009] chỉ ra rằng trong một số trường hợp chúng ta có thể tìm $x_{t+1} = f(x_t) + \epsilon$ khi có thêm nhiễu, trong khi điều ngược lại thì không đúng. Đây là một tin tuyệt vời vì chúng ta thường quan tâm tới việc ước lượng theo chiều thuận hơn. Để tìm hiểu thêm về chủ đề này, có thể tìm đọc cuốn sách [Peters et al., 2017a]. Chúng ta sẽ chỉ tìm hiểu sơ qua trong phần này.

7.1.2. Một ví dụ đơn giản

Sau khi đề cập nhiều về lý thuyết, bây giờ chúng ta hãy thử lập trình minh họa. Đầu tiên, hãy khởi tạo một vài dữ liệu như sau. Để đơn giản, chúng ta tạo chuỗi thời gian bằng cách sử dụng hàm sin cộng thêm một chút nhiễu.



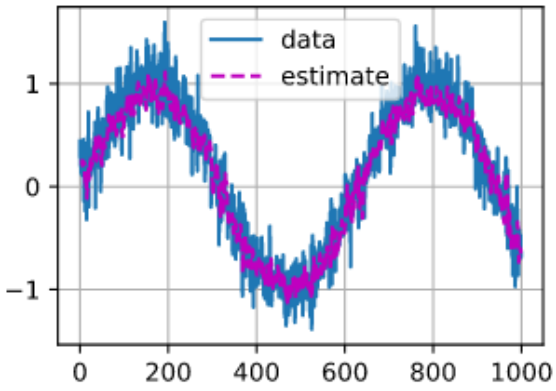
Hình 7.3. Dữ liệu được tạo bằng cách sử dụng hàm sin theo chuỗi thời gian

Tiếp theo, chúng ta cần biến chuỗi thời gian này thành các đặc trưng và nhãn có thể được sử dụng để huấn luyện mạng. Dựa trên kích thước embedding τ, chúng ta ánh xạ dữ liệu thành các cặp $y_t=x_t$ và $z_t=(x_{t-1},\dots,x_{t-\tau})$. Để ý kĩ, có thể thấy rằng ta sẽ mất τ điểm dữ liệu đầu tiên, vì chúng ta không có đủ τ điểm dữ liệu trong quá khứ để làm đặc trưng cho chúng. Một cách đơn giản để khắc phục điều này, đặc biệt là khi chuỗi thời gian rất dài, là loại bỏ đi số ít các phần tử đó. Một cách khác là đệm giá trị 0 vào chuỗi thời gian. Mã nguồn dưới đây về cơ bản là giống hệt với mã nguồn huấn luyện trong các phần trước. Chúng tôi cố gắng giữ cho kiến trúc đơn giản với vài tầng kết nối đầy đủ, hàm kích hoạt ReLU và hàm mất mát ℓ_2. Do việc mô hình hóa phần lớn là giống

với khi ta xây dựng các bộ ước lượng hồi quy viết bằng Gluon trong các phần trước, nên chúng ta sẽ không đi sâu vào chi tiết trong phần này.

7.1.3. Dự đoán của Mô hình

Vì cả hai giá trị mất mát trên tập huấn luyện và kiểm tra đều nhỏ, chúng ta kỳ vọng mô hình trên sẽ hoạt động tốt. Hãy cùng xác nhận điều này trên thực tế. Điều đầu tiên cần kiểm tra là mô hình có thể dự đoán những gì sẽ xảy ra trong bước thời gian kế tiếp tốt như thế nào.

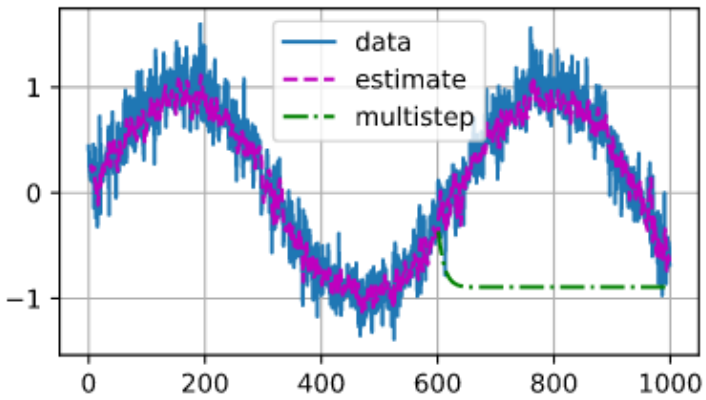


Hình 7.4. Kết quả ước lượng và số liệu bạn đầu

Kết quả khá tốt, đúng như những gì chúng ta mong đợi. Thậm chí sau hơn 600 mẫu quan sát, phép ước lượng vẫn trông khá tin cậy. Chỉ có một chút vấn đề: nếu chúng ta quan sát dữ liệu tới bước thời gian thứ 600, chúng ta không thể hy vọng sẽ nhận được nhãn gốc cho tất cả các dự đoán tương lai. Thay vào đó, chúng ta cần tiến lên từng bước một:

$$
\begin{aligned}
\hat{x}_{605} &= f(x_{601}, x_{602}, x_{603}, x_{604}),\\
\hat{x}_{606} &= f(x_{602}, x_{603}, x_{604}, \hat{x}_{605}),\\
\hat{x}_{607} &= f(x_{603}, x_{604}, \hat{x}_{605}, \hat{x}_{606}),\\
\hat{x}_{608} &= f(x_{604}, \hat{x}_{605}, \hat{x}_{606}, \hat{x}_{607}),\\
\hat{x}_{609} &= f(\hat{x}_{605}, \hat{x}_{606}, \hat{x}_{607}, \hat{x}_{608}),\\
&\ldots
\end{aligned}
$$

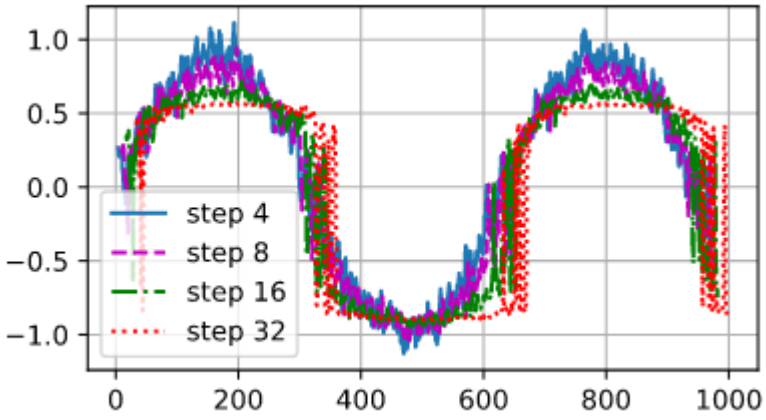
Nói cách khác, chúng ta sẽ phải sử dụng những dự đoán của mình để đưa ra dự đoán trong tương lai. Hãy cùng xem cách này có ổn không.



Hình 7.5. Kết quả dự đoán tương lai không chính xác

Ví dụ trên cho thấy, cách này đã thất bại thảm hại. Các giá trị ước lượng rất nhanh chóng suy giảm thành một hằng số chỉ sau một vài bước. Tại sao thuật toán trên hoạt động tệ đến thế? Suy cho cùng, lý do là trên thực tế các sai số dự đoán bị chồng chất qua các bước thời gian. Cụ thể, sau bước thời gian 1 chúng ta có nhận được sai số $\epsilon 1 = \bar{\epsilon}$. Tiếp theo, đầu vào cho bước thời gian 2 bị nhiễu loạn bởi ϵ_1, do đó chúng ta nhận được sai số dự đoán $\epsilon_2 = \bar{\epsilon} + L\epsilon_1$. Tương tự như thế cho các bước thời gian tiếp theo. Sai số có thể phân kỳ khá nhanh khỏi các quan sát đúng. Đây là một hiện tượng phổ biến. Ví dụ, dự báo thời tiết trong 24 giờ tới có độ chính xác khá cao nhưng nó giảm đi nhanh chóng với những dự báo xa hơn quãng thời gian đó.

Chúng ta hãy kiểm chứng quan sát trên bằng cách tính toán dự đoán k bước thời gian trên toàn bộ chuỗi.



Hình 7.6. Kết quả dự đoán k bước

Điều này minh họa rõ ràng chất lượng của các ước lượng thay đổi như thế nào khi chúng ta cố gắng dự đoán xa hơn trong tương lai. Mặc dù những dự đoán có độ dài là 8 bước vẫn còn khá tốt, bất cứ kết quả dự đoán nào vượt ra ngoài khoảng đó thì khá là vô dụng.

7.1.4. Kết luận

- Các mô hình chuỗi thường yêu cầu các công cụ thống kê chuyên biệt để ước lượng. Hai lựa chọn phổ biến đó là các mô hình tự hồi quy và mô hình tự hồi quy biến tiềm ẩn.

- Sai số bị tích lũy và chất lượng của phép ước lượng suy giảm đáng kể khi mô hình dự đoán các bước thời gian xa hơn.

- Khó khăn trong phép nội suy và ngoại suy khá khác biệt. Do đó, nếu bạn có một kiểu dữ liệu chuỗi thời gian, hãy luôn để ý trình tự thời gian của dữ liệu khi huấn luyện, hay nói cách khác, không bao giờ huấn luyện trên dữ liệu thuộc về bước thời gian trong tương lai.

- Đối với các mô hình nhân quả (ví dụ, ở đó thời gian đi về phía trước), ước lượng theo chiều xuôi thường dễ dàng hơn rất nhiều so với chiều ngược lại.

7.2. Tiền Xử lý Dữ liệu Văn bản

Dữ liệu văn bản là một ví dụ điển hình của dữ liệu chuỗi. Một bài báo có thể coi là một chuỗi các từ, hoặc một chuỗi các ký tự. Dữ liệu văn bản là một dạng dữ liệu quan trọng bên cạnh dữ liệu hình ảnh được sử dụng trong cuốn sách này, phần này sẽ được dành để giải thích các bước tiền xử lý thường gặp cho loại dữ liệu này. Quá trình tiền xử lý thường bao gồm bốn bước sau:

- Nạp dữ liệu văn bản ở dạng chuỗi ký tự vào bộ nhớ.
- Chia chuỗi thành các token trong đó một token có thể là một từ hoặc một ký tự.
- Xây dựng một bộ từ vựng cho các token để ánh xạ chúng thành các chỉ số (index).
- Ánh xạ tất cả các token trong dữ liệu văn bản thành các chỉ số để dễ dàng đưa vào các mô hình.

7.2.1. Đọc Bộ dữ liệu

Để bắt đầu chúng ta nạp dữ liệu văn bản từ cuốn sách Cỗ máy Thời gian (Time Machine) của tác giả H. G. Wells. Đây là một kho ngữ liệu khá nhỏ chỉ hơn 30.000 từ, nhưng nó đủ tốt cho mục đích minh họa. Nhiều bộ dữ liệu trên thực tế chứa hàng tỷ từ. Hàm sau đây đọc dữ liệu thành một danh sách các câu, mỗi câu là một chuỗi. Chúng ta bỏ qua dấu câu và chữ viết hoa.

7.2.2. Token hoá

Với mỗi câu, chúng ta chia nó thành một danh sách các token. Một token là một điểm dữ liệu mà mô hình sẽ huấn luyện và đưa ra dự đoán từ nó. Hàm dưới đây làm nhiệm vụ tách một câu thành các từ hoặc các ký tự, và trả về một danh sách các chuỗi đã được phân tách.

7.2.3. Bộ Từ vựng

Token kiểu chuỗi không phải là kiểu dữ liệu tiện lợi được sử dụng bởi các mô hình, thay vào đó chúng thường nhận dữ liệu đầu vào dưới dạng số. Bây giờ, chúng ta sẽ xây dựng một bộ từ điển, thường được gọi là bộ từ vựng (vocabulary), để ánh xạ chuỗi token thành các chỉ số bắt đầu từ 0. Để làm điều này, đầu tiên chúng ta lấy các token xuất hiện (không lặp lại) trong toàn bộ tài liệu, thường được gọi là kho ngữ liệu (corpus), và sau đó gán một giá trị số (chỉ số) cho mỗi token dựa trên tần suất xuất hiện của chúng. Các token có tần suất xuất hiện rất ít thường được loại bỏ để giảm độ phức tạp. Một token không xuất hiện trong kho ngữ liệu hay đã bị loại bỏ thường được ánh xạ vào một token vô danh đặc biệt (“<unk>”). Chúng ta có thể tùy chọn thêm vào các token dự trữ, ví dụ token “<pad>” được sử dụng để đệm từ, token “<bos>” để biểu thị vị trí bắt đầu của câu, và token “<eos>” để biểu thị vị trí kết thúc của câu.

Chúng ta xây dựng một bộ từ vựng với tập dữ liệu cỗ máy thời gian nói trên thành một kho ngữ liệu, và in ra phép ánh xạ giữa một vài token với các chỉ số của chúng.

Sau đó, chúng ta có thể chuyển đổi từng câu vào một danh sách các chỉ số. Để minh họa một cách chi tiết, chúng ta in hai câu với các chỉ số tương ứng của chúng.

7.2.4. Kết hợp Tất cả lại

Chúng ta đóng gói tất cả các hàm trên thành hàm load_corpus_time_machine, trả về corpus, một danh sách các chỉ số của token, và bộ từ vựng vocab của kho ngữ liệu cỗ máy thời gian. Chúng ta đã sửa đổi một vài thứ ở đây là: corpus là một danh sách đơn nhất, không phải một danh sách các danh sách token, vì chúng ta không lưu các thông tin chuỗi trong các mô hình bên dưới. Bên cạnh đó, chúng ta sẽ sử dụng các token ký tự để đơn giản hóa việc huấn luyện mô hình trong các phần sau.

7.2.5. Kết luận

Chúng ta đã tiền xử lý các tài liệu văn bản bằng cách token hóa chúng thành các từ hoặc ký tự, và sau đó ánh xạ chúng thành các chỉ số tương ứng.

