fadings patterns shadows.blur shapes decorations.pathreplacing vietnamese

vietnamese **ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH**
TRƯỜNG ĐẠI HỌC BÁCH KHOA
Quản Thành Thơ

MẠNG NƠ-RON NHÂN TẠO: TỪ HỒI QUY ĐẾN HỌC SÂU

TP HỒ CHÍ MINH - 2020
NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA

To laoshi, Bao Bao and Dong Dong

Lời nói đầu

Với sự phát triển mạnh mẽ về tài nguyên tính toán từ những năm cuối thế kỉ 20, kỹ thuật *học sâu* đã tìm được cho mình năng lượng để đóng góp vào mọi mặt của cuộc sống. Khởi nguồn của sự thành công đó là sự ra đời của *mạng nơ-ron nhân tạo*. Lấy ý tưởng từ cách nơ-ron hoạt động trong bộ não, mạng nơ-ron đã được thiết kế để mô hình hóa những đặc điểm về não bộ, đồng thời tối ưu cho việc giải quyết các vấn đề trong cuộc sống.

Đã có nhiều hội thảo khoa học liên quan đến học sâu được tổ chức, thu hút nhiều công bố và thảo luận có giá trị khoa học cao. Tuy nhiên, bên cạnh những thành tựu không thể phủ nhận của học sâu như là một đóng góp quan trọng để đưa các ứng dụng *Trí Tuệ Nhân Tạo* tiến vào một kỷ nguyên mới, cũng có nhiều tranh luận về sự đóng góp của kỹ thuật này vào ngành *Khoa Học Máy Tính*. Nhiều ý kiến cho rằng cơ chế hoạt động của một mạng học sâu vẫn chủ yếu dựa vào các kỹ thuật *gradient descend* và *back propagation*, vốn là các kỹ thuật nền tảng của một mạng nơ-ron. Như vậy mạng học sâu cũng chỉ là một mạng nơ-ron đa tầng và vì vậy tính mới về học thuật của kỹ thuật này là rất hạn chế.

Sách *"Mạng nơ-ron nhân tạo: từ hồi quy đến học sâu"* này được viết ra với mục đích góp phần làm sáng tỏ các luận điểm trên. Nội dung chính của quyển sách này sẽ xoay quanh chủ đề kiến trúc và các loại mạng nơ-ron khác nhau. Khởi đầu sách là sự giới thiệu về mạng nơ-ron, cơ chế hoạt động của một nút nơ-ron dưới góc nhìn hồi quy logistic cho đến mạng nơ-ron đa tầng. Tiếp đó, các kiến trúc mạng học sâu phổ biến bao gồm *mạng nơ-ron tích chập* (Convolutional Neural Networks - CNN) và *mạng nơ-ron truy hồi* (Recurrent Neural Network - RNN) được trình bày. Từ đó, người đọc sẽ thấy được sự phát triển của các mạng học sâu từ các mạng nơ-ron đa tầng truyền thống, đồng thời thấy được các đặc trưng riêng của các mạng học sâu chuyên dụng này. Đặc biệt, sách cũng trình bày sự ứng dụng của học sâu trong hai bài toán kinh điển của lĩnh vực *Xử Lý Ngôn Ngữ Tự Nhiên* (Natural Language Processing - NLP) bao gồm mô hình *nhúng từ* (Word Embedding) và xây dựng *mô hình ngôn ngữ* (Language Modelling)

Các chủ đề này sẽ được trải dài trong 7 chương. Ở mỗi chương sẽ có các bài tập củng cố kiến thức sau mỗi chương. Sách có thể dùng để giảng dạy cho sinh viên ở bậc đại

học và học viên sau đại học trong các lĩnh vực liên quan đến *Học Máy* (Machine Learning) và *Xử Lý Ngôn Ngữ Tự Nhiên*. Trong quá trình hoàn thiện quyển sách, tác giả đã nhận được sự giúp đỡ của rất nhiều đồng nghiệp cũng như Ban Chủ nhiệm Khoa Khoa học và Kỹ thuật Máy tính, Trường Đại học Bách Khoa - ĐHQG-HCM. Xin chân thành gửi lời cảm ơn đến các đồng nghiệp trong Khoa và Ban Chủ nhiệm khoa. Đặc biệt, xin cám ơn các bạn học viên của lớp cao học Hệ Thống Thông Minh CK_HK192 của Trường Đại học Bách Khoa - ĐHQG-HCM (danh sách đầy đủ ở Phụ lục B), các bạn sinh viên Nguyễn Thành Thông và Nguyễn Trần Công Duy, anh Nguyễn Xuân Mão và bạn Băng Ngọc Bảo Tâm đã đóng góp nhiều công sức và ý kiến quý báu trong quá trình hoàn thiện nội dung sách. Tác giả cũng xin chân thành gửi lời cảm ơn đến các tác giả của các tài liệu tham khảo đã cung cấp những thông tin quý báu giúp hoàn thành quyển sách này.

Tóm tắt

Sách *"Mạng nơ-ron nhân tạo: từ hồi quy đến học sâu"* bao gồm 7 chương, mỗi chương gồm phần giới thiệu (hay dẫn dắt) cho đến những kiến thức chung và các ví dụ, kết chương. Kết thúc mỗi chương sẽ là phần bài tập luyện tập thêm, làm rõ các nội dung trình bày trong chương.

Chương **1**: **Giới thiệu về mạng nơ-ron nhân tạo**. Chương này tập trung chủ yếu vào việc giới thiệu về mạng nơ-ron nhân tạo, nền tảng cho học sâu nói chung. Chương này sẽ bao gồm lịch sử mạng nơ-ron, giới thiệu sâu hơn vào thành phần của một mạng nơ-ron, cách thức hoạt động và mạng nơ-ron đa tầng. Chương này cũng đưa ra một ứng dụng minh họa cho bài toán phân loại dựa trên mạng nơ-ron.

Chương **2**: **Logistic Regression**. Chương này tập trung chủ yếu vào mô hình hồi quy logistic cho bài toán phân loại. Nội dung chương xoay quanh các bước của tính toán và huấn luyện bao gồm hàm mất mát, kỹ thuật gradient descent, đạo hàm và lan truyền ngược (back propagation).

Chương **3**: **Multi Layer Perceptron và Deep Learning**. Tiếp nối nội dung của chương 2, trong chương này sẽ khảo sát từ mạng nơ-ron đơn giản đến mạng nơ-ron đa tầng, là nền tảng của mạng học sâu. Bên cạnh đó, nội dung chương cũng sẽ giới thiệu các hàm kích hoạt, giải thích thêm về việc huấn luyện cho mạng nơ-ron phức tạp và việc biểu diễn cấu trúc cùng quá trình hoạt động của một mạng nơ-ron dưới dạng ma trận và các toán tử trên ma trận.

Chương **4**: **Mô hình xử lý ngôn ngữ tự nhiên bằng mạng học sâu**. Chương này sẽ giới thiệu một số khái niệm cơ bản của bài toán xử lý ngôn ngữ tự nhiên như phân loại văn bản cùng các bài toán cổ điển khác trong lĩnh vực này cùng các phương pháp trích xuất đặc trưng từ văn bản. Đi sâu hơn sẽ là mô hình xử lý ngôn ngữ tự nhiên bằng học sâu như biểu diễn từ bằng mô hình nhúng từ , được trình bày bao gồm cả lý thuyết và những ví dụ chi tiết.

Chương **5**: **Mạng nơ-ron Tích chập (Convolutional Neural Network)**. Ở chương này, sách sẽ giới thiệu ứng dụng của mạng nơ-ron tích chập, vốn đã rất thành công trong bài toán xử lý ảnh, vào bài toán ngôn ngữ. Nội dung chương sẽ đưa ra các lý thuyết cơ bản cùng ví dụ trực quan cho mạng nơ-ron tích chập trên

bài toán xử lý ngôn ngữ tự nhiên.

Chương **6**: **Mạng nơ-ron hồi quy - Recurrent Neural Network**. Chương này sẽ thảo luận một mạng học sâu rất quan trọng trong các bài toán xử lý ngôn ngữ tự nhiên hay các bài toán dạng mạng nơ-ron truy hồi. Chương sẽ đưa ra các lý thuyết cơ bản cùng ví dụ trực quan cho mạng nơ-ron truy hồi trên bài toán xử lý ngôn ngữ tự nhiên cùng với các ưu nhược điểm của nó.