7.3. Mô hình Ngôn ngữ và Tập dữ liệu

Mục 7.2 đã trình bày cách ánh xạ dữ liệu văn bản sang token, những token này có thể được xem như một chuỗi thời gian của các quan sát rời rạc. Giả sử văn bản độ dài T có dãy token là $x_1, x_2, \dots, x_T$, thì $1 \leq t \leq T$) có thể coi là đầu ra (hoặc nhãn) tại bước thời gian t. Khi đã có chuỗi thời gian trên, mục tiêu của mô hình ngôn ngữ là ước tính xác suất của

$$P(x_1, x_2, \dots, x_T).$$

Mô hình ngôn ngữ vô cùng hữu dụng. Chẳng hạn, một mô hình lý tưởng có thể tự tạo ra văn bản tự nhiên, chỉ bằng cách chọn một từ w_t tại thời điểm t với $w_t \sim P(w_t \mid w_{t-1}, \dots, w_1)$. Khác hoàn toàn với việc chỉ gõ phím ngẫu nhiên như trong định lý con khỉ vô hạn (infinite monkey theorem), văn bản được sinh ra từ mô hình này giống ngôn ngữ tự nhiên, giống tiếng Anh chẳng hạn. Hơn nữa, mô hình đủ khả năng tạo ra một đoạn hội thoại có ý nghĩa mà chỉ cần dựa vào đoạn hội thoại trước đó. Trên thực tế, còn rất xa để thiết kế được hệ thống như vậy, vì mô hình sẽ cần hiểu văn bản hơn là chỉ tạo ra nội dung đúng ngữ pháp.

Tuy nhiên, mô hình ngôn ngữ vẫn rất hữu dụng ngay cả khi còn hạn chế. Chẳng hạn, cụm từ "nhận dạng giọng nói" và "nhân gian rộng lối" có phát âm khá giống nhau. Điều này có thể gây ra sự mơ hồ trong việc nhận dạng giọng nói, nhưng có thể dễ dàng được giải quyết với một mô hình ngôn ngữ. Mô hình sẽ loại bỏ ngay phương án thứ hai do mang ý nghĩa kì lạ. Tương tự, một thuật toán tóm tắt tài liệu nên phân biệt được rằng câu "chó cắn người" xuất hiện thường xuyên hơn nhiều so với "người cắn chó", hay như "Cháu muốn ăn bà ngoại" nghe khá kinh dị trong khi "Cháu muốn ăn, bà ngoại" lại là bình thường.

7.3.1. Ước tính một Mô hình Ngôn ngữ

Làm thế nào để mô hình hóa một tài liệu hay thậm chí là một chuỗi các từ? Ta có thể sử dụng cách phân tích đã dùng trong mô hình chuỗi ở phần trước. Bắt đầu bằng việc áp dụng quy tắc xác suất cơ bản sau:

$$P(w_1, w_2, \dots, w_T) = \prod_{t=1}^{T} P(x_t \mid x_1, \dots, x_{t-1}).$$

7.4. Mạng nơ-ron Hồi tiếp

Mục 7.3 đã giới thiệu mô hình n-gram, trong đó xác suất có điều kiện của từ x_t tại vị trí t chỉ phụ thuộc vào n−1 từ trước đó. Nếu muốn kiểm tra ảnh hưởng có thể có của các từ ở trước vị trí t−(n−1) đến từ x_t, ta cần phải tăng n. Tuy nhiên, cùng với đó số lượng tham số của mô hình cũng sẽ tăng lên theo hàm mũ, vì ta cần lưu $|V|^n$ giá trị với một từ điển V nào đó. Do đó, thay vì mô hình hóa p(xt|xt−1,…,xt−n+1), sẽ tốt hơn nếu ta sử dụng mô hình biến tiềm ẩn (latent variable model), trong đó

$$P(x_t \mid x_{t-1}, \dots, x_1) \approx P(x_t \mid h_{t-1}),$$

h_t được gọi là biến tiềm ẩn và nó lưu trữ thông tin của chuỗi. Biến tiềm ẩn còn được gọi là biến ẩn (hidden variable), trạng thái ẩn (hidden state) hay biến trạng thái ẩn (hidden state variable). Trạng thái ẩn tại thời điểm t có thể được tính dựa trên cả đầu vào x_t và trạng thái ẩn h_t−1 như sau

$$h_t = f(x_t, h_{t-1}).$$

Với một hàm f đủ mạnh, mô hình biến tiềm ẩn không phải là một phép xấp xỉ. Sau cùng, h_t có thể chỉ đơn thuần lưu lại tất cả dữ liệu đã quan sát được cho đến thời điểm hiện tại. Điều này đã được thảo luận tại Mục 7.1. Tuy nhiên nó có thể khiến cho việc tính toán và lưu trữ trở nên nặng nề.

Chú ý rằng ta cũng sử dụng h để kí hiệu số lượng nút ẩn trong một tầng ẩn. Tầng ẩn và trạng thái ẩn là hai khái niệm rất khác nhau. Tầng ẩn, như đã được giải thích, là các tầng không thể nhìn thấy trong quá trình đi từ đầu vào đến đầu ra. Trạng thái ẩn, về mặt kỹ thuật là đầu vào của một bước tính toán tại một thời điểm xác định. Chúng chỉ có thể được tính dựa vào dữ liệu tại các vòng lặp trước đó. Về điểm này, trạng thái ẩn giống với các mô hình biến tiềm ẩn trong thống kê như mô hình phân cụm hoặc mô hình chủ đề (topic model), trong đó các cụm tác động đến đầu ra nhưng không thể quan sát trực tiếp.

Mạng nơ-ron hồi tiếp là mạng nơ-ron với các trạng thái ẩn. Trước khi tìm hiểu mô hình này, hãy cùng xem lại perceptron đa tầng tại Mục 4.1.

7.4.1. Mạng Hồi tiếp không có Trạng thái ẩn

Xét một perception đa tầng với một tầng ẩn duy nhất. Giả sử ta có một minibatch $\mathbf{X} \in \mathbb{R}^{n\times d}$ với n mẫu và d đầu vào. Gọi hàm kích hoạt của tầng ẩn là ϕ. Khi đó, đầu ra của tầng ẩn $\mathbf{H} \in \mathbb{R}^{n\times h}$ được tính như sau

$$\mathbf{H} = \phi(\mathbf{X}\mathbf{W}_{xh} + \mathbf{b}_h).$$

Trong đó, $\mathbf{W}_{xh} \in \mathbb{R}^{d\times h}$ là tham số trọng số, $\mathbf{b}_h \in \mathbb{R}^{1\times h}$ là hệ số điều chỉnh và h là số nút ẩn của tầng ẩn.

Biến ẩn $\mathbf{H}$ được sử dụng làm đầu vào của tầng đầu ra. Tầng đầu ra được tính toán bởi

$$\mathbf{O} = \mathbf{H}\mathbf{W}_{hq} + \mathbf{b}_q,$$

Trong đó $\mathbf{O} \in \mathbb{R}^{n\times q}$ là biến đầu ra, $\mathbf{W}_{hq} \in \mathbb{R}^{h\times q}$ là tham số trọng số và $\mathbf{b}_q \in \mathbb{R}^{1\times q}$ là hệ số điều chỉnh của tầng đầu ra. Nếu đang giải quyết bài toán phân loại, ta có thể sử dụng softmax($\mathbf{O}$) để tính phân phối xác suất của các lớp đầu ra.

Do bài toán này hoàn toàn tương tự với bài toán hồi quy được giải quyết trong Mục 7.1, ta sẽ bỏ qua các chi tiết ở đây. Và chỉ cần biết thêm rằng ta có thể chọn các cặp (x_t,x_t−1) một cách ngẫu nhiên và ước lượng các tham số $\mathbf{W}$ và $\mathbf{b}$ của mạng thông qua phép vi phân tự động và hạ gradient ngẫu nhiên.

7.4.2. Mạng Hồi tiếp có Trạng thái ẩn

Vấn đề sẽ khác đi hoàn toàn nếu ta sử dụng các trạng thái ẩn. Hãy xem xét cấu trúc này một cách chi tiết hơn. Chúng ta thường gọi vòng lặp thứ t là thời điểm t trong thuật toán tối ưu, nhưng trong mạng nơ-ron hồi tiếp, thời điểm t lại tương ứng với các bước trong một vòng lặp. Giả sử trong một vòng lặp ta có $\mathbf{X}_t \in \mathbb{R}^{n\times d}$, t=1,…,T. Và $\mathbf{H}_t \in \mathbb{R}^{n\times h}$ là biến ẩn tại bước thời gian t của chuỗi. Khác với perceptron đa tầng, ở đây ta lưu biến ẩn $\mathbf{H}_{t-1}$ từ bước thời gian trước đó và dùng thêm một tham số trọng số mới $\mathbf{W}_{hh} \in \mathbb{R}^{h\times h}$ để mô tả việc sử dụng biến ẩn của bước thời gian trước đó trong bước thời gian hiện tại. Cụ thể, biến ẩn của bước thời gian hiện tại được xác định bởi đầu vào của bước thời gian hiện tại cùng với biến ẩn của bước thời gian trước đó:

$$\mathbf{H}_t = \phi(\mathbf{X}_t\mathbf{W}_{xh} + \mathbf{H}_{t-1}\mathbf{W}_{hh} + \mathbf{b}_h).$$

Phương trình này có thêm $\mathbf{H}_{t-1}\mathbf{W}_{hh}$. Từ mối quan hệ giữa các biến ẩn $\mathbf{H}_t$ và $\mathbf{H}_{t-1}$ của các bước thời gian liền kề, ta biết rằng chúng đã lưu lại thông tin lịch sử của chuỗi cho tới bước thời gian hiện tại, giống như trạng thái hay bộ nhớ hiện thời của mạng nơ-ron. Vì vậy, một biến ẩn còn được gọi là một trạng thái ẩn (hidden state). Vì trạng thái ẩn ở bước thời gian hiện tại và

trước đó đều có cùng định nghĩa, phương trình trên được tính toán theo phương pháp hồi tiếp. Và đây cũng là lý do dẫn đến cái tên mạng nơ-ron hồi tiếp (Recurrent Neural Network - RNN).

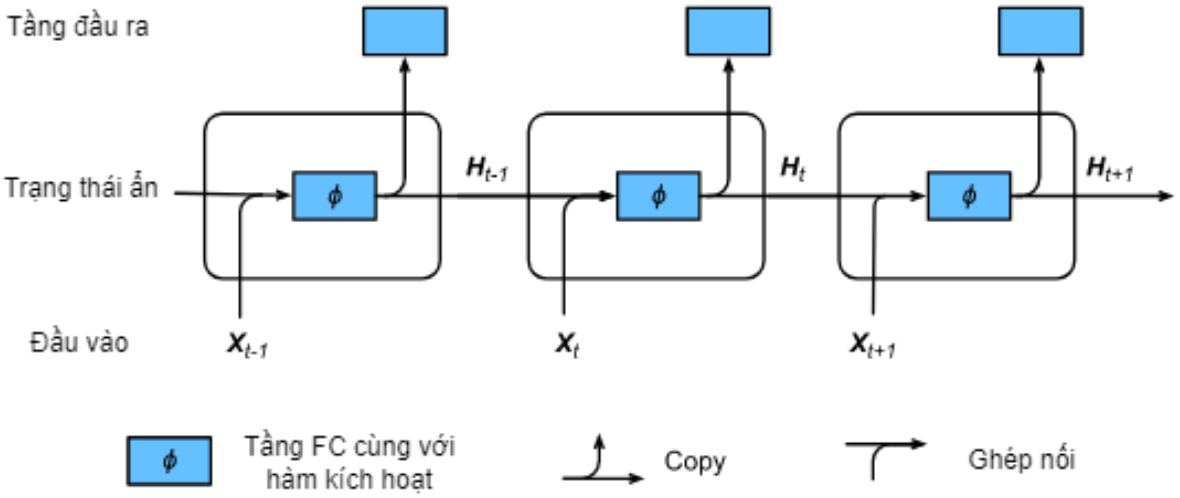
Có rất nhiều phương pháp xây dựng RNN. Trong số đó, phổ biến nhất là RNN có trạng thái ẩn như định nghĩa ở phương trình trên. Tại bước thời gian t, tầng đầu ra trả về kết quả tính toán tương tự như trong perceptron đa tầng:

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q.$$

Các tham số trong mô hình RNN bao gồm trọng số $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}, \mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$, của tầng ẩn với hệ số điều chỉnh $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$, và trọng số $\mathbf{W}_{hq} \in \mathbb{R}^{h \times q}$ của tầng đầu ra với hệ số điều chỉnh $\mathbf{b}_q \in \mathbb{R}^{1 \times q}$. Lưu ý rằng RNN luôn sử dùng cùng một bộ tham số mô hình cho dù tính toán ở các bước thời gian khác nhau. Vì thế, việc tăng số bước thời gian không làm tăng lượng tham số mô hình của RNN.

Hình 7.7 minh họa logic tính toán của một RNN tại ba bước thời gian liền kề. Tại bước thời gian t, sau khi nối đầu vào X_t với trạng thái ẩn H_{t-1} tại bước thời gian trước đó, ta có thể coi nó như đầu vào của một tầng kết nối đầy đủ với hàm kích hoạt ϕ.

Đầu ra của tầng kết nối đầy đủ chính là trạng thái ẩn ở bước thời gian hiện tại H_t. Tham số mô hình ở bước thời gian hiện tại là W_{xh} nối với W_{hh}, cùng với hệ số điều chỉnh b_h. Trạng thái ẩn ở bước thời gian hiện tại t, H_t được sử dụng để tính trạng thái ẩn H_{t+1} tại bước thời gian kế tiếp t+1. Hơn nữa, H_t sẽ trở thành đầu vào cho tầng đầu ra O_t, một tầng kết nối đầy đủ, ở bước thời gian hiện tại.



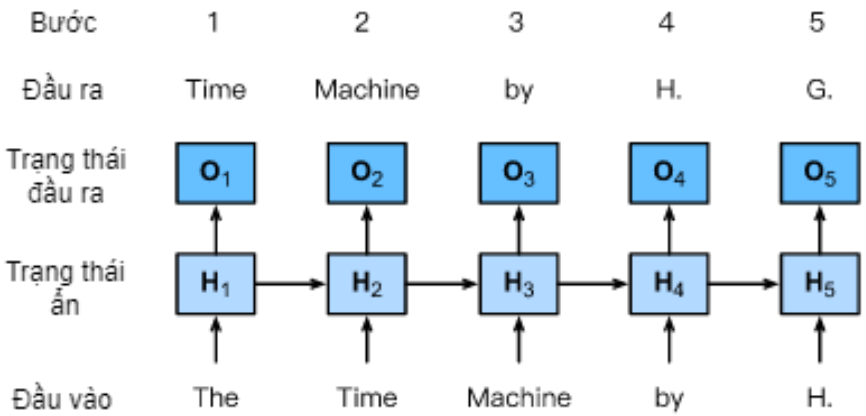
Hình 7.7. Một RNN với một trạng thái ẩn.

7.4.3. Các bước trong một Mô hình Ngôn ngữ

Bây giờ hãy cùng xem cách xây dựng mô hình ngôn ngữ bằng RNN. Vì dùng từ thường dễ hiểu hơn dùng chữ, nên các từ sẽ được dùng làm đầu vào trong ví dụ đơn giản này.

Đặt kích thước minibatch là 1, với chuỗi văn bản là phần đầu của tập dữ liệu: "the time machine by H. G. Wells". Hình 7.8 minh họa cách ước lượng từ tiếp theo dựa trên từ hiện tại và các từ trước đó. Trong quá trình huấn luyện, chúng ta áp dụng softmax cho đầu ra tại mỗi bước thời gian, sau đó sử dụng hàm mất mát entropy chéo để tính toán sai số giữa kết quả và nhãn. Do trạng thái ẩn trong tầng ẩn được tính toán hồi tiếp, đầu ra của bước thời gian thứ 3, O_3, được xác định bởi chuỗi các từ "the", "time" và "machine". Vì từ tiếp theo của chuỗi trong dữ liệu huấn luyện là "by", giá trị mất mát tại bước thời gian thứ 3 sẽ phụ thuộc vào phân phối xác suất của từ

tiếp theo được tạo dựa trên chuỗi đặc trưng “the”, “time”, “machine” và nhãn “by” tại bước thời gian này.



Hình 7.8. Mô hình ngôn ngữ ở mức từ ngữ RNN. Đầu vào và chuỗi nhãn lần lượt là the time machine by H. và time machine by H. G.

Trong thực tế, mỗi từ được biểu diễn bởi một vector d chiều và kích thước batch thường là n>1. Do đó, đầu vào X_t tại bước thời gian t sẽ là ma trận n×d, giống hệt với những gì chúng ta đã thảo luận trước đây.

7.4.4. Perplexity

Cuối cùng, hãy thảo luận về cách đo lường chất lượng của mô hình chuỗi. Một cách để làm việc này là kiểm tra mức độ gây ngạc nhiên của văn bản. Một mô hình ngôn ngữ tốt có thể dự đoán chính xác các token tiếp theo. Hãy xem xét các cách điền tiếp vào câu “Trời đang mưa” sau, được đề xuất bởi các mô hình ngôn ngữ khác nhau:

1. “Trời đang mưa bên ngoài”
2. “Trời đang mưa cây chuối”
3. “Trời đang mưa piouw;kcj pwepoiut”

Về chất lượng, ví dụ 1 rõ ràng là tốt nhất. Các từ được sắp xếp hợp lý và mạch lạc về mặt logic. Mặc dù nó có thể không phản ánh chính xác hoàn toàn mặt ngữ nghĩa của các từ theo sau (“ở San Francisco” và “vào mùa đông” cũng là các phần mở rộng hoàn toàn hợp lý), mô hình vẫn có thể nắm bắt những từ nghe khá phù hợp. Ví dụ 2 thì tệ hơn đáng kể, mô hình này đã nối dài câu ra theo cách vô nghĩa. Tuy nhiên, ít nhất mô hình đã viết đúng chính tả và học được phần nào sự tương quan giữa các từ. Cuối cùng, ví dụ 3 là một mô hình được huấn luyện kém, không khớp được dữ liệu.

Chúng ta có thể đo lường chất lượng của mô hình bằng cách tính xác suất p(w), tức độ hợp lý của một chuỗi w. Thật không may, đây là một con số khó để hiểu và so sánh. Xét cho cùng, các chuỗi ngắn có khả năng xuất hiện cao hơn các chuỗi dài, do đó việc đánh giá mô hình trên kiệt tác “Chiến tranh và Hòa bình” của Tolstoy chắc chắn sẽ cho kết quả thấp hơn nhiều so với tiểu thuyết “Hoàng tử bé” của Saint-Exupery. Thứ còn thiếu ở đây là một cách tính trung bình qua độ dài chuỗi.

Lý thuyết thông tin rất có ích trong trường hợp này. Nếu chúng ta muốn nén văn bản, ta có thể yêu cầu ước lượng ký hiệu tiếp theo với bộ ký hiệu hiện tại. Số lượng bit tối thiểu cần thiết là $-\log_2 p(x_t|x_{t-1},\ldots,x_1)$. Một mô hình ngôn ngữ tốt sẽ cho phép chúng ta dự đoán từ tiếp theo một

cách khá chính xác và do đó số bit cần thiết để nén chuỗi là rất thấp. Vì vậy, ta có thể đo lường mô hình ngôn ngữ bằng số bit trung bình cần sử dụng.

$$\frac{1}{n}\sum_{t=1}^{n} -\log P(x_t \mid x_{t-1}, \ldots, x_1),$$

Điều này giúp ta so sánh được chất lượng mô hình trên các tài liệu có độ dài khác nhau. Vì lý do lịch sử, các nhà khoa học xử lý ngôn ngữ tự nhiên thích sử dụng một đại lượng gọi là perplexity (độ rối rắm, hỗn độn) thay vì bitrate (tốc độ bit). Nói ngắn gọn, nó là luỹ thừa của biểu thức trên:

$$\text{PLL} := \exp\left(-\frac{1}{n}\sum_{t=1}^{n} \log P(x_t \mid x_{t-1}, \ldots, x_1)\right).$$

Giá trị này có thể được hiểu rõ nhất như là trung bình điều hòa của số lựa chọn thực tế mà ta có khi quyết định chọn từ nào là từ tiếp theo. Lưu ý rằng perplexity khái quát hóa một cách tự nhiên ý tưởng của hàm mất mát entropy chéo định nghĩa ở phần hồi quy softmax (Section 3.4). Điều này có nghĩa là khi xét một ký hiệu duy nhất, perplexity chính là lũy thừa của entropy chéo. Hãy cùng xem xét một số trường hợp:

- Trong trường hợp tốt nhất, mô hình luôn ước tính xác suất của ký hiệu tiếp theo là 1. Khi đó perplexity của mô hình là 1.
- Trong trường hợp xấu nhất, mô hình luôn dự đoán xác suất của nhãn là 0. Khi đó perplexity là vô hạn.
- Tại mức nền, mô hình dự đoán một phân phối đều trên tất cả các token. Trong trường hợp này, perplexity bằng với kích thước của từ điển len(vocab).
- Thực chất, nếu chúng ta lưu trữ chuỗi không nén, đây là cách tốt nhất có thể để mã hóa chúng. Vì vậy, nó cho ta một cận trên mà bất kỳ mô hình nào cũng phải thỏa mãn.

7.4.5. Kết luận

- Một mạng sử dụng tính toán hồi tiếp được gọi là mạng nơ-ron hồi tiếp (RNN).

- Trạng thái ẩn của RNN có thể tổng hợp được thông tin lịch sử của chuỗi cho tới bước thời gian hiện tại.

- Số lượng tham số của mô hình RNN không tăng khi số lượng bước thời gian tăng.

- Ta có thể tạo các mô hình ngôn ngữ sử dụng một RNN ở cấp độ ký tự.

7.5. Lan truyền Ngược qua Thời gian

Cho đến nay chúng ta liên tục nhắc đến những vấn đề như bùng nổ gradient, tiêu biến gradient, cắt xén lan truyền ngược và sự cần thiết của việc tách đồ thị tính toán. Ví dụ, trong phần trước chúng ta gọi hàm s.detach() trên chuỗi. Vì muốn nhanh chóng xây dựng và quan sát cách một mô hình hoạt động nên những vấn đề này chưa được giải thích một cách đầy đủ. Trong phần này chúng ta sẽ nghiên cứu sâu và chi tiết hơn về lan truyền ngược cho các mô hình chuỗi và giải thích nguyên lý toán học đằng sau. Để hiểu chi tiết hơn về tính ngẫu nhiên và lan truyền ngược, hãy tham khảo bài báo [Tallec & Ollivier, 2017].

Để có cái nhìn rõ hơn về vấn đề này, trong phần này chúng ta sẽ xem xét cách tính gradient cho các mô hình chuỗi. Lưu ý rằng, về mặt khái niệm thì không có gì mới ở đây. Sau cùng, chúng ta vẫn chỉ đơn thuần áp dụng các quy tắc dây chuyền để tính gradient. Tuy nhiên, việc ôn lại lan truyền ngược vẫn rất hữu ích.

Lượt truyền xuôi trong mạng nơ-ron hồi tiếp tương đối đơn giản. Lan truyền ngược qua thời gian thực chất là một ứng dụng cụ thể của lan truyền ngược trong mạng nơ-ron hồi tiếp. Nó đòi hỏi chúng ta mở rộng mạng nơ-ron hồi tiếp theo từng bước thời gian một để thu được sự phụ thuộc giữa các biến mô hình và các tham số. Sau đó, dựa trên quy tắc dây chuyền, chúng ta áp dụng lan truyền ngược để tính toán và lưu các giá trị gradient. Vì chuỗi có thể khá dài nên sự phụ thuộc trong chuỗi cũng có thể rất dài. Ví dụ, đối với một chuỗi gồm 1000 ký tự, ký tự đầu tiên có thể ảnh hưởng đáng kể tới ký tự ở vị trí 1000. Điều này không thực sự khả thi về mặt tính toán (cần quá nhiều thời gian và bộ nhớ) và nó đòi hỏi hơn 1000 phép nhân ma trận-vector trước khi thu được các giá trị gradient khó nắm bắt này. Đây là một quá trình chứa đầy sự bất định về mặt tính toán và thống kê. Trong phần tiếp theo chúng ta sẽ làm sáng tỏ những gì sẽ xảy ra và cách giải quyết vấn đề này trong thực tế.

7.5.1. Mạng Hồi tiếp Giản thể

Hãy bắt đầu với một mô hình đơn giản về cách mà mạng RNN hoạt động. Mô hình này bỏ qua các chi tiết cụ thể của trạng thái ẩn và cách trạng thái này được cập nhật. Những chi tiết này không quan trọng đối với việc phân tích dưới đây mà chỉ khiến các ký hiệu trở nên lộn xộn và phức tạp quá mức. Trong mô hình đơn giản này, chúng ta ký hiệu h_t là trạng thái ẩn, x_t là đầu vào, và o_t là đầu ra tại bước thời gian t. Bên cạnh đó, w_h và w_o tương ứng với trọng số của các trạng thái ẩn và tầng đầu ra. Kết quả là, các trạng thái ẩn và kết quả đầu ra tại mỗi bước thời gian có thể được giải thích như sau

$$\begin{aligned} h_t &= f(x_t, h_{t-1}, w_h), \\ o_t &= g(h_t, w_o), \end{aligned}$$

Do đó, chúng ta có một chuỗi các giá trị $\{\dots, (x_{t-1}, h_{t-1}, o_{t-1}), (x_t, h_t, o_t), \dots\}$ phụ thuộc vào nhau thông qua phép tính đệ quy. Lượt truyền xuôi khá đơn giản. Những gì chúng ta cần là lặp qua từng bộ ba (x_t, h_t, o_t) một. Sau đó, sự khác biệt giữa kết quả đầu ra o_t và các giá trị mục tiêu mong muốn y_t được tính bằng một hàm mục tiêu

$$L(x_1, \dots, x_T, y_1, \dots, y_T, w_h, w_o) = \frac{1}{T}\sum_{t=1}^{T} l(y_t, o_t).$$

Đối với lan truyền ngược, mọi thứ lại phức tạp hơn một chút, đặc biệt là khi chúng ta tính gradient theo các tham số w_h của hàm mục tiêu L. Cụ thể, theo quy tắc dây chuyền ta có

$$\begin{aligned} \frac{\partial L}{\partial w_h} &= \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial w_h} \\ &= \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t} \frac{\partial g(h_t, w_o)}{\partial h_t} \frac{\partial h_t}{\partial w_h}. \end{aligned}$$

Ta có thể tính phần đầu tiên và phần thứ hai của đạo hàm một cách dễ dàng. Phần thứ ba $\partial h_t / \partial w_h$ khiến mọi thứ trở nên khó khăn, vì chúng ta cần phải tính toán ảnh hưởng của các tham số tới h_t.