Chương **7**: **Language modeling**. Ở chương cuối, sách sẽ trình bày về mô hình ngôn ngữ, một dạng ứng dụng phổ biến của mạng nơ-ron truy hồi cho các bài toán xử lý ngôn ngữ tự nhiên. Chương sẽ thảo luận các kỹ thuật và kiến trúc mô hình ngôn ngữ dựa trên mạng nơ-ron truy hồi. Ứng dụng của mô hình ngôn ngữ vào hai bài toán thường gặp là sinh chuỗi và sửa lỗi chính tả cũng sẽ được thảo luận trong chương này.

Mục lục

Danh sách hình vẽ

Danh sách bảng

Chương 1

Giới thiệu về mạng nơ-ron nhân tạo

1.1 Lịch sử mạng nơ-ron

Bước đầu tiên đối với các *mạng nơ-ron* (neural network) đến vào năm 1943 khi Warren McCulloch, một nhà sinh lý học thần kinh và một nhà toán học trẻ, Walter Pitts, đã phát triển các mô hình đầu tiên của mạng nơ-ron. Họ đã viết bài báo *"Tính toán logic của các ý tưởng trong hoạt động thần kinh"* (The Logical Calculus of the Ideas Immanent in Nervous Activity) về cách các nơ-ron có thể hoạt động (McCulloch and Pitts, 1943). Họ đã mô hình hóa một mạng nơ-ron đơn giản với các *mạch điện* (electrical circuits).

Vào năm 1949, Donald Hebb, một nhà tâm lý học, đã củng cố khái niệm về nơ-ron trong cuốn sách của ông *"Sự tổ chức của hành vi"* (The Organization of Behavior) (Hebb, 1949), một công trình chỉ ra rằng các *lối mòn thần kinh* (neural pathways) được gia cố mỗi khi chúng được sử dụng.

Vào năm 1958, Frank Rosenblatt, một nhà tâm lý học, đã tiến hành một công trình đầu tiên về *perceptron* (Rosenblatt, 1958). Perceptron là một thiết bị điện tử được chế tạo theo nguyên tắc sinh học và có khả năng học hỏi. Ông cũng đã viết một cuốn sách trước đó về điện toán thần kinh, (Rosenblatt, 1959). Một hệ thống khác là ADALINE (ADAptive LInear Element) được phát triển vào năm 1960 bởi hai

kỹ sư điện Bernard Widrow and Marcian Hoff (Widrow and Hoff, 1960). Phương pháp được sử dụng cho việc học trong hệ thống khác này với phương pháp của perceptron, nó sử dụng một thuật toán gọi là *trung bình bình phương nhỏ nhất* (least mean square filter - LMS). Máy ADALINE của Widrow và Hoff không chỉ là "đồ chơi" mà được dùng thực sự trong các sản phẩm công nghiệp cho đến ngày nay, ví dụ như để lọc đi tiếng ồn trong điện thoại.

Vào năm 1969, Marvin Minsky và Seymour Papert xuất bản một quyển sách về perceptron có tên "Perceptrons: An Introduction to Computational Geometry", trong đó đưa ra nhiều phê phán, và nói rằng những người nghiên cứu perceptron đã quá lạc quan, vẽ ra một viễn tưởng về perceptron mà họ không thể thực hiện được. Minsky và Papert chỉ ra rằng máy perceptron của Rosenblatt thậm chí không mô phỏng được một số hàm logic cơ bản như là XOR.

Bản thân Minsky và Papert biết rằng, nếu dùng *mạng thần kinh nhân tạo* (artificial neural networks – ANN) với từ hai lớp nơ-ron trở lên (là điều mà các ANN ngày nay có, càng nhiều lớp thì được coi là càng sâu), thay vì chỉ có một lớp nơ-ron như perceptron, thì giải quyết được vấn đề mô phỏng các hàm logic mà ANN một lớp không mô phỏng được. Tuy nhiên, từ khi quyển sách của Minsky và Papert xuất hiện, việc nghiên cứu ANN bị chững lại trong vòng cả chục năm, cùng với "mùa đông" của toàn bộ lĩnh vực *trí tuệ nhân tạo* (Artificial Intelligence), ít ai còn dám dùng cụm từ "neural network" trong thời gian này. Trong những năm 1970, người ta vẫn dè dặt nghiên cứu ANN, nhưng đội lốt dưới các tên gọi khác, như là *xử lý tín hiện thích nghi* (adaptive signal processing), *nhận dạng mẫu* (pattern recognition), *mô hình sinh học* (biological modeling).

Cho đến thập kỉ 1980, có nhiều sự kiện diễn ra gây nên sự quan tâm mới trong lĩnh vực này. Vào năm 1982, John Hopfield đã trình bày một bài báo có tên "Neural Networks and Physical Systems with Emergent Collective Computational Abilities" (Hopfield, 1982). Hopfield mô tả mạng thần kinh nhân tạo tái phát phục vụ như *hệ thống nội dung bộ nhớ địa chỉ* (content-addressable memory system). Các công trình của ông đã thuyết phục hàng trăm nhà khoa học, nhà toán học và nhà công nghệ có trình độ cao tham gia vào lĩnh vực mới nổi của mạng nơ-ron.

Đến năm 1985, *Viện Vật lý Hoa Kỳ* (the American Institute of Physics) đã bắt đầu một hội thảo thường niên có tên *"Mạng Nơ-ron cho Máy tính"* (Neural Networks

for Computing). Năm 1987, hội thảo quốc tế đầu tiên về mạng nơ-ron trong thời hiện đại; *"Hội nghị Quốc tế về Mạng Nơ-ron"* (IEEE International Conference on Neural Networks) được tổ chức tại San Diego và *"Hiệp hội Mạng Nơ-ron Quốc tế"* (International Neural Network Society - INNS) được thành lập. Năm 1988, tạp chí "Neural Networks" được thành lập, tiếp theo đó là "Neural Computation" vào năm 1989 và "IEEE Transactions on Neural Networks" vào năm 1990.

Những tiến bộ đáng kể đã được thực hiện trong lĩnh vực mạng nơ-ron đủ để thu hút rất nhiều sự chú ý và tài trợ cho nghiên cứu sâu hơn. Ngày nay, các cuộc thảo luận về mạng nơ-ron đang diễn ra ở khắp mọi nơi. Các chip dựa trên lý thuyết nơ-ron đang nổi lên và ứng dụng cho các vấn đề phức tạp đang phát triển. Rõ ràng, ngày nay là thời kỳ phát triển cho công nghệ mạng nơ-ron, đặc biệt là *mạng nơ-ron học sâu* (deep neural network).

1.2 Các thành phần của mạng nơ-ron

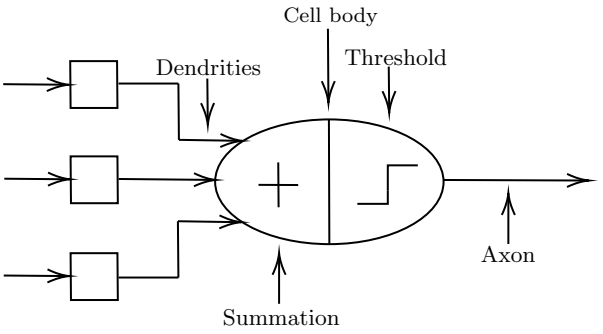
Mạng nơ-ron nhân tạo được lấy cảm hứng và xây dựng nên từ mạng nơ-ron sinh học, vì vậy một mạng nơ-ron nhân tạo cũng bao gồm nhiều nơ-ron đơn lẻ, được gọi là perceptron. Tuy nhiên, để có thể nắm được cách thức hoạt động của một perceptron, trước tiên chúng ta cần phải hiểu được một nơ-ron sinh học hoạt động như thế nào.