Để tính được gradient ở trên, giả sử rằng chúng ta có ba chuỗi $\{a_t\}, \{b_t\}, \{c_t\}$ thỏa mãn $a_0=0, a_1=b_1$ và $a_t=b_t+c_t a_t-1$ với $t=1,2,\ldots$. Sau đó, với $t\geq 1$ ta có

$$a_t = b_t + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} c_j\right) b_i.$$

Bây giờ chúng ta áp dụng với

$$\begin{aligned} a_t &= \frac{\partial h_t}{\partial w_h}, \\ b_t &= \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h}, \\ c_t &= \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}}, \end{aligned}$$

Vì vậy, công thức $a_t = b_t + c_t a_{t-1}$ trở thành phép đệ quy dưới đây

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h}.$$

phần thứ ba sẽ trở thành

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}}\right) \frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}.$$

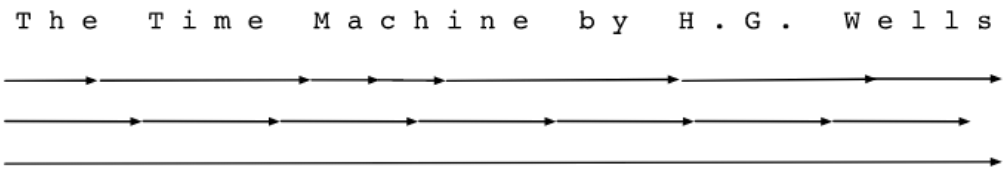
Dù chúng ta có thể sử dụng quy tắc dây chuyền để tính $\partial_w h_t$ một cách đệ quy, dây chuyền này có thể trở nên rất dài khi giá trị t lớn. Hãy cùng thảo luận về một số chiến lược để giải quyết vấn đề này.

- Tính toàn bộ tổng. Cách này rất chậm và gradient có thể bùng nổ vì những thay đổi nhỏ trong các điều kiện ban đầu cũng có khả năng ảnh hưởng đến kết quả rất nhiều. Điều này tương tự như trong hiệu ứng cánh bướm, khi những thay đổi rất nhỏ trong điều kiện ban đầu dẫn đến những thay đổi không cân xứng trong kết quả. Đây thực sự là điều không mong muốn khi xét tới mô hình mà chúng ta muốn ước lượng. Sau cùng, chúng ta đang cố tìm kiếm một bộ ước lượng mạnh mẽ và có khả năng khái quát tốt. Do đó chiến lược này hầu như không bao giờ được sử dụng trong thực tế.
- Cắt xén tổng sau τ bước. Cho đến giây phút hiện tại, đây là những gì chúng ta đã thảo luận. Điều này dẫn tới một phép xấp xỉ của gradient, đơn giản bằng cách kết thúc tổng trên tại $\partial_w h_{t-\tau}$. Do đó lỗi xấp xỉ là $\partial_h f(x_t, h_{t-1}, w) \partial_w h_t - 1$ (nhân với tích của gradient liên quan đến ∂hf). Trong thực tế, chiến lược này hoạt động khá tốt. Phương pháp này thường được gọi là BPTT (backpropagation through time — lan truyền ngược qua thời gian) bị cắt xén. Một trong những hệ quả của phương pháp này là mô hình sẽ tập trung chủ yếu vào ảnh hưởng ngắn hạn thay vì dài hạn. Đây thực sự là điều mà chúng ta mong muốn, vì nó hướng sự ước lượng tới các mô hình đơn giản và ổn định hơn.
- Cắt xén Ngẫu nhiên. Cuối cùng, chúng ta có thể thay thế $\partial_{wh} h_t$ bằng một biến ngẫu nhiên có giá trị kỳ vọng đúng nhưng vẫn cắt xén chuỗi.

- Điều này có thể đạt được bằng cách sử dụng một chuỗi các ξ_t trong đó $E[\xi t]=1$, $P(\xi_t=0)=1-\pi$ và $P(\xi_t=\pi-1)=\pi$.
- Chúng ta sẽ sử dụng chúng thay vì gradient:

$$z_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \xi_t \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h}.$$

Từ định nghĩa của ξ_t, ta có $E[z_t]=\partial_w h_t$. Bất cứ khi nào $\xi_t=0$, khai triển sẽ kết thúc tại điểm đó. Điều này dẫn đến một tổng trọng số của các chuỗi có chiều dài biến thiên, trong đó chuỗi dài sẽ hiếm hơn nhưng được đánh trọng số cao hơn tương ứng. [Tallec & Ollivier, 2017] đưa ra đề xuất này trong bài báo nghiên cứu của họ. Không may, dù phương pháp này khá hấp dẫn về mặt lý thuyết, nó lại không tốt hơn phương pháp cắt xén đơn giản, nhiều khả năng do các yếu tố sau. Thứ nhất, tác động của một quan sát đến quá khứ sau một vài lượt lan truyền ngược đã là tương đối đủ để nắm bắt các phụ thuộc trên thực tế. Thứ hai, phương sai tăng lên làm phản tác dụng của việc có gradient chính xác hơn. Thứ ba, ta thực sự muốn các mô hình có khoảng tương tác ngắn. Do đó, BPTT có một hiệu ứng điều chuẩn nhỏ mà có thể có ích.



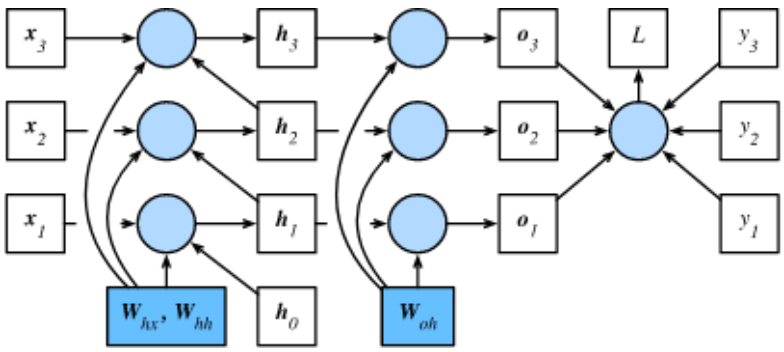
Hình 7.9. Từ trên xuống dưới: BPTT ngẫu nhiên, BPTT bị cắt xén đều và BPTT đầy đủ

Hình 7.9 minh họa ba trường hợp trên khi phân tích một số từ đầu tiên trong Cỗ máy Thời gian:

- Dòng đầu tiên biểu diễn sự cắt xén ngẫu nhiên, chia văn bản thành các phần có độ dài biến thiên.
- Dòng thứ hai biểu diễn BPTT bị cắt xén đều, chia văn bản thành các phần có độ dài bằng nhau.
- Dòng thứ ba là BPTT đầy đủ, dẫn đến một biểu thức không khả thi về mặt tính toán.

7.5.2. Đồ thị Tính toán

Để minh họa trực quan sự phụ thuộc giữa các biến và tham số mô hình trong suốt quá trình tính toán của mạng nơ-ron hồi tiếp, ta có thể vẽ đồ thị tính toán của mô hình, như trong Hình 7.10. Ví dụ, việc tính toán trạng thái ẩn ở bước thời gian 3, h_3, phụ thuộc vào các tham số $\mathbf{W}_{hx}$ và $\mathbf{W}_{hh}$ của mô hình, trạng thái ẩn ở bước thời gian trước đó h_2, và đầu vào ở bước thời gian hiện tại x_3.



Hình 7.10. Sự phụ thuộc về mặt tính toán của mạng nơ-ron hồi tiếp với ba bước thời gian. Ô vuông tượng trưng cho các biến (không tô đậm) hoặc các tham số (tô đậm), hình tròn tượng trưng cho các phép toán.

7.5.3. BPTT chi tiết

Sau khi thảo luận các nguyên lý chung, hãy phân tích BPTT một cách chi tiết. Bằng cách tách $\mathbf{W}$ thành các tập ma trận trọng số khác nhau $\mathbf{W}_{hx}$,$\mathbf{W}_{hh}$ và $\mathbf{W}_{oh}$), ta thu được mô hình biến tiềm ẩn tuyến tính đơn giản:

$$\begin{aligned}\mathbf{h}_t &= \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1},\\ \mathbf{o}_t &= \mathbf{W}_{qh}\mathbf{h}_t,\end{aligned}$$

Ta tính các gradient $\partial L/\partial \mathbf{W}_{hx}$, $\partial L/\partial \mathbf{W}_{hh}$ và $\partial L/\partial \mathbf{W}_{qh}$ cho

$$L(x, y, \boldsymbol{W}) = \frac{1}{T}\sum_{t=1}^{T} l(\mathbf{o}_t, y_t).$$

Với $l(\cdot)$ là hàm mất mát đã chọn trước. Tính đạo hàm theo $\mathbf{W}_{qh}$ khá đơn giản, ta có

$$\frac{\partial L}{\partial \mathbf{W}_{qh}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \mathbf{o}_t}, \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_{qh}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{o}_t}\mathbf{h}_t^\top,$$

với prod(·) là tích của hai hoặc nhiều ma trận.

Sự phụ thuộc vào $\mathbf{W}_{hx}$ và $\mathbf{W}_{hh}$ thì khó khăn hơn một chút vì cần sử dụng quy tắc dây chuyền khi tính toán đạo hàm. Ta sẽ bắt đầu với

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{W}_{hx}} &= \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \mathbf{h}_t}, \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{hx}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t}\mathbf{x}_t^\top,\\ \frac{\partial L}{\partial \mathbf{W}_{hh}} &= \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \mathbf{h}_t}, \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{hh}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t}\mathbf{h}_{t-1}^\top,\end{aligned}$$

Sau cùng, các trạng thái ẩn phụ thuộc lẫn nhau và phụ thuộc vào đầu vào quá khứ. Một đại lượng quan trọng là sư ảnh hưởng của các trạng thái ẩn quá khứ tới các trạng thái ẩn tương lai.

$$\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} = W_{hh}^T$$

Do đó

$$\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} = (W_{hh}^T)^{T-t}$$

Áp dụng quy tăc dây chuyền, ta được

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{hx}} = \sum_{j=1}^{t} (\boldsymbol{W}_{hh}^T)^{t-j}\, x_j$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{hh}} = \sum_{j=1}^{t} (\boldsymbol{W}_{hh}^T)^{t-j}\, h_j$$

Ta có thể rút ra nhiều điều từ biểu thức phức tạp này. Đầu tiên, việc lưu lại các kết quả trung gian, tức các luỹ thừa của $\mathbf{W}_{hh}$ khi tính các số hạng của hàm mất mát L, là rất hữu ích. Thứ hai, ví dụ tuyến tính này dù đơn giản nhưng đã làm lộ ra một vấn đề chủ chốt của các mô hình chuỗi dài: ta có thể phải làm việc với các luỹ thừa rất lớn của $\mathbf{W}_{hh}^{j}$. Trong đó, khi j lớn, các trị riêng nhỏ hơn 1 sẽ tiêu biến, còn các trị riêng lớn hơn 1 sẽ phân kì. Các mô hình này không có tính ổn định số học, dẫn đến việc chúng quan trọng hóa quá mức các chi tiết không liên quan trong quá khứ. Một cách giải quyết vấn đề này là cắt xén các số hạng trong tổng ở một mức độ thuận tiện cho việc tính toán. Sau này ở Section 9, ta sẽ thấy cách các mô hình chuỗi phức tạp như LSTM giải quyết vấn đề này tốt hơn. Khi lập trình, ta cắt xén các số hạng bằng cách tách rời gradient sau một số lượng bước nhất định.

7.5.4. Kết luận

- Lan truyền ngược theo thời gian chỉ là việc áp dụng lan truyền ngược cho các mô hình chuỗi có trạng thái ẩn.

- Việc cắt xén là cần thiết để thuận tiện cho việc tính toán và ổn định các giá trị số.

- Luỹ thừa lớn của ma trận có thể làm các trị riêng tiêu biến hoặc phân kì, biểu hiện dưới hiện tượng tiêu biến hoặc bùng nổ gradient.

- Để tăng hiệu năng tính toán, các giá trị trung gian được lưu lại.

7.6. Mạng Nơ-ron Hồi tiếp Hiện đại

Mặc dù đã biết về các kiến thức cơ bản của mạng nơ-ron hồi tiếp, chúng vẫn chưa đủ để ta giải quyết các bài toán học chuỗi hiện nay. Ví dụ như RNN có hiện tượng bất ổn số học khi tính gradient, do đó các mạng nơ-ron hồi tiếp có cổng được sử dụng phổ biến hơn nhiều trong thực tiễn. Chúng ta bắt đầu chương này bằng việc giới thiệu hai cấu trúc mạng phổ biến: nút hồi tiếp có cổng (gated recurrent unit - GRU) và bộ nhớ ngắn hạn dài (long short term memory - LSTM).

Hơn nữa, chúng ta sẽ sửa đổi mạng nơ-ron hồi tiếp với một tầng ẩn đơn chiều. Ta cũng sẽ mô tả các kiến trúc mạng sâu và thảo luận về thiết kế hai chiều (bidirectional) gồm cả hồi tiếp xuôi và ngược. Chúng thường xuyên được sử dụng trong các mạng nơ-ron hồi tiếp hiện đại.

Trên thực tế, phần lớn các bài toán học chuỗi như nhận dạng giọng nói tự động, chuyển đổi văn bản thành giọng nói và dịch máy, đều có đầu vào và đầu ra là các chuỗi với chiều dài bất kì. Cuối cùng, ta sẽ lấy bài toán dịch máy làm ví dụ để giới thiệu kiến trúc mã hóa - giải mã (encoder-decoder) dựa trên mạng nơ-ron hồi tiếp cùng các kỹ thuật hiện đại để giải quyết bài toán học từ chuỗi sang chuỗi.

7.6.1. Nút Hồi tiếp có Cổng (GRU)

Trong phần trước, chúng ta đã thảo luận cách tính gradient trong mạng nơ-ron hồi tiếp. Cụ thể ta đã biết rằng tích của một chuỗi dài các ma trận có thể dẫn đến việc gradient tiêu biến hoặc bùng nổ. Hãy điểm qua các tình huống thực tế thể hiện rõ hai bất thường đó:

Ta có thể gặp tình huống mà những quan sát xuất hiện sớm có ảnh hưởng lớn đến việc dự đoán toàn bộ những quan sát trong tương lai. Xét một ví dụ có chút cường điệu, trong đó quan sát đầu tiên chứa giá trị tổng kiểm (checksum) và mục tiêu là kiểm tra xem liệu giá trị tổng kiểm đó có đúng hay không tại cuối chuỗi. Trong trường hợp này, ảnh hưởng của token đầu tiên là tối quan trọng. Do đó ta muốn có cơ chế để lưu trữ những thông tin quan trọng ban đầu trong ô nhớ. Nếu không, ta sẽ phải gán một giá trị gradient cực lớn cho quan sát ban đầu vì nó ảnh hưởng đến toàn bộ các quan sát tiếp theo.

Một tình huống khác là khi một vài ký hiệu không chứa thông tin phù hợp. Ví dụ, khi phân tích một trang web, ta có thể gặp các mã HTML không giúp ích gì cho việc xác định thông tin được truyền tải. Do đó, ta cũng muốn có cơ chế để bỏ qua những ký hiệu như vậy trong các biểu diễn trạng thái tiềm ẩn.

Ta cũng có thể gặp những khoảng ngắt giữa các phần trong một chuỗi. Ví dụ như những phần chuyển tiếp giữa các chương của một quyển sách, hay chuyển biến xu hướng giữa thị trường giá lên và thị trường giá xuống trong chứng khoán. Trong trường hợp này, sẽ tốt hơn nếu có một cách để xóa hay đặt lại các biểu diễn trạng thái ẩn về giá trị ban đầu.

Nhiều phương pháp đã được đề xuất để giải quyết những vấn đề trên. Một trong những phương pháp ra đời sớm nhất là Bộ nhớ ngắn hạn dài (Long Short Term Memory - LSTM) [Hochreiter & Schmidhuber, 1997]. Nút Hồi tiếp có Cổng (Gated Recurrent Unit - GRU) [Cho et al., 2014] là một biến thể gọn hơn của LSTM, thường có chất lượng tương đương và tính toán nhanh hơn đáng kể. Tham khảo [Chung et al., 2014] để biết thêm chi tiết. Trong chương này, ta sẽ bắt đầu với GRU do nó đơn giản hơn.

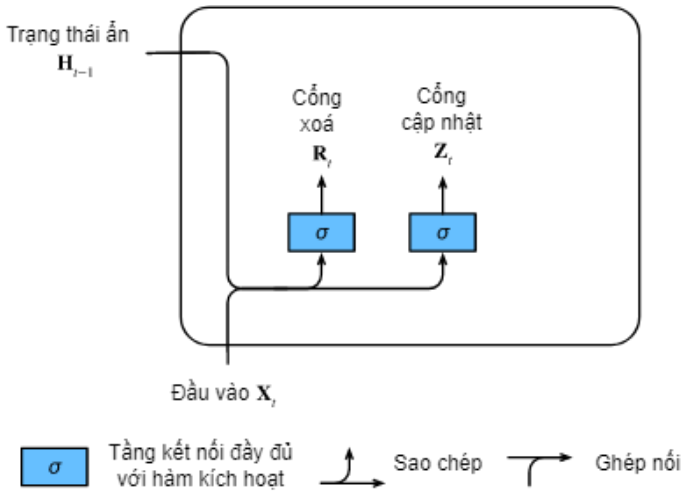
7.6.1.1. Kiểm soát Trạng thái Ẩn

Sự khác biệt chính giữa RNN thông thường và GRU là GRU hỗ trợ việc kiểm soát trạng thái ẩn. Điều này có nghĩa là ta có các cơ chế được học để quyết định khi nào nên cập nhật và khi nào nên xóa trạng thái ẩn. Ví dụ, nếu ký tự đầu tiên có mức độ quan trọng cao, mô hình sẽ học để không cập nhật trạng thái ẩn sau lần quan sát đầu tiên. Tương tự, mô hình sẽ học cách bỏ qua những quan sát tạm thời không liên quan, cũng như cách xóa trạng thái ẩn khi cần thiết. Dưới đây ta sẽ thảo luận chi tiết vấn đề này.

7.6.1.1.1. Cổng Xóa và Cổng Cập nhật

Đầu tiên ta giới thiệu cổng xóa và cổng cập nhật. Ta thiết kế chúng thành các vector có các phần tử trong khoảng (0,1) để có thể biểu diễn các tổ hợp lồi. Chẳng hạn, một biến xóa cho phép kiểm soát bao nhiêu phần của trạng thái trước đây được giữ lại. Tương tự, một biến cập nhật cho phép kiểm soát bao nhiêu phần của trạng thái mới sẽ giống trạng thái cũ.

Ta bắt đầu bằng việc thiết kế các cổng tạo ra các biến này. Hình 7.11 minh họa các đầu vào cho cả cổng xóa và cổng cập nhật trong GRU, với đầu vào ở bước thời gian hiện tại $\mathbf{X}_t$ và trạng thái ẩn ở bước thời gian trước đó $\mathbf{H}_{t-1}$. Đầu ra được tạo bởi một tầng kết nối đầy đủ với hàm kích hoạt sigmoid.



Hình 7.11. Cổng xóa và cổng cập nhật trong GRU.

Tại bước thời gian t, với đầu vào minibatch là $\mathbf{X}_t \in \mathbb{R}^{n\times d}$ (số lượng mẫu: n, số lượng đầu vào: d) và trạng thái ẩn ở bước thời gian gần nhất là $\mathbf{H}_{t-1} \in \mathbb{R}^{n\times h}$ (số lượng trạng thái ẩn: h), cổng xóa $\mathbf{R}_t \in \mathbb{R}^{n\times h}$ và cổng cập nhật $\mathbf{Z}_t \in \mathbb{R}^{n\times h}$ được tính như sau:

$$\mathbf{R}_t = \sigma(\mathbf{X}_t\mathbf{W}_{xr} + \mathbf{H}_{t-1}\mathbf{W}_{hr} + \mathbf{b}_r),$$
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t\mathbf{W}_{xz} + \mathbf{H}_{t-1}\mathbf{W}_{hz} + \mathbf{b}_z),$$

Ở đây, $\mathbf{W}_{xr}, \mathbf{W}_{xz} \in \mathbb{R}^{d\times h}$ và $\mathbf{W}_{hr}, \mathbf{W}_{hz} \in \mathbb{R}^{h\times h}$là các tham số trọng số và $\mathbf{b}_h \in \mathbb{R}^{1\times h}$ là các hệ số điều chỉnh. Ta sẽ sử dụng hàm sigmoid (như trong Section 4.1) để biến đổi các giá trị đầu vào nằm trong khoảng (0,1).

7.6.1.1.2. Hoạt động của Cổng Xóa

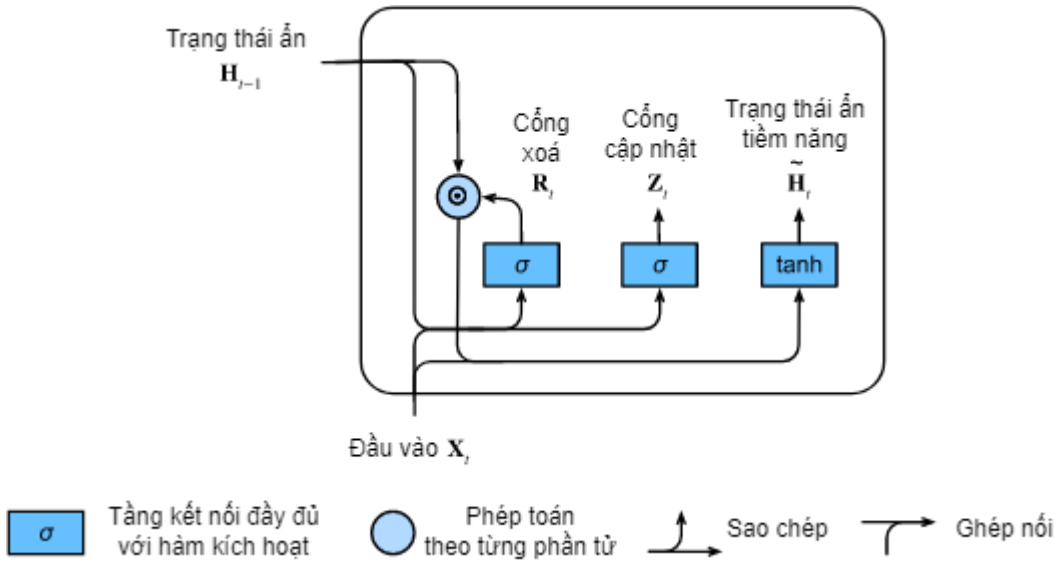
Ta bắt đầu bằng việc tích hợp cổng xóa với một cơ chế cập nhật trạng thái tiềm ẩn thông thường. Trong RNN thông thường, ta cập nhật trạng thái ẩn theo công thức

$$\boldsymbol{H_t} = \tanh(\mathbf{X}_t\mathbf{W}_{xh} + \mathbf{H}_{t-1}\mathbf{W}_{hh} + \mathbf{b}_h),$$

Điều này về cơ bản giống với những gì đã thảo luận ở phần trước, mặc dù có thêm tính phi tuyến dưới dạng hàm tanh để đảm bảo rằng các giá trị trạng thái ẩn nằm trong khoảng $(-1,1)$. Nếu muốn giảm ảnh hưởng của các trạng thái trước đó, ta có thể nhân $\mathbf{H}_{t-1}$ với $\mathbf{R}_t$ theo từng phần tử. Nếu các phần tử trong cổng xóa $\mathbf{R}_t$ có giá trị gần với 1, kết quả sẽ giống RNN thông thường. Nếu tất cả các phần tử của cổng xóa $\mathbf{R}_t$ gần với 0, trạng thái ẩn sẽ là đầu ra của một perceptron đa tầng với đầu vào là $\mathbf{X}_t$. Bất kỳ trạng thái ẩn nào tồn tại trước đó đều được đặt lại về giá trị mặc định. Tại đây nó được gọi là trạng thái ẩn tiềm năng, và chỉ là tiềm năng vì ta vẫn cần kết hợp thêm đầu ra của cổng cập nhật.

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t\mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1})\mathbf{W}_{hh} + \mathbf{b}_h),$$

Hình 7.12 minh họa luồng tính toán sau khi áp dụng cổng xóa. Ký hiệu $\odot$ biểu thị phép nhân theo từng phần tử giữa các tensor.



Hình 7.12. Tính toán của trạng thái ẩn tiềm năng trong một GRU. Phép nhân được thực hiện theo phần tử.

Nếu các giá trị trong cổng cập nhật $\mathbf{Z}_t$ bằng 1, chúng ta chỉ đơn giản giữ lại trạng thái cũ. Trong trường hợp này, thông tin từ $\mathbf{X}_t$ về cơ bản được bỏ qua, tương đương với việc bỏ qua bước thời gian t trong chuỗi phụ thuộc. Ngược lại, nếu $\mathbf{Z}_t$ gần giá trị 0, trạng thái ẩn $\mathbf{H}_t$ sẽ gần với trạng thái

ẩn tiềm năng $\tilde{\mathbf{H}}_t$. Những thiết kế trên có thể giúp chúng ta giải quyết vấn đề tiêu biến gradient trong các mạng RNN và nắm bắt tốt hơn sự phụ thuộc xa trong chuỗi thời gian. Tóm lại, các mạng GRU có hai tính chất nổi bật sau:

- Cổng xóa giúp nắm bắt các phụ thuộc ngắn hạn trong chuỗi thời gian.
- Cổng cập nhật giúp nắm bắt các phụ thuộc dài hạn trong chuỗi thời gian.

7.6.1.4. Kết luận

- Các mạng nơ-ron hồi tiếp có cổng nắm bắt các phụ thuộc xa trong chuỗi thời gian tốt hơn.

- Cổng xóa giúp nắm bắt phụ thuộc ngắn hạn trong chuỗi thời gian.

- Cổng cập nhật giúp nắm bắt các phụ thuộc dài hạn trong chuỗi thời gian.

- Trường hợp đặc biệt khi cổng xóa được kích hoạt, GRU trở thành RNN cơ bản. Chúng cũng có thể bỏ qua các các thành phần trong chuỗi khi cần.

7.6.2. Bộ nhớ Ngắn hạn Dài (LSTM)

Thách thức đối với việc lưu trữ những thông tin dài hạn và bỏ qua đầu vào ngắn hạn trong các mô hình biến tiềm ẩn đã tồn tại trong một thời gian dài. Một trong những phương pháp tiếp cận sớm nhất để giải quyết vấn đề này là LSTM [Hochreiter & Schmidhuber, 1997]. Nó có nhiều tính chất tương tự Nút Hồi tiếp có Cổng (GRU). Điều thú vị là thiết kế của LSTM chỉ phức tạp hơn GRU một chút nhưng đã xuất hiện trước GRU gần hai thập kỷ.

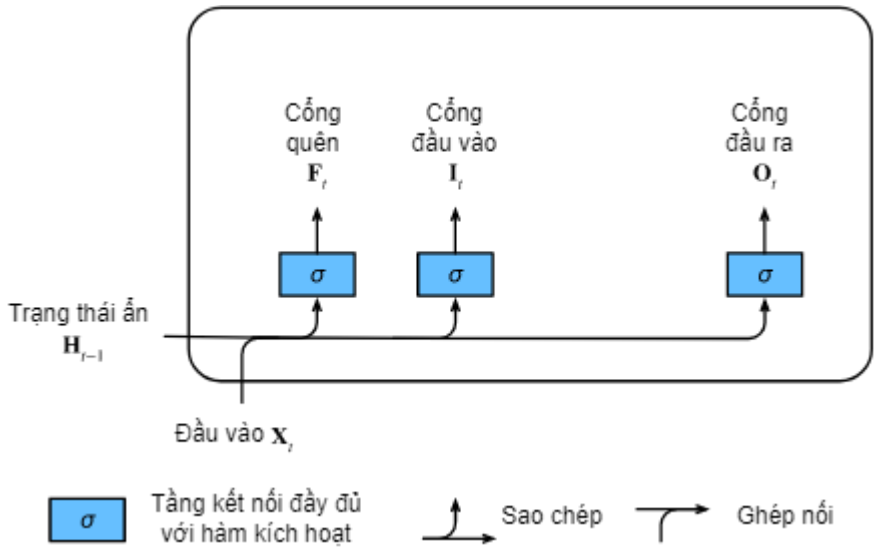
Có thể cho rằng thiết kế này được lấy cảm hứng từ các cổng logic trong máy tính. Để kiểm soát một ô nhớ chúng ta cần một số các cổng. Một cổng để đọc các thông tin từ ô nhớ đó (trái với việc đọc từ các ô khác). Chúng ta sẽ gọi cổng này là cổng đầu ra (output gate). Một cổng thứ hai để quyết định khi nào cần ghi dữ liệu vào ô nhớ. Chúng ta gọi cổng này là cổng đầu vào (input gate). Cuối cùng, chúng ta cần một cơ chế để thiết lập lại nội dung chứa trong ô nhớ, được chi phối bởi một cổng quên (forget gate). Động lực của thiết kế trên cũng tương tự như trước đây, đó là để đưa ra quyết định khi nào cần nhớ và khi nào nên bỏ qua đầu vào trong trạng thái tiềm ẩn thông qua một cơ chế chuyên dụng. Chúng ta hãy xem thiết kế này hoạt động như thế nào trong thực tế.