Cấu tạo của một nơ-ron được minh họa bằng Hình 1.1. Một nơ-ron sẽ nhận các tín hiệu điện (mức độ mạnh yếu khác nhau) có chứa thông tin từ các *khớp thần kinh* (synapse) của một hay nhiều nơ-ron khác thông qua các *đuôi gai* (dendrit). Các giá trị tín hiệu đầu vào sẽ được tích tụ tại *thân nơ-ron* (cell body). Tại đây, nơ-ron sẽ tiến hành thực hiện một quá trình tính toán bao gồm hai bước.

- Bước 1, nơ-ron sẽ thực hiện *phép tổng chập* (summation) để đạt được tổng giá trị các tín hiệu điện.
- Bước 2, nơ-ron sẽ so sánh tổng tìm được ở Bước 1 với một *ngưỡng cụ thể* (threshold).

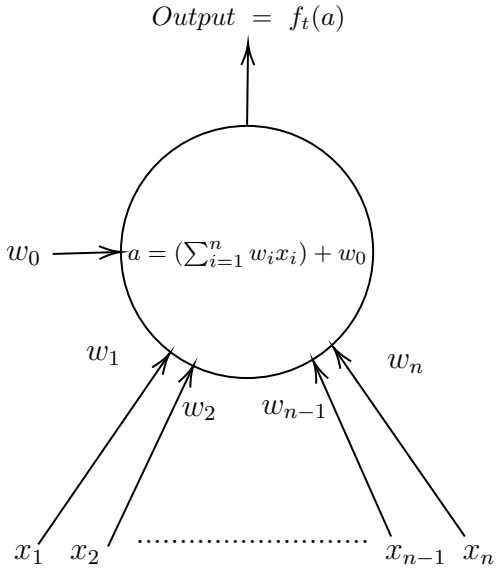
Nếu vượt quá ngưỡng trên, nơ-ron sẽ phát ra một tín hiệu điện, tín hiệu này được truyền qua *sợi trục* (axon) để tới các *khớp thần kinh*. Lúc này, nơ-ron đó được gọi

là đang *kích hoạt (activation)*.



Hình 1.1: Hình ảnh minh họa cấu tạo của một nơ-ron sinh học.

Cấu tạo và cách thức hoạt động của một perceptron Một nơ-ron nhân tạo (perceptron) được cấu tạo bao gồm các thành phần minh họa như Hình 1.2:



Hình 1.2: Hình ảnh minh họa cấu tạo của một perceptron.

Trong đó:

- Các giá trị $x_1, x_2, ..., x_n$ là các tín hiệu đầu vào;
- Các giá trị $w_1, w_2, ..., w_n$ là các trọng số của các nhánh tương ứng truyền giá trị $x_1, x_2, ..., x_n$. Giá trị w_0 được gọi là giá trị bias, có thể nhận một giá trị bất kỳ khác 0;

- Giá trị a được tính là w_0 cộng với tổng của tích các trọng số nhân với giá trị đầu vào tương ứng của nó (tương ứng với bước *summation*). Giá trị a này sẽ được dùng để tính toán hàm kích hoạt $f_{\mathrm{t}}(a)$;
- Kết quả của hàm $f_{\mathrm{t}}(a)$ sẽ được so sánh với *một ngưỡng (threshold)* nhằm xác định perceptron có được kích hoạt hay không. Thông thường, giá trị ngưỡng này sẽ được chọn là 0 để tiện trong việc tính toán. Ngoài ra, người ta thường dùng cặp số (1,0) hoặc (1,-1) để đại diện cho trạng thái kích hoạt/không kích hoạt của perceptron;

Một điểm lưu ý là perceptron hoạt động dựa trên các phép tính toán số học. Vì vậy, các tín hiệu đầu vào cũng như các trọng số đều cần được biểu diễn dưới dạng các giá trị số.

1.3 Mạng nơ-ron đa tầng

Có thể thấy rằng, một perceptron đã có thể được coi là một mạng nơ-ron, tính toán kết quả dựa trên tín hiệu đầu vào và kết quả đầu ra là perceptron đó có được kích hoạt hay không. Trong thực tế, với chỉ một perceptron thì đã có thể giải quyết được bài toán phân loại tuyến tính. Tuy nhiên, đối với các bài toán phức tạp hơn hoặc yêu cầu phân loại phi tuyến (ví dụ như dùng mạng nơ-ron để mô phỏng phép XOR) thì một perceptron không thể đáp ứng được.

1.3.1 Thế nào là mạng nơ-ron đa tầng?

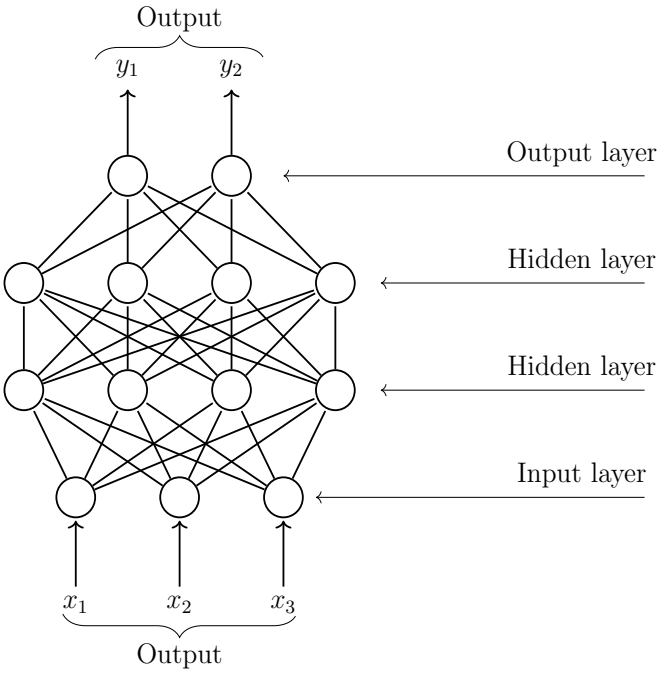
Mạng nơ-ron đa tầng (multilayer perceptrons - MLP) là một trong những hướng tiếp cận để giải quyết vấn đề trên. Trong mạng nơ-ron đa tầng, chúng ta sử dụng nhiều perceptron được sắp xếp thành các *tầng* (layer) khác nhau. Tất cả các perceptron ở tầng sau đều được nối với tất cả các perceptron ở tầng trước, cơ chế này gọi là *fully-connected*.

1.3.2 Các thành phần của một mạng nơ-ron đa tầng

Một mạng nơ-ron đa tầng điển hình thường bao gồm:

1. *Tầng dữ kiện* (input layer): là tầng đầu tiên của mạng, thể hiện các dữ kiện đầu vào;
2. *Tầng kết quả* (output layer): là tầng nằm ở vị trí cuối cùng, thể hiện kết quả đầu ra của mạng;
3. *Tầng ẩn* (hidden layer): là tầng nằm ở giữa, được dùng để tính toán, biến đổi dữ kiện ra đến kết quả;

Lưu ý rằng, một mạng nơ-ron đa tầng chỉ có một tầng dữ kiện và một tầng kết quả, tuy nhiên có thể có một hoặc nhiều tầng ẩn nằm ở giữa. Ví dụ, Hình 1.3 mô tả một mạng nơ-ron trong đó tầng dữ kiện có ba perceptron và tầng kết quả gồm 2 perceptron. Ta có thể thấy mạng này có hai tầng ẩn nằm ở giữa tầng dữ kiện và tầng kết quả.



Hình 1.3: Hình ảnh mạng nơ-ron đa tầng với 2 tầng ẩn.

Ví dụ 1.3.1. Cho một mạng nơ-ron 3 tầng và các véc-tơ trọng số như Hình 1.4 dưới.