7.6.2.1. Các Ô nhớ có Cổng

Ba cổng được giới thiệu trong LSTM đó là: cổng đầu vào, cổng quên và cổng đầu ra. Bên cạnh đó chúng ta sẽ giới thiệu một ô nhớ có kích thước giống với trạng thái ẩn. Nói đúng hơn đây chỉ là phiên bản đặc biệt của trạng thái ẩn, được thiết kế để ghi lại các thông tin bổ sung.

7.6.2.1.1. Cổng Đầu vào, Cổng Quên và Cổng Đầu ra

Tương tự như với GRU, dữ liệu được đưa vào các cổng LSTM là đầu vào ở bước thời gian hiện tại $\mathbf{X}_t$ và trạng thái ẩn ở bước thời gian trước đó $\mathbf{H}_{t-1}$. Những đầu vào này được xử lý bởi một tầng kết nối đầy đủ và một hàm kích hoạt sigmoid để tính toán các giá trị của các cổng đầu vào, cổng quên và cổng đầu ra. Kết quả là, tất cả các giá trị đầu ra tại ba cổng đều nằm trong khoảng [0,1]. :numref:lstm_0 minh hoạ luồng dữ liệu cho các cổng đầu vào, cổng quên, và cổng đầu ra.



Hình 7.13. Các phép tính tại cổng đầu vào, cổng quên và cổng đầu ra trong một đơn vị LSTM.

Chúng ta giả sử rằng có h nút ẩn, mỗi minibatch có kích thước n và kích thước đầu vào là d . Như vậy, đầu vào là $\mathbf{X}_t \in \mathbb{R}^{n\times d}$và trạng thái ẩn của bước thời gian trước đó là $\mathbf{H}_{t-1} \in \mathbb{R}^{n\times h}$. Tương tự, các cổng được định nghĩa như sau: cổng đầu vào là $\mathbf{I}_t \in \mathbb{R}^{n\times h}$, cổng quên là $\mathbf{F}_t \in \mathbb{R}^{n\times h}$, và cổng đầu ra là $\mathbf{O}_t \in \mathbb{R}^{n\times h}$. Chúng được tính như sau:

$$\begin{aligned}\mathbf{I}_t &= \sigma(\mathbf{X}_t\mathbf{W}_{xi} + \mathbf{H}_{t-1}\mathbf{W}_{hi} + \mathbf{b}_i),\\ \mathbf{F}_t &= \sigma(\mathbf{X}_t\mathbf{W}_{xf} + \mathbf{H}_{t-1}\mathbf{W}_{hf} + \mathbf{b}_f),\\ \mathbf{O}_t &= \sigma(\mathbf{X}_t\mathbf{W}_{xo} + \mathbf{H}_{t-1}\mathbf{W}_{ho} + \mathbf{b}_o),\end{aligned}$$

trong đó $\mathbf{W}_{xi}, \mathbf{W}_{xf}, \mathbf{W}_{xo} \in \mathbb{R}^{d\times h}$ và $\mathbf{W}_{hi}, \mathbf{W}_{hf}, \mathbf{W}_{ho} \in \mathbb{R}^{h\times h}$ là các trọng số và $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o \in \mathbb{R}^{1\times h}$ là các hệ số điều chỉnh.

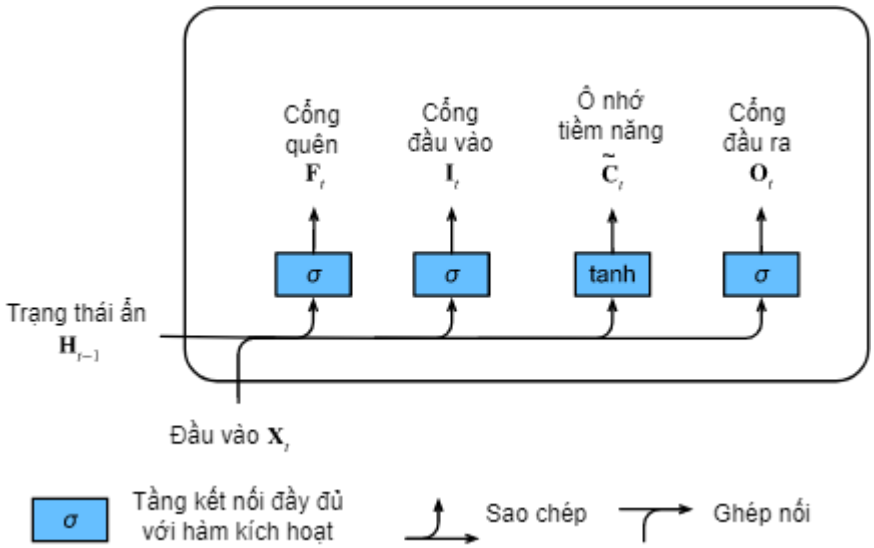
7.6.2.1.2. Ô nhớ Tiềm năng

Tiếp theo, chúng ta sẽ thiết kế một ô nhớ. Vì ta vẫn chưa chỉ định tác động của các cổng khác nhau, nên đầu tiên ta sẽ giới thiệu ô nhớ tiềm năng $\tilde{\mathbf{C}}_t \in \mathbb{R}^{n\times h}$. Các phép tính toán cũng tương tự như ba cổng mô tả ở trên, ngoài trừ việc ở đây ta sử dụng hàm kích hoạt tanh với miền giá trị nằm trong khoảng [−1,1]. Điều này dẫn đến phương trình sau tại bước thời gian t.

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t\mathbf{W}_{xc} + \mathbf{H}_{t-1}\mathbf{W}_{hc} + \mathbf{b}_c),$$

Ở đây $\mathbf{W}_{xc} \in \mathbb{R}^{d\times h}$ và $\mathbf{W}_{hc} \in \mathbb{R}^{h\times h}$ là các tham số trọng số và $\mathbf{b}_c \in \mathbb{R}^{1\times h}$ là một hệ số điều chỉnh.

Ô nhớ tiềm năng được mô tả ngắn gọn trong Hình 7.14.



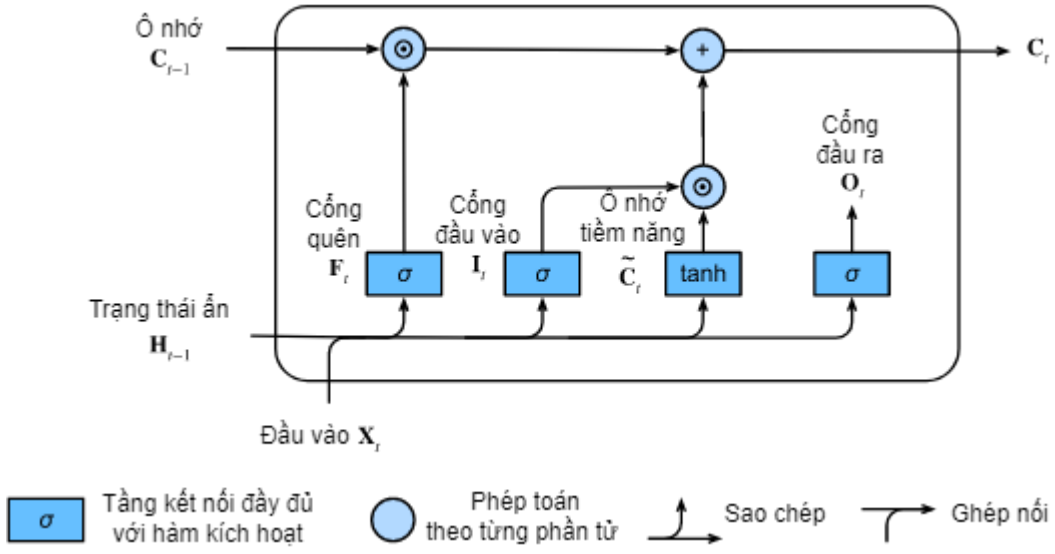
Hình 7.14. Các phép tính toán trong ô nhớ tiềm năng của LSTM.

7.6.2.1.3. Ô nhớ

Trong GRU, chúng ta chỉ có một cơ chế duy nhất để quản lý cả việc nhớ và quên. Trong LSTM, chúng ta có hai tham số, $\mathbf{I}_t$ điều chỉnh lượng dữ liệu mới được lấy vào thông qua $\tilde{\mathbf{C}}_t$ và tham số quên $\mathbf{F}_t$ chỉ định lượng thông tin cũ cần giữ lại trong ô nhớ $\mathbf{C}_{t-1} \in \mathbb{R}^{n \times h}$. Sử dụng cùng một phép nhân theo từng điểm (pointwise) như trước đây, chúng ta đi đến phương trình cập nhật như sau.

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t.$$

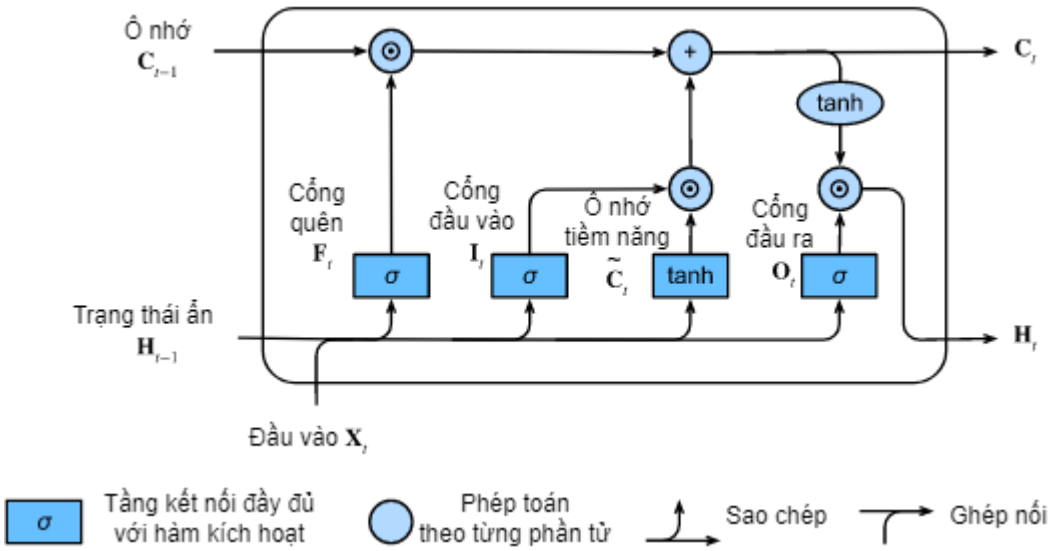
Nếu giá trị ở cổng quên luôn xấp xỉ bằng 1 và cổng đầu vào luôn xấp xỉ bằng 0, thì giá trị ô nhớ trong quá khứ $\mathbf{C}_{t-1}$ sẽ được lưu lại qua thời gian và truyền tới bước thời gian hiện tại. Thiết kế này được giới thiệu nhằm giảm bớt vấn đề tiêu biến gradient cũng như nắm bắt các phụ thuộc dài hạn trong chuỗi thời gian tốt hơn. Do đó chúng ta có sơ đồ luồng trong Hình 7.15.



Hình 7.15. Các phép tính toán trong ô nhớ của LSTM. Ở đây, ta sử dụng phép nhân theo từng phần tử.

7.6.2.1.4. Các Trạng thái Ẩn

Cuối cùng, chúng ta cần phải xác định cách tính trạng thái ẩn $\mathbf{H}_t \in \mathbb{R}^{n \times h}$. Đây là nơi cổng đầu ra được sử dụng. Trong LSTM, đây chỉ đơn giản là một phiên bản có kiểm soát của hàm kích hoạt tanh trong ô nhớ. Điều này đảm bảo rằng các giá trị của $\mathbf{H}_t$ luôn nằm trong khoảng $(-1,1)$. Bất cứ khi nào giá trị của cổng đầu ra là 1, thực chất chúng ta đang đưa toàn bộ thông tin trong ô nhớ tới bộ dự đoán. Ngược lại, khi giá trị của cổng đầu ra là 0, chúng ta giữ lại tất cả các thông tin trong ô nhớ và không xử lý gì thêm. Hình 7.16 minh họa các luồng dữ liệu.



Hình 7.16. Các phép tính của trạng thái ẩn. Phép tính nhân được thực hiện trên từng phần tử.

Trong nhiều trường hợp, các mô hình LSTM hoạt động tốt hơn một chút so với các mô hình GRU nhưng việc huấn luyện và thực thi các mô hình này khá là tốn kém do chúng có kích thước trạng thái tiềm ẩn lớn hơn. LSTM là nguyên mẫu điển hình của một mô hình tự hồi quy biến tiềm ẩn có cơ chế kiểm soát trạng thái phức tạp. Nhiều biến thể đã được đề xuất qua từng năm, ví dụ như các kiến trúc đa tầng, các kết nối phần dư hay các kiểu điều chuẩn khác nhau. Tuy nhiên, việc huấn luyện LSTM và các mô hình chuỗi khác (như GRU) khá là tốn kém do sự phụ thuộc dài hạn của chuỗi. Sau này ta có thể sử dụng các mô hình khác như Transformer để song song hoá việc huấn luyện chuỗi.

7.6.2.4. Kết luận

- LSTM có ba loại cổng để kiểm soát luồng thông tin: cổng đầu vào, cổng quên và cổng đầu ra.

- Đầu ra tầng ẩn của LSTM bao gồm các trạng thái ẩn và các ô nhớ. Chỉ các trạng thái ẩn là được truyền tới tầng đầu ra. Các ô nhớ hoàn toàn được sử dụng nội bộ trong tầng.

- LSTM có thể đối phó với vấn đề tiêu biến và bùng nổ gradient.

7.6.3. Mạng Nơ-ron Hồi tiếp Sâu

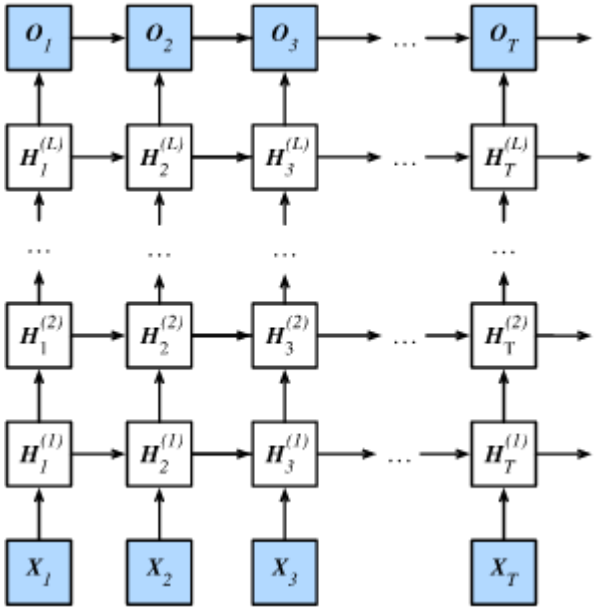
Cho đến nay, chúng ta mới chỉ thảo luận về các mạng nơ-ron hồi tiếp với duy nhất một tầng ẩn đơn hướng. Trong đó, cách các biến tiềm ẩn và các quan sát tương tác với nhau còn khá khá tuỳ ý. Đây không phải là một vấn đề lớn miễn là ta vẫn có đủ độ linh hoạt để mô hình hóa các loại tương tác khác nhau. Tuy nhiên, đây lại là một thách thức với các mạng đơn tầng. Trong trường hợp của perceptron, chúng ta giải quyết vấn đề này bằng cách đưa thêm nhiều tầng vào mạng. Cách này hơi phức tạp một chút với trường hợp của mạng RNN, vì đầu tiên chúng ta cần

phải quyết định thêm tính phi tuyến vào mạng ở đâu và như thế nào. Thảo luận dưới đây tập trung chủ yếu vào LSTM, nhưng cũng có thể áp dụng cho các mô hình chuỗi khác.

Chúng ta có thể bổ sung thêm tính phi tuyến vào các cơ chế cổng. Nghĩa là, thay vì sử dụng một tầng perceptron duy nhất, chúng ta có thể sử dụng nhiều tầng perceptron. Cách này không làm thay đổi cơ chế của mạng LSTM, ngược lại, còn làm cho nó tinh xảo hơn. Điều này chỉ có lợi nếu chúng ta tin rằng cơ chế LSTM biểu diễn một hình thái phổ quát nào đó về cách hoạt động của các mô hình tự hồi quy biến tiềm ẩn.

Chúng ta có thể chồng nhiều tầng LSTM lên nhau. Cách này tạo ra một cơ chế linh hoạt hơn nhờ vào sự kết hợp giữa các tầng đơn giản. Đặc biệt là, các đặc tính liên quan của dữ liệu có thể được biểu diễn ở các tầng khác nhau. Ví dụ, chúng ta có thể muốn lưu dữ liệu về tình hình thị trường tài chính (thị trường giá lên hay giá xuống) ở tầng cao hơn, trong khi đó chỉ ghi lại động lực thời hạn ngắn hơn ở một tầng thấp hơn.

Ngoài những thứ khá trừu tượng trên, để hiểu được các nhóm mô hình chúng ta đang thảo luận một cách dễ dàng nhất, chúng ta nên xem lại Hình 7.17. Hình trên mô tả một mạng nơ-ron hồi tiếp sâu với L tầng ẩn. Mỗi trạng thái ẩn liên tục được truyền tới bước thời gian kế tiếp ở tầng hiện tại và tới bước thời gian hiện tại ở tầng kế tiếp.



Hình 7.17. Kiến trúc của một mạng nơ-ron hồi tiếp sâu.

7.6.3.1. Các Phụ thuộc Hàm

Tại bước thời gian t, giả sử rằng chúng ta có một minibatch $\mathbf{X}_t \in \mathbb{R}^{n\times d}$ (số lượng mẫu: n, số lượng đầu vào: d). Trạng thái ẩn của tầng ẩn ℓ (ℓ=1,…,T) là $\mathbf{H}_t^{(l)} \in \mathbb{R}^{n\times h}$ (số đơn vị ẩn: h), biến tầng đầu ra là $\mathbf{O}_t \in \mathbb{R}^{n\times q}$ (số lượng đầu ra: q) và một hàm kích hoạt tầng ẩn f_l cho tầng l . Chúng ta tính toán trạng thái ẩn của tầng đầu tiên như trước đây, sử dụng đầu vào là $\mathbf{X}_t$. Đối với tất cả các tầng tiếp theo, trạng thái ẩn của tầng trước được sử dụng thay cho $\mathbf{X}_t$.

$$\mathbf{H}_t^{(l)} = f_l\left(\boldsymbol{X_t}, \mathbf{H}_{t-1}^{(1)}\right),$$

$$\mathbf{H}_t^{(l)} = f_l\left(\mathbf{H}_t^{(l-1)}, \mathbf{H}_{t-1}^{(l)}\right),$$

Cuối cùng, tầng đầu ra chỉ dựa trên trạng thái ẩn của tầng ẩn **L**. Chúng ta sử dụng một hàm đầu ra g để xử lý trạng thái này:

$$\mathbf{O}_t = g\left(\mathbf{H}_t^{(L)}\right)$$

Giống như perceptron đa tầng, số tầng ẩn **L** và số đơn vị ẩn h được coi là các siêu tham số. Đặc biệt, chúng ta có thể chọn một trong các kiến trúc RNN, GRU, hoặc LSTM thông thường để xây dựng mô hình.

7.6.3.4. Kết luận

- Trong các mạng nơ-ron hồi tiếp sâu, thông tin trạng thái ẩn được truyền tới bước thời gian kế tiếp ở tầng hiện tại và truyền tới bước thời gian hiện tại ở tầng kế tiếp.

- Có nhiều phiên bản khác nhau của mạng RNN sâu, ví dụ như LSTM, GRU hoặc RNN thông thường. Những mô hình này được lập trình sẵn trong mô-đun rnn của Gluon.

- Chúng ta cần phải cẩn thận trong việc khởi tạo mô hình. Nhìn chung, các mạng RNN sâu thường đòi hỏi khá nhiều công sức (ví dụ như việc chọn tốc độ học hay việc gọt gradient) để đảm bảo quá trình học hội tụ một cách hợp lý.

Tài liệu tham khảo

[Ahmed et al., 2012] Ahmed, A., Aly, M., Gonzalez, J., Narayanamurthy, S., & Smola, A. J. (2012). Scalable inference in latent variable models. Proceedings of the fifth ACM international conference on Web search and data mining (pp. 123–132).

[Aji & McEliece, 2000] Aji, S. M., & McEliece, R. J. (2000). The generalized distributive law. IEEE transactions on Information Theory, 46(2), 325–343.

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

[Bishop, 1995] Bishop, C. M. (1995). Training with noise is equivalent to tikhonov regularization. Neural computation, 7(1), 108–116.

[Bishop, 2006] Bishop, C. M. (2006). Pattern recognition and machine learning. springer.

[Bojanowski et al., 2017] Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 5, 135–146.

[Bollobas, 1999] Bollobás, B. (1999). Linear analysis. Cambridge University Press, Cambridge.

[Bowman et al., 2015]Bowman, S. R., Angeli, G., Potts, C., & Manning, C. D. (2015). A large annotated corpus for learning natural language inference. arXiv preprint arXiv:1508.05326.

[Boyd & Vandenberghe, 2004] Boyd, S., & Vandenberghe, L. (2004). Convex Optimization. Cambridge, England: Cambridge University Press.

[Brown & Sandholm, 2017] Brown, N., & Sandholm, T. (2017). Libratus: the superhuman ai for no-limit poker. IJCAI (pp. 5226–5228).

[Campbell et al., 2002] Campbell, M., Hoane Jr, A. J., & Hsu, F.-h. (2002). Deep blue. Artificial intelligence, 134(1-2), 57–83.

[Cer et al., 2017] Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., & Specia, L. (2017). Semeval-2017 task 1: semantic textual similarity multilingual and crosslingual focused evaluation. Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017) (pp. 1–14).

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: encoder-decoder approaches. arXiv preprint arXiv:1409.1259.

[Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

[Csiszar, 2008] Csiszár, I. (2008). Axiomatic characterizations of information measures. Entropy, 10(3), 261–273.

[DeCock, 2011] De Cock, D. (2011). Ames, iowa: alternative to the boston housing data as an end of semester regression project. Journal of Statistics Education, 19(3).

[DeCandia et al., 2007] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., … Vogels, W. (2007). Dynamo: amazon's highly available key-value store. ACM SIGOPS operating systems review (pp. 205–220).

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[Doucet et al., 2001] Doucet, A., De Freitas, N., & Gordon, N. (2001). An introduction to sequential monte carlo methods. Sequential Monte Carlo methods in practice (pp. 3–14). Springer.

[Duchi et al., 2011] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul), 2121–2159.

[Dumoulin & Visin, 2016] Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285.

[Edelman et al., 2007] Edelman, B., Ostrovsky, M., & Schwarz, M. (2007). Internet advertising and the generalized second-price auction: selling billions of dollars worth of keywords. American economic review, 97(1), 242–259.

[Flammarion & Bach, 2015] Flammarion, N., & Bach, F. (2015). From averaging to acceleration, there is only a step-size. Conference on Learning Theory (pp. 658–695).

[Gatys et al., 2016] Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). Image style transfer using convolutional neural networks. Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2414–2423).

[Ginibre, 1965] Ginibre, J. (1965). Statistical ensembles of complex, quaternion, and real matrices. Journal of Mathematical Physics, 6(3), 440–449.

[Girshick, 2015] Girshick, R. (2015). Fast r-cnn. Proceedings of the IEEE international conference on computer vision (pp. 1440–1448).

[Girshick et al., 2014] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580–587).

[Glorot & Bengio, 2010] Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249–256).

[Goh, 2017] Goh, G. (2017). Why momentum really works. Distill. URL: http://distill.pub/2017/momentum, doi:10.23915/distill.00006

[Goldberg et al., 1992] Goldberg, D., Nichols, D., Oki, B. M., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. Communications of the ACM, 35(12), 61–71.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. http://www.deeplearningbook.org.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., … Bengio, Y. (2014). Generative adversarial nets. Advances in neural information processing systems (pp. 2672–2680).

[Gotmare et al., 2018] Gotmare, A., Keskar, N. S., Xiong, C., & Socher, R. (2018). A closer look at deep learning heuristics: learning rate restarts, warmup and distillation. arXiv preprint arXiv:1810.13243.

[Graves & Schmidhuber, 2005] Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. Neural networks, 18(5-6), 602–610.

[Gunawardana & Shani, 2015] Gunawardana, A., & Shani, G. (2015). Evaluating recommender systems. Recommender systems handbook (pp. 265–308). Springer.

[Guo et al., 2017] Guo, H., Tang, R., Ye, Y., Li, Z., & He, X. (2017). Deepfm: a factorization-machine based neural network for ctr prediction. Proceedings of the 26th International Joint Conference on Artificial Intelligence (pp. 1725–1731).

[He et al., 2017a] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. Proceedings of the IEEE international conference on computer vision (pp. 2961–2969).

[He et al., 2016a] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770–778).

[He et al., 2016b] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Identity mappings in deep residual networks. European conference on computer vision (pp. 630–645).

[He & Chua, 2017] He, X., & Chua, T.-S. (2017). Neural factorization machines for sparse predictive analytics. Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval (pp. 355–364).

[He et al., 2017b] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. Proceedings of the 26th international conference on world wide web (pp. 173–182).

[Hebb & Hebb, 1949] Hebb, D. O., & Hebb, D. (1949). The organization of behavior. Vol. 65. Wiley New York.

[Hendrycks & Gimpel, 2016] Hendrycks, D., & Gimpel, K. (2016). Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415.

[Hennessy & Patterson, 2011] Hennessy, J. L., & Patterson, D. A. (2011). Computer architecture: a quantitative approach. Elsevier.

[Herlocker et al., 1999] Herlocker, J. L., Konstan, J. A., Borchers, A., & Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1999 (pp. 230–237).

[Hidasi et al., 2015] Hidasi, B., Karatzoglou, A., Baltrunas, L., & Tikk, D. (2015). Session-based recommendations with recurrent neural networks. arXiv preprint arXiv:1511.06939.

[Hochreiter & Schmidhuber, 1997] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735–1780.

[Hoyer et al., 2009] Hoyer, P. O., Janzing, D., Mooij, J. M., Peters, J., & Schölkopf, B. (2009). Nonlinear causal discovery with additive noise models. Advances in neural information processing systems (pp. 689–696).

[Hu et al., 2018] Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks. Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7132–7141).

[Hu et al., 2008] Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. 2008 Eighth IEEE International Conference on Data Mining (pp. 263–272).

[Huang et al., 2017] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700–4708).