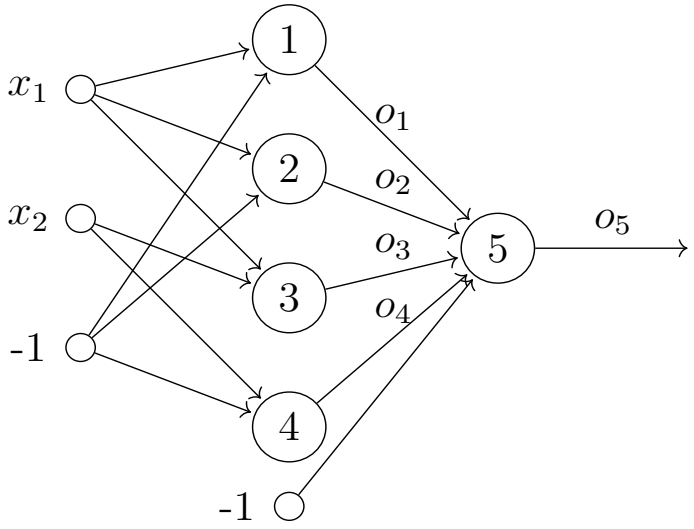
Các véc-tơ trọng số có giá trị như sau:

$$w_1 = [1 \;\; 0 \;\; -1]^T \qquad w_2 = [-1 \;\; 0 \;\; -2]^T$$
$$w_3 = [0 \;\; 1 \;\; 0]^T \qquad w_4 = [0 \;\; -1 \;\; -3]^T$$
$$w_5 = [1 \;\; 1 \;\; 1 \;\; 1 \;\; 3.5]^T$$

Hàm kích hoạt:

$$\varphi(v) = \begin{cases} 1 & \text{nếu } v \geq 0; \\ -1 & \text{nếu } v < 0. \end{cases}$$

Với giá trị nào của x_1 và x_2 thì mạng nơ-ron này có kết quả bằng 1?



Hình 1.4: Mạng nơ-ron biểu diễn Ví dụ 1.3.1

Trả lời. Chúng ta có thể thấy để thỏa mãn yêu cầu, o_5 phải bằng 1, thì nơ-ron số

5 phải được kích hoạt, tương đương với:

$$\begin{aligned} & v_5 >= 0 \\ \Leftrightarrow\ & o_1 * 1 + o_2 * 1 + o_3 * 1 + o_4 * 1 + (-1) * 3.5 >= 0 \end{aligned}$$

Như vậy, để cho biểu thức trên đúng, neuron số 1, 2, 3 và 4 phải được kích hoạt để o_1, o_2, o_3 và o_4 bằng 1 . Nghĩa là v_1, v_2, v_3, v_4 đều phải lớn hơn hoặc bằng 0. Ta có:

$$\begin{aligned} & v_1 = x_1 * 1 + x_2 * 0 + (-1) * (-1) = x_1 + 1 \\ \Rightarrow\ & v_1 >= 0 \Leftrightarrow x_1 >= -1; && (1.1) \\ & v_2 = x_1 * (-1) + x_2 * 0 + (-1) * (-2) = -x_1 + 2 \\ \Rightarrow\ & v_2 >= 0 \Leftrightarrow x_1 <= 2; && (1.2) \\ & v_3 = x_1 * 0 + x_2 * 1 + (-1) * 0 = x_2 \\ \Rightarrow\ & v_3 >= 0 \Leftrightarrow x_2 >= 0; && (1.3) \\ & v_4 = x_1 * 0 + x_2 * (-1) + (-1) * (-3) = -x_2 + 3 \\ \Rightarrow\ & v_4 >= 0 \Leftrightarrow x_2 <= 3; && (1.4) \end{aligned}$$

Từ (1.1), (1.2), (1.3) và (1.4), suy ra để thỏa mãn yêu cầu đã cho thì:

$$-1 <= x_1 <= 2 \ \ \& \ \ 0 <= x_2 <= 3$$

Miền giá trị cần tìm là miền được tô màu đen như Hình 1.5

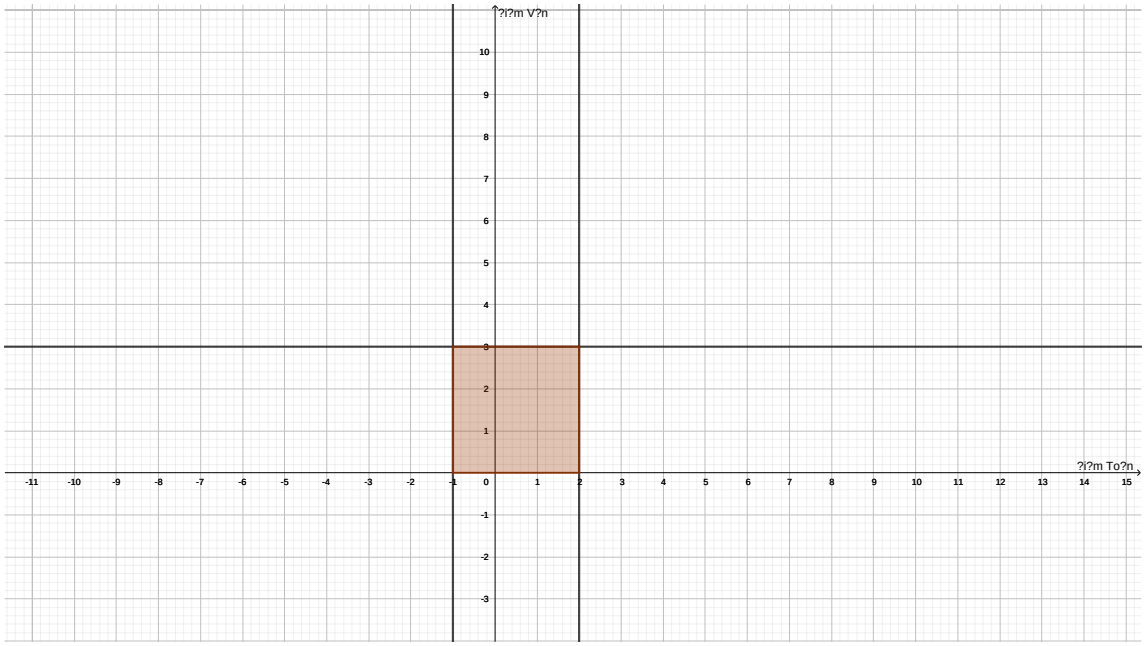
Ví dụ 1.3.2. Cho một mạng nơ-ron ba tầng như Hình 1.6.

Các véc-tơ trọng số có giá trị như sau:

$$w_1 = [1 \ \ 1 \ \ 1.5]^T \quad w_2 = [1 \ \ 1 \ \ 0.5]^T \quad w_3 = [-2 \ \ 1 \ \ 0.5]^T$$

Giả sử rằng:

$$\begin{aligned} 0 \leq x_1 \leq 1 \\ 0 \leq x_2 \leq 1 \end{aligned}$$



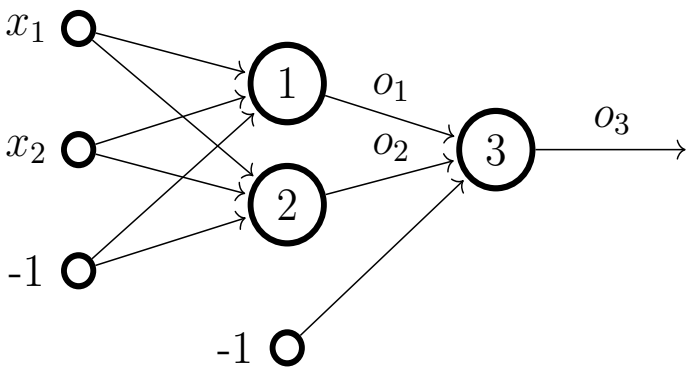
Hình 1.5: Miền giá trị cần tìm trong Ví dụ 1.3.1

Với giá trị nào của x_1 và x_2 thì mạng nơ-ron này có kết quả là 1 với hàm kích hoạt như sau:

$$\varphi(v) = \begin{cases} 1 & \text{nếu } v \geq 0; \\ 0 & \text{nếu } v < 0. \end{cases}$$

Trả lời. Tương tự như ví dụ trên, ta có thể thấy rằng:

do o_1 và o_2 chỉ nhận giá trị là 0 hoặc 1, nên để thỏa mãn biểu thức trên, ta có:



Hình 1.6: Mạng nơ-ron biểu diễn Ví dụ 1.3.2

$o_1 = 0$ và $o_1 = 1$, trong khi đó:

$$\begin{aligned} & o_1 = 0 \\ \Leftrightarrow & v_1 = x_1 * 1 + x_2 * 1 + (-1) * 1.5 = x_1 + x_2 - 1.5 < 0 \\ \Rightarrow & x_1 + x_2 < 1.5 \end{aligned} \tag{1.5}$$

$$\begin{aligned} & o_2 = 1 \\ \Leftrightarrow & v_2 = x_1 * 1 + x_2 * 1 + (-1) * 0.5 = x_1 + x_2 - 0.5 >= 0 \\ \Rightarrow & x_1 + x_2 >= 0.5 \end{aligned} \tag{1.6}$$

Từ 1.5 và 1.6 suy ra kết quả cần tìm: $0.5 <= x_1 + x_2 < 1.5$

Như vậy, kết quả cần tìm là cặp giá trị x_1 và x_2 sao cho điều kiện trên thỏa mãn.

Một điểm đặc biệt là nếu như x_1 và x_2 chỉ nhận giá trị 0 và 1 thì mạng nơ-ron này có thể áp dụng để biểu diễn phép toán luận lý XOR (tham khảo Bảng sự thật 1.7).

Đầu vào		**Đầu ra**
x1	*x2*	
1	0	1
1	1	0
0	0	0
0	1	1

Hình 1.7: Bảng sự thật biểu diễn phép XOR trong Ví dụ 1.3.2

1.4 Sử dụng mạng nơ-ron cho bài toán phân loại

Một trong những ứng dụng quan trọng của mạng nơ-ron là sử dụng cho bài toán phân loại. Đối với bài toán phân loại tuyến tính thì chỉ cần xây dựng mạng nơ-ron với một perceptron. Xét ví dụ sau:

Ví dụ 1.4.1. Một trường chuyên tổ chức thi để tuyển sinh năm học mới, bao gồm hai môn toán và văn, trong đó toán nhân hệ số 2. Để có thể đậu, học sinh cần phải có tổng điểm xét tuyển lớn hơn hoặc bằng 21.

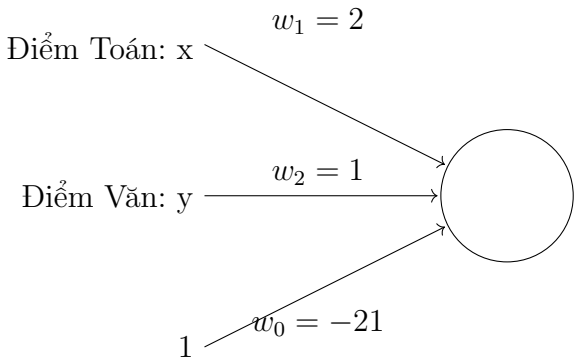
Hãy xây dựng một mạng nơ-ron để phân loại học sinh, trong đó điểm toán được ký hiệu là x, điểm văn được ký hiệu là y. Kết quả của mạng là 0 hoặc 1, tương đương với học sinh đó đậu hoặc rớt.

Trả lời. Theo yêu cầu bài toán, ta cần xây dựng mạng nơ-ron có biểu thức sau:

$$2x + y >= 21$$
$$\Leftrightarrow 2x + y - 21 >= 0$$

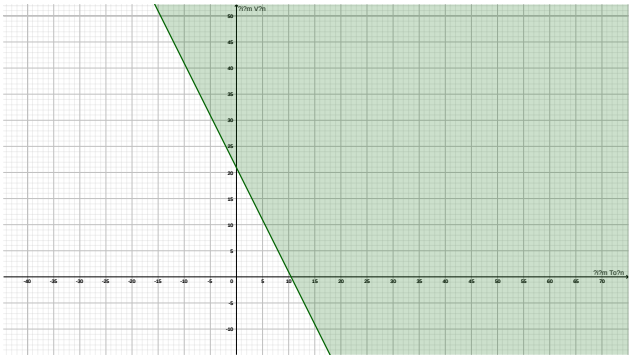
Như vậy, khi chuyển sang biểu diễn dưới dạng một mạng nơ-ron, ta có tập giá trị đầu vào là một véc-tơ [x,y,1] và tập trọng số tương ứng có giá trị là [2,1,-21], với $w_0 = -21$ (trọng số bias tùy ý khác 0).

Ta có tổng trọng số $a = 2x + y - 21$. Mạng nơ-ron này chỉ được kích hoạt (tương



Hình 1.8: Mạng nơ-ron biểu diễn Ví dụ 1.4.1

đương với học sinh đó thỏa mãn điều kiện đậu) khi và chỉ khi $a >= 0$, tương ứng với vùng được tô màu trong Hình 1.9.

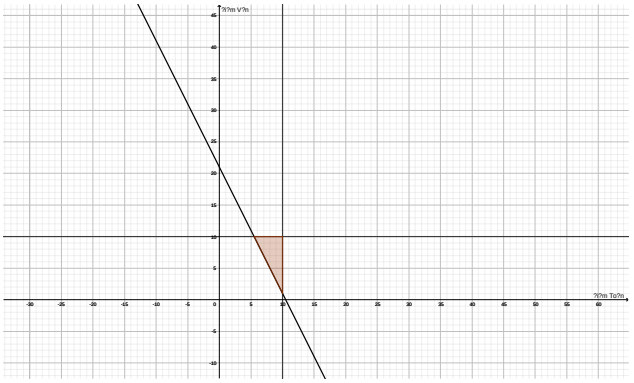


Hình 1.9: Miền giá trị thỏa mãn yêu cầu đề bài

Tuy nhiên, thực tế điểm toán và điểm văn luôn nhận giá trị là số dương $<= 10$. Khi gán thêm điều kiện này, ta có kết quả là miền giá trị được tô màu đen trong đồ thị Hình 1.10.

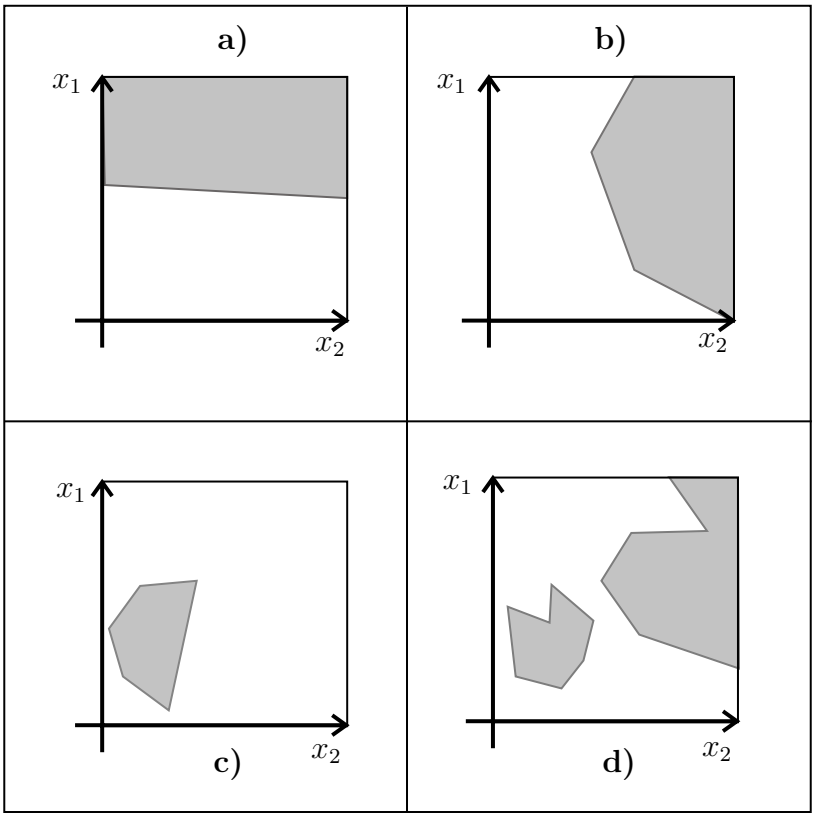
Cần lưu ý rằng trọng số bias phải khác 0 để tránh trường hợp phương trình tổng trọng số a được biểu diễn bởi một đường thẳng đi qua gốc tọa độ $O(0, 0)$. Điều này dẫn tới khi huấn luyện mạng nơ-ron, thì các kết quả mà mạng này học được chỉ xoay quanh gốc tọa độ O, không đảm bảo tính tổng quát để áp dụng cho các bài toán trong thực tiễn.

Ví dụ trên đã sử dụng mạng nơ-ron với một perceptron để phân loại bài toán tuyến tính (như Hình 1.11.a). Tuy nhiên, đối với bài toán phi tuyến (như các Hình 1.11.b,



Hình 1.10: Miền giá trị thỏa mãn yêu cầu đề bài sau khi ràng buộc về khoảng giá trị của điểm toán và điểm văn

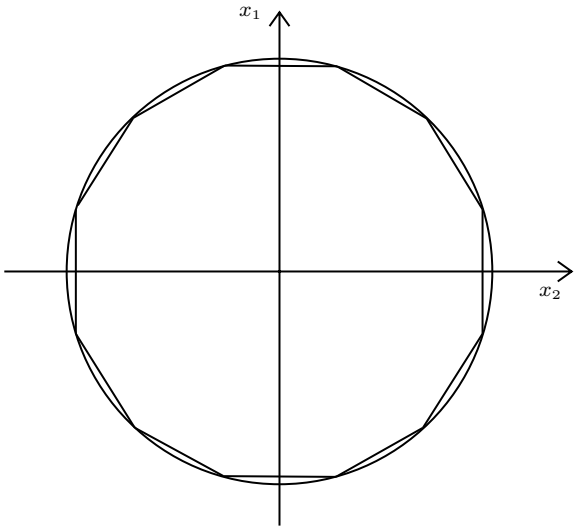
c, d) thì cần phải xây dựng mạng nơ-ron ba tầng.



Hình 1.11: Sử dụng mạng nơ-ron ba tầng để biểu diễn các miền giá trị phi tuyến

Trong đó, tầng đầu tiên (tầng dữ kiện) sẽ chia đồ thị thành hai nửa bởi một đường

thẳng. Tầng thứ hai (tầng ẩn) sẽ hình thành các miền giá trị cần tìm bằng cách kết hợp các phép toán luận lý (AND, OR và NOT) trên nửa đồ thị đã được phân chia của tầng đầu tiên. Bằng cách tăng số lượng tầng ẩn, mạng nơ-ron thậm chí có thể biểu diễn gần đúng các hình tròn thông qua một tập các đường thẳng, ví dụ như Hình 1.12



Hình 1.12: Sử dụng mạng nơ-ron để biển diễn gần đúng miền giá trị là hình tròn

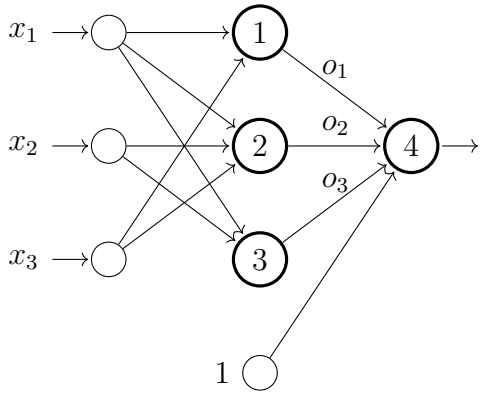
1.5 Kết chương

Chúng ta có thể thấy rằng, mạng nơ-ron được biểu diễn toàn bộ bằng các phép toán số học. Vì vậy, nếu tìm được các bộ trọng số phù hợp với từng bài toán phân loại thì ta hoàn toàn có thể áp dụng mạng nơ-ron một cách hiệu quả, nhanh chóng bằng cách cấu hình sẵn phần cứng với các trọng số cần thiết.

Tuy nhiên, trong thực tế thì việc tìm được các bộ trọng số không dễ dàng, đặc biệt là đối với những bài toán phức tạp và cần mô hình mạng nơ-ron đa tầng với nhiều tầng ẩn và nhiều perceptron. Để giải quyết vấn đề này, các nhà khoa học đã tìm ra được phương pháp huấn luyện mạng nơ-ron để có thể tìm ra bộ trọng số tối ưu, vấn đề này sẽ được trình bày trong chương tiếp theo.

1.6 Bài tập

Bài tập 1.6.1. Cho một mạng nơ-ron ba tầng và các véc-tơ trọng số như Hình 1.13. Với giá trị nào của x_1 và x_2 thì mạng nơ-ron này có kết quả bằng 1? Vẽ biểu đồ minh họa cho mạng trên.



Hình 1.13: Mạng nơ-ron trong bài tập 1.6.1

Các véc-tơ trọng số có giá trị như sau:

$$w_1 = [2, 0, 1]^{\mathrm{T}} \quad w_2 = [1, 2, 1]^{\mathrm{T}} \quad w_3 = [1, 3, 0]^{\mathrm{T}} \quad w_4 = [2, 3, 1, 1]^{\mathrm{T}}$$

Hàm kích hoạt

$$\varphi(v) = \begin{cases} 1 & \text{nếu } v \geq 0; \\ 0 & \text{nếu } v < 0. \end{cases}$$

Bài tập 1.6.2. Thiết kế một mạng nơ-ron để chọn ra các sinh viên toàn diện. Tiêu chí để chọn bao gồm điểm trung bình (số nguyên dương), hạnh kiểm (bao gồm các mức 0-Yếu, 1-Trung Bình, 2-Khá, 3-Giỏi) và điểm rèn luyện (thấp nhất là 0 điểm và cao nhất là 10 điểm). Xét biểu thức sau:

Điểm danh hiệu = điểm trung bình x 2 + điểm hạnh kiểm + điểm rèn luyện;

Sinh viên sẽ được nhận danh hiệu toàn diện nếu như *điểm danh hiệu* lớn hơn hoặc

bằng 24.

Bài tập 1.6.3. Trong Bài tập 1.6.2, nếu như thay đổi bias đã chọn ban đầu thành một giá trị khác (tùy chọn), thì tập trọng số sẽ cần thay đổi như thế nào?

Bài tập 1.6.4. Trong Bài tập 1.6.1, nếu như thay đổi giá trị bias ở tầng thứ 2 từ 1 sang -1 thì miền giá trị x_1 và x_2 cần tìm có thay đổi không? Vẽ lại biểu đồ minh họa cho trường hợp này và so sánh với trường hợp ban đầu.

Chương 2

Hồi quy Logistic

2.1 Giới thiệu về hồi quy logistic

Hồi quy logistic (*logistic regression* Tolles and Meurer, 2016) là mô hình được đặt tên dựa trên hàm số đóng vai trò cốt lõi của mô hình - hàm *logistic*. Hàm logistic được tạo ra ra bởi những nhà thống kê để đặc tả sự bùng nổ dân số. Hàm này có dạng hình chữ S có thể nhận đầu vào là số thực và có miền giá trị là $(0, L)$. Biểu thức toán học tổng quát của hàm logistic:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}, \text{ với } \begin{array}{l} x_0 \text{ là giá trị chính giữa của đường cong logistic.} \\ k \text{ là tỷ lệ tăng của hàm logistic.} \\ L \text{ là giá trị cực đại của hàm logistic.} \end{array} \quad (2.1)$$

Mô hình hồi quy logistic thường được sử dụng cho các bài toán phân loại mặc dù trong tên có chứa từ hồi quy ("*regression*"). Trong phần tiếp theo, chúng ta sẽ làm rõ tại sao hồi quy logistic lại thích hợp với bài toán phân loại, đặc biệt là phân loại nhị phân.

2.2 Bài toán học có giám sát và hàm kích hoạt sigmoid

Cũng giống như *hồi quy tuyến tính* (*linear regression* Freedman, 2009), hồi quy logistic được sử dụng để giải bài toán *học có giám sát* (*supervised learning* Russel, Norvig, et al., 2013). Tuy nhiên, hồi quy logistic được dùng để giải các bài toán phân loại, còn hồi quy tuyến tính được dùng để dự đoán cho các bài toán có giá trị liên tục (như dự đoán giá nhà). Câu hỏi đặt ra là tại sao không dùng hồi quy tuyến tính để giải bài toán phân loại. Chúng ta sẽ lấy ví dụ để minh chứng sự vượt trội của hồi quy logistic so với hồi quy tuyến tính khi áp dụng vào bài toán phân loại, cụ thể là phân loại nhị phân.

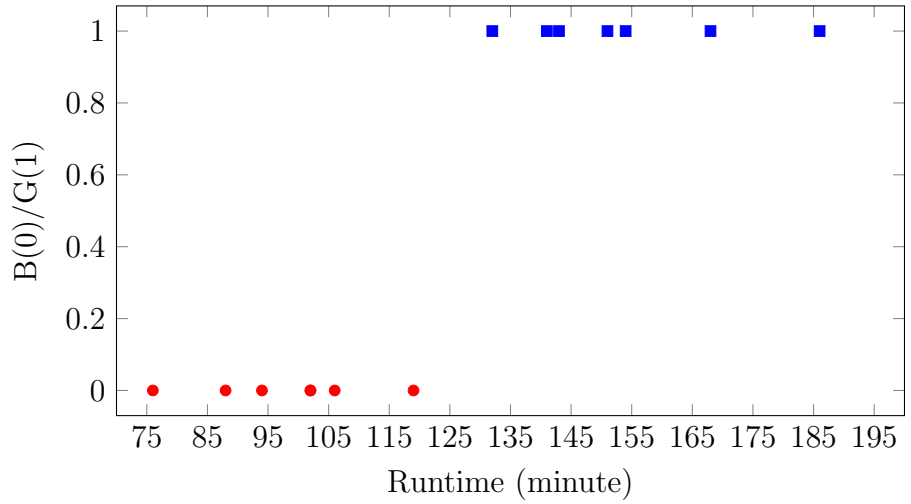
Ví dụ 2.2.1. Một tập hợp gồm 14 bộ phim chiếu rạp có thời lượng phim từ 76 tới 186 phút. Từ thời lượng phim, chúng ta sẽ dự đoán xem bộ phim đó có được khán giả đánh giá tốt (G) hay không tốt (B).

Ta có tập dữ liệu như Bảng 2.1:

Bảng 2.1: Dữ liệu đầu vào của Ví dụ 2.2.1

Runtime	Rating
132	G
141	G
88	B
154	G
151	G
102	B
119	B
106	B
102	B
143	G
76	B
168	G
94	B
186	G

Để trực quan hơn, ta biểu diễn dữ liệu trên tọa độ không gian 2 chiều ở Hình 2.1. Trong đó, điểm hình vuông biểu diễn phim được đánh giá tốt (G) và điểm hình tròn biểu diễn phim không được đánh giá tốt (B).



Hình 2.1: Đồ thị các điểm dữ liệu.

Trong ví dụ này, đầu vào của mô hình là vector $\mathbf{X}$ tương ứng với thời lượng phim và đầu ra là vector $\mathbf{y}$ tương ứng với mức độ đánh giá của khán giả *(G/B)*. Ta khởi tạo vector trọng số $\mathbf{W}$ có chiều dài bằng với vector $\mathbf{X}$ và biến $\mathbf{b}$ (bias). Gọi $\hat{\mathbf{y}}$ là xác suất một bộ phim được đánh giá tốt khi biết thời lượng phim là $(\hat{y} = P(y = 1|x))$ và $\hat{\mathbf{y}}$ nằm trong khoảng $[0, 1]$.

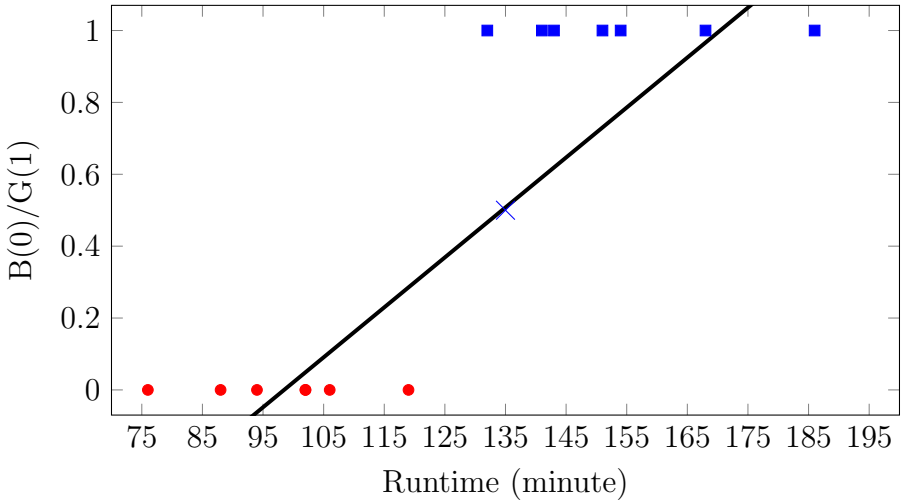
Trong mô hình hồi quy tuyến tính, ta có đầu ra dự đoán là:

$$\hat{\mathbf{y}} = \mathbf{z} = \mathbf{W}^T\mathbf{X} + \mathbf{b} \tag{2.2}$$

Khi áp dụng hồi quy tuyến tính vào bài toán trên chúng ta có thể thu được kết quả như Hình 2.2:

Trong Hình 2.2, đường thẳng chi cắt các điểm hình tròn và các điểm hình vuông biểu diễn đường hồi quy tuyến tính. Có hai vấn đề có thể thấy được từ kết quả này:

- Đường thẳng hồi quy tuyến tính không bị chặn nên nó thể cho ra kết quả không nằm trong vùng mong muốn. Như trên Hình 2.2 đường thẳng có thể



Hình 2.2: Kết quả khi dùng hồi quy tuyến tính.

cho ra giá trị $\hat{\mathbf{y}} > 1$ và $\hat{\mathbf{y}} < 0$ nên mô hình này không phù hợp cho bài toán này. Tuy nhiên, chúng ta có thể chặn đường thẳng trên bằng cách cắt phần nhỏ hơn 0 bằng cách cho chúng bằng 0, cắt các phần lớn hơn 1 bằng cách cho chúng bằng 1.

- Hồi quy tuyến tính không có khả năng phân loại tốt với dữ liệu bị mất cân bằng. Nếu chúng ta lấy điểm trên đường thẳng này có tung độ bằng 0.5 là điểm ngưỡng phân chia hai lớp (đánh giá tốt nếu $\hat{\mathbf{y}} > 0.5$, ngược lại là không tốt). Giả sử có thêm một vài bộ phim dài khoảng 175 phút và được đánh giá tốt. Khi áp dụng mô hình hồi quy tuyến tính, đường thẳng sẽ bị lệch về bên phải (như Hình 2.2) sẽ dẫn đến hiện tượng toàn bộ những bộ phim không tốt sẽ được dự báo là không tốt, nhưng những bộ phim được đánh giá tốt sẽ được dự báo là không tốt.

Vì vậy chúng ta cần một mô hình khác thể cho ta đường phân chia tốt hơn so với hồi quy tuyến tính và thỏa mãn các đặc tính sau:

- Là hàm số nhận giá trị thực, bị chặn trong khoảng (0,1).
- Nếu coi điểm có tung độ là 0.5 làm điểm phân chia thì các điểm càng xa điểm này về phía bên trái có giá trị càng gần 0. Ngược lại, các điểm càng xa điểm này về phía phải có giá trị càng gần 1. Điều này khớp với nhận xét rằng học

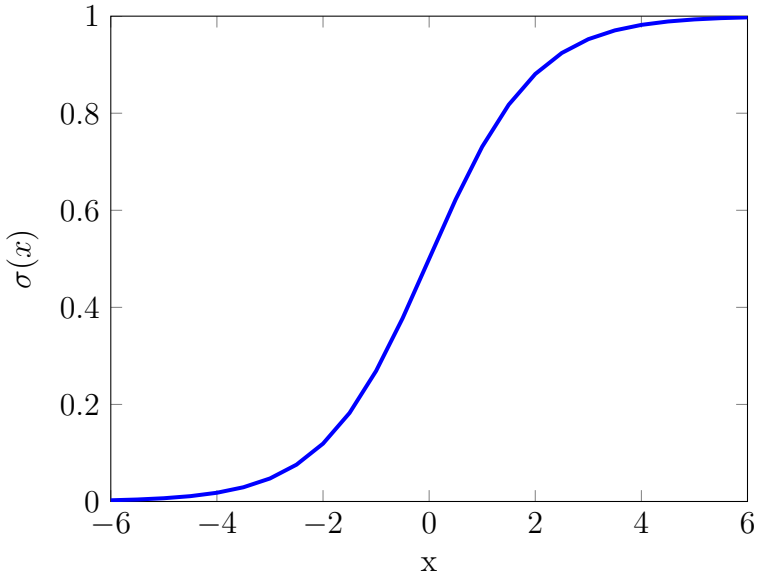
càng nhiều thì xác suất đỗ càng cao và ngược lại.

- Hàm liên tục để có thể đạo hàm mọi nơi, để giúp tìm điểm tối ưu khi sử dụng phương pháp *gradient descent* (sẽ trình bày sau).

Hàm *sigmoid* (Han and Moraga, 1995) (là hàm logistic với $L = 1$, $k = 1$ và $x_0 = 0$) thỏa mãn 3 tính chất nói trên. Phương trình của hàm sigmoid như sau

$$f(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

Đồ thị của hàm sigmoid:



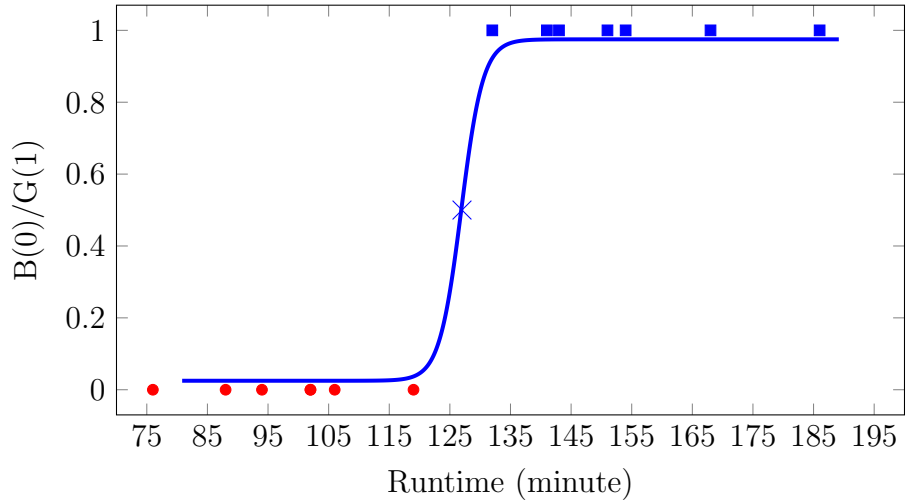
Hình 2.3: Đồ thị các điểm dữ liệu.

Từ phương trình (2.3) và đồ thị Hình 2.3, ta có thể dễ dàng thấy
$\lim_{s\to-\infty} \sigma(s) = 0$ và $\lim_{s\to+\infty} \sigma(s) = 1$. Vì vậy miền giá trị của hàm sigmoid luôn là $(0, 1)$, rất thích hợp với việc phân loại nhị phân. Ngoài ra, khi đầu vào rất lớn (hoặc rất nhỏ) thì giá trị đầu ra của hàm sigmoid luôn tiến tới tiệm cận giá trị 1 hoặc 0. Áp dụng hàm sigmoid vào Ví dụ 2.2.1, ta lấy $\mathbf{W^T X} + \mathbf{b}$ làm dữ liệu đầu vào của

hàm sigmoid. Phương trình mới của kết quả dự đoán $\hat{y}$:

$$\hat{\mathbf{y}} = \sigma(\mathbf{z}) = \sigma(\mathbf{W^T X} + \mathbf{b}) \tag{2.4}$$

Với ngưỡng phân loại là 0.5, như kết quả thu được ở Hình 2.4: ta có thể thấy, hồi quy logistic có khả năng phân loại tốt hơn nhiều so với hồi quy tuyến tính,

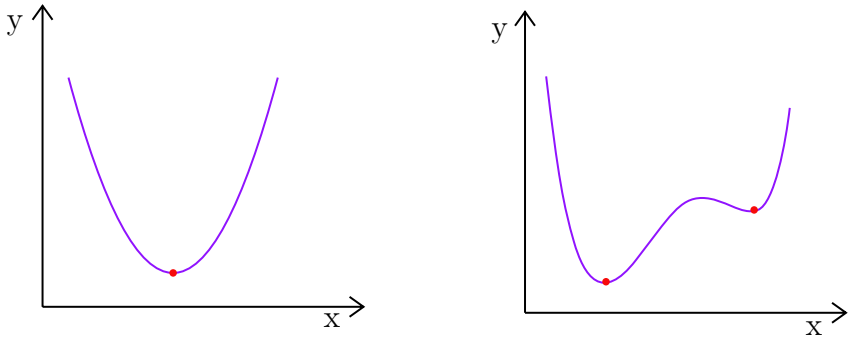


Hình 2.4: Kết quả khi dùng hồi quy logistic.

2.3 Hàm chi phí của hồi quy logistic

Trước khi đi vào nội dung chính của phần này, chúng ta sẽ tìm hiểu về khái niệm *hàm lồi* (*convex function* Bertsekas et al., 2003) và *hàm không lồi* (*non-convex function*):

- Hàm f được gọi là lồi trên đoạn $[\alpha, \beta] \subset R$ nếu với mọi $x, y \in [\alpha, \beta]$ và với mọi $a, b \geq 0$ thỏa $a + b = 1$ thì: $f(ax + by) \geq af(x) + bf(y)$. Nếu f là hàm lồi trên $[\alpha, \beta]$ thì ta có: $min(f) = min\{f(\alpha), f(\beta)\}$ nên hàm lồi chỉ có 1 điểm cực tiểu và đó là điểm mà hàm có giá trị nhỏ nhất trên miền giá trị của nó (*global minimum* Stewart et al., 2020).
- Hàm f được gọi là hàm không lồi khi nó không có tính chất lồi. Đồ thị hàm không lồi thường có dạng sóng với nhiều điểm cực trị (cực tiểu hoặc cực đại) (*local optimal* Stewart et al., 2020).



Hình 2.5: Đồ thị hàm lồi (bên trái) và đồ thị hàm không lồi (bên phải)

Tiếp theo ta sẽ đi vào nội dung chính của phần này. Nhắc lại, trong mô hình hồi quy logistic, ta có:

$$\hat{\mathbf{y}} = \sigma(\mathbf{z}) = \sigma(\mathbf{W^T X} + \mathbf{b})$$

Cũng như phương pháp giải các bài toán học có giám sát, quá trình học của mô hình hồi quy logisitc là quá trình cập nhật trọng số $\mathbf{W}$ vì vậy hồi quy logistic cũng cần có một *hàm mất mát* (*loss function*) để đánh giá sự sai biệt giữa kết quả dự đoán $\hat{y}$ và kết quả thực tế y. Ta ký hiệu $\mathcal{L}(\hat{y}, y)$ là hàm mất mát.

Hàm mất mát đơn giản nhất chính là *Least Squared Error (LSE) Charnes et al., 1976* có phương trình:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2 \tag{2.5}$$

Tuy nhiên, đối với hồi quy logistic, chúng ta không dùng LSE làm hàm mất mát bởi vì phương trình kết quả dự đoán của hồi quy logistic $\hat{\mathbf{y}} = \sigma(\mathbf{W^T X} + \mathbf{b})$ nên khi đó LSE là một hàm không lồi với nhiều điểm cực tiểu (Hình 2.5 bên phải).

Do LSE không đảm bảo tìm được điểm tối ưu toàn cục khi giải bài toán hồi quy logistic nên chúng ta cần có một hàm mất mát khác cho hồi quy logistic - hàm *cross-entropy* (Murphy, 2012). Phương trình hàm mất mát khi đó sẽ là:

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log (1 - \hat{y})) \tag{2.6}$$

Vì tập giá trị của y là $\{0, 1\}$ nên từ phương trình (2.6), ta có thể thấy:

- Khi $y = 0$: phương trình (2.6) sẽ trở thành $L(\hat{y}, y) = -\log(1 - \hat{y})$. Trong trường hợp này, nếu kết quả dự đoán $\hat{y}$ càng gần về giá trị 0 thì hàm mất mát cho ra giá trị càng nhỏ và ngược lại, đáp ứng đúng như yêu cầu chúng ta mong muốn.
- Khi $y = 