[Ioffe, 2017] Ioffe, S. (2017). Batch renormalization: towards reducing minibatch dependence in batch-normalized models. Advances in neural information processing systems (pp. 1945–1953).

[Ioffe & Szegedy, 2015] Ioffe, S., & Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

[Izmailov et al., 2018] Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., & Wilson, A. G. (2018). Averaging weights leads to wider optima and better generalization. arXiv preprint arXiv:1803.05407.

[Jia et al., 2018] Jia, X., Song, S., He, W., Wang, Y., Rong, H., Zhou, F., … others. (2018). Highly scalable deep learning training system with mixed-precision: training imagenet in four minutes. arXiv preprint arXiv:1807.11205.

[Jouppi et al., 2017] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., … others. (2017). In-datacenter performance analysis of a tensor processing unit. 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA) (pp. 1–12).

[Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882.

[Kingma & Ba, 2014] Kingma, D. P., & Ba, J. (2014). Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[Koller & Friedman, 2009] Koller, D., & Friedman, N. (2009). Probabilistic graphical models: principles and techniques. MIT press.

[Kolter, 2008] Kolter, Z. (2008). Linear algebra review and reference. Available online: http.

[Koren, 2009] Koren, Y. (2009). Collaborative filtering with temporal dynamics. Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 447–456).

[Koren et al., 2009] Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, pp. 30–37.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems (pp. 1097–1105).

[Kung, 1988] Kung, S. Y. (1988). Vlsi array processors. Englewood Cliffs, NJ, Prentice Hall, 1988, 685 p. Research supported by the Semiconductor Research Corp., SDIO, NSF, and US Navy.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., & others. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324.

[Li, 2017] Li, M. (2017). Scaling Distributed Machine Learning with System and Algorithm Co-design (Doctoral dissertation). PhD Thesis, CMU.

[Li et al., 2014] Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., … Su, B.-Y. (2014). Scaling distributed machine learning with the parameter server. 11th $\$USENIX$\$ Symposium on Operating Systems Design and Implementation ($\$OSDI$\$ 14) (pp. 583–598).

[Lin et al., 2013] Lin, M., Chen, Q., & Yan, S. (2013). Network in network. arXiv preprint arXiv:1312.4400.

[Lin et al., 2017] Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. Proceedings of the IEEE international conference on computer vision (pp. 2980–2988).

[Lin et al., 2010] Lin, Y., Lv, F., Zhu, S., Yang, M., Cour, T., Yu, K., … others. (2010). Imagenet classification: fast descriptor coding and large-scale svm training. Large scale visual recognition challenge.

[Lipton & Steinhardt, 2018] Lipton, Z. C., & Steinhardt, J. (2018). Troubling trends in machine learning scholarship. arXiv preprint arXiv:1807.03341.

[Liu et al., 2016] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). Ssd: single shot multibox detector. European conference on computer vision (pp. 21–37).

[Liu et al., 2019] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., … Stoyanov, V. (2019). Roberta: a robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

[Long et al., 2015] Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3431–3440).

[Loshchilov & Hutter, 2016] Loshchilov, I., & Hutter, F. (2016). Sgdr: stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983.

[Luo et al., 2018] Luo, P., Wang, X., Shao, W., & Peng, Z. (2018). Towards understanding regularization in batch normalization. arXiv preprint.

[Maas et al., 2011] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning word vectors for sentiment analysis. Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1 (pp. 142–150).

[McCann et al., 2017] McCann, B., Bradbury, J., Xiong, C., & Socher, R. (2017). Learned in translation: contextualized word vectors. Advances in Neural Information Processing Systems (pp. 6294–6305).

[McCulloch & Pitts, 1943] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115–133.

[McMahan et al., 2013] McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., … others. (2013). Ad click prediction: a view from the trenches. Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 1222–1230).

[Merity et al., 2016] Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2016). Pointer sentinel mixture models. arXiv preprint arXiv:1609.07843.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems (pp. 3111–3119).

[Mirhoseini et al., 2017] Mirhoseini, A., Pham, H., Le, Q. V., Steiner, B., Larsen, R., Zhou, Y., … Dean, J. (2017). Device placement optimization with reinforcement learning. Proceedings of the 34th International Conference on Machine Learning-Volume 70 (pp. 2430–2439).

[Morey et al., 2016] Morey, R. D., Hoekstra, R., Rouder, J. N., Lee, M. D., & Wagenmakers, E.-J. (2016). The fallacy of placing confidence in confidence intervals. Psychonomic bulletin & review, 23(1), 103–123.

[Nesterov & Vial, 2000] Nesterov, Y., & Vial, J.-P. (2000). Confidence level solutions for stochastic programming, Stochastic Programming E-Print Series.

[Nesterov, 2018] Nesterov, Y. (2018). Lectures on convex optimization. Vol. 137. Springer.

[Neyman, 1937] Neyman, J. (1937). Outline of a theory of statistical estimation based on the classical theory of probability. Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences, 236(767), 333–380.

[Parikh et al., 2016] Parikh, A. P., Täckström, O., Das, D., & Uszkoreit, J. (2016). A decomposable attention model for natural language inference. arXiv preprint arXiv:1606.01933.

[Pennington et al., 2017] Pennington, J., Schoenholz, S., & Ganguli, S. (2017). Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. Advances in neural information processing systems (pp. 4785–4795).

[Pennington et al., 2014] Pennington, J., Socher, R., & Manning, C. (2014). Glove: global vectors for word representation. Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532–1543).

[Peters et al., 2017a] Peters, J., Janzing, D., & Schölkopf, B. (2017). Elements of causal inference: foundations and learning algorithms. MIT press.

[Peters et al., 2017b] Peters, M., Ammar, W., Bhagavatula, C., & Power, R. (2017). Semi-supervised sequence tagging with bidirectional language models. Proceedings of the 55th Annual

Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 1756–1765).

[Peters et al., 2018] Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers) (pp. 2227–2237).

[Petersen et al., 2008] Petersen, K. B., Pedersen, M. S., & others. (2008). The matrix cookbook. Technical University of Denmark, 7(15), 510.

[Polyak, 1964]Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 4(5), 1–17.

[Quadrana et al., 2018] Quadrana, M., Cremonesi, P., & Jannach, D. (2018). Sequence-aware recommender systems. ACM Computing Surveys (CSUR), 51(4), 66.

[Radford et al., 2015] Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

[Radford et al., 2018] Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. OpenAI.

[Radford et al., 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI Blog, 1(8), 9.

[Rajpurkar et al., 2016] Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250.

[Reddi et al., 2019] Reddi, S. J., Kale, S., & Kumar, S. (2019). On the convergence of adam and beyond. arXiv preprint arXiv:1904.09237.

[Reed & DeFreitas, 2015] Reed, S., & De Freitas, N. (2015). Neural programmer-interpreters. arXiv preprint arXiv:1511.06279.

[Ren et al., 2015] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: towards real-time object detection with region proposal networks. Advances in neural information processing systems (pp. 91–99).

[Rendle, 2010]Rendle, S. (2010). Factorization machines. 2010 IEEE International Conference on Data Mining (pp. 995–1000).

[Rendle et al., 2009] Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2009). Bpr: bayesian personalized ranking from implicit feedback. Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence (pp. 452–461).

[Rumelhart et al., 1988] Rumelhart, D. E., Hinton, G. E., Williams, R. J., & others. (1988). Learning representations by back-propagating errors. Cognitive modeling, 5(3), 1.

[Russell & Norvig, 2016] Russell, S. J., & Norvig, P. (2016). Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited,.

[Santurkar et al., 2018] Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization? Advances in Neural Information Processing Systems (pp. 2483–2493).

[Sarwar et al., 2001] Sarwar, B. M., Karypis, G., Konstan, J. A., Riedl, J., & others. (2001). Item-based collaborative filtering recommendation algorithms. Www, 1, 285–295.

[Schein et al., 2002] Schein, A. I., Popescul, A., Ungar, L. H., & Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval (pp. 253–260).

[Schuster & Paliwal, 1997] Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11), 2673–2681.

[Sedhain et al., 2015] Sedhain, S., Menon, A. K., Sanner, S., & Xie, L. (2015). Autorec: autoencoders meet collaborative filtering. Proceedings of the 24th International Conference on World Wide Web (pp. 111–112).

[Sennrich et al., 2015] Sennrich, R., Haddow, B., & Birch, A. (2015). Neural machine translation of rare words with subword units. arXiv preprint arXiv:1508.07909.

[Sergeev & DelBalso, 2018] Sergeev, A., & Del Balso, M. (2018). Horovod: fast and easy distributed deep learning in tensorflow. arXiv preprint arXiv:1802.05799.

[Shannon, 1948] Shannon, C. E. (1948 , 7). A mathematical theory of communication. The Bell System Technical Journal, 27(3), 379–423.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., … others. (2016). Mastering the game of go with deep neural networks and tree search. nature, 529(7587), 484.

[Simonyan & Zisserman, 2014] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[Smola & Narayanamurthy, 2010] Smola, A., & Narayanamurthy, S. (2010). An architecture for parallel topic models. Proceedings of the VLDB Endowment, 3(1-2), 703–710.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929–1958.

[Strang, 1993] Strang, G. (1993). Introduction to linear algebra. Vol. 3. Wellesley-Cambridge Press Wellesley, MA.

[Su & Khoshgoftaar, 2009] Su, X., & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. Advances in artificial intelligence, 2009.

[Sukhbaatar et al., 2015] Sukhbaatar, S., Weston, J., Fergus, R., & others. (2015). End-to-end memory networks. Advances in neural information processing systems (pp. 2440–2448).

[Sutskever et al., 2013] Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. International conference on machine learning (pp. 1139–1147).

[Szegedy et al., 2017] Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. Thirty-First AAAI Conference on Artificial Intelligence.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2015). Going deeper with convolutions. Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1–9).

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818–2826).

[Tallec & Ollivier, 2017] Tallec, C., & Ollivier, Y. (2017). Unbiasing truncated backpropagation through time. arXiv preprint arXiv:1705.08209.

[Tang & Wang, 2018] Tang, J., & Wang, K. (2018). Personalized top-n sequential recommendation via convolutional sequence embedding. Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (pp. 565–573).

[Teye et al., 2018] Teye, M., Azizpour, H., & Smith, K. (2018). Bayesian uncertainty estimation for batch normalized deep networks. arXiv preprint arXiv:1802.06455.

[Tieleman & Hinton, 2012] Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), 26–31.

[Treisman & Gelade, 1980] Treisman, A. M., & Gelade, G. (1980). A feature-integration theory of attention. Cognitive psychology, 12(1), 97–136.

[Toscher et al., 2009] Töscher, A., Jahrer, M., & Bell, R. M. (2009). The bigchaos solution to the netflix grand prize. Netflix prize documentation, pp. 1–52.

[Uijlings et al., 2013] Uijlings, J. R., Van De Sande, K. E., Gevers, T., & Smeulders, A. W. (2013). Selective search for object recognition. International journal of computer vision, 104(2), 154–171.

[VanLoan & Golub, 1983] Van Loan, C. F., & Golub, G. H. (1983). Matrix computations. Johns Hopkins University Press.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems (pp. 5998–6008).

[Wang et al., 2018] Wang, L., Li, M., Liberty, E., & Smola, A. J. (2018). Optimal message scheduling for aggregation. NETWORKS, 2(3), 2–3.

[Wang et al., 2016] Wang, Y., Davidson, A., Pan, Y., Wu, Y., Riffel, A., & Owens, J. D. (2016). Gunrock: a high-performance graph processing library on the gpu. ACM SIGPLAN Notices (p. 11).

[Warstadt et al., 2019] Warstadt, A., Singh, A., & Bowman, S. R. (2019). Neural network acceptability judgments. Transactions of the Association for Computational Linguistics, 7, 625–641.

[Wasserman, 2013] Wasserman, L. (2013). All of statistics: a concise course in statistical inference. Springer Science & Business Media.

[Watkins & Dayan, 1992] Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279–292.

[Welling & Teh, 2011] Welling, M., & Teh, Y. W. (2011). Bayesian learning via stochastic gradient langevin dynamics. Proceedings of the 28th international conference on machine learning (ICML-11) (pp. 681–688).

[Wigner, 1958] Wigner, E. P. (1958). On the distribution of the roots of certain symmetric matrices. Ann. Math (pp. 325–327).

[Wood et al., 2011] Wood, F., Gasthaus, J., Archambeau, C., James, L., & Teh, Y. W. (2011). The sequence memoizer. Communications of the ACM, 54(2), 91–98.

[Wu et al., 2017] Wu, C.-Y., Ahmed, A., Beutel, A., Smola, A. J., & Jing, H. (2017). Recurrent recommender networks. Proceedings of the tenth ACM international conference on web search and data mining (pp. 495–503).

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., … others. (2016). Google's neural machine translation system: bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144.

[Xiao et al., 2017] Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.

[Xiong et al., 2018] Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X., & Stolcke, A. (2018). The microsoft 2017 conversational speech recognition system. 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 5934–5938).

[Ye et al., 2011] Ye, M., Yin, P., Lee, W.-C., & Lee, D.-L. (2011). Exploiting geographical influence for collaborative point-of-interest recommendation. Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval (pp. 325–334).

[You et al., 2017] You, Y., Gitman, I., & Ginsburg, B. (2017). Large batch training of convolutional networks. arXiv preprint arXiv:1708.03888.

[Zaheer et al., 2018] Zaheer, M., Reddi, S., Sachan, D., Kale, S., & Kumar, S. (2018). Adaptive methods for nonconvex optimization. Advances in Neural Information Processing Systems (pp. 9793–9803).

[Zeiler, 2012] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

[Zhang et al., 2019] Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep learning based recommender system: a survey and new perspectives. ACM Computing Surveys (CSUR), 52(1), 5.

[Zhu et al., 2017] Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. Proceedings of the IEEE international conference on computer vision (pp. 2223–2232).

[Zhu et al., 2015] Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Aligning books and movies: towards story-like visual explanations by watching movies and reading books. Proceedings of the IEEE international conference on computer vision (pp. 19–27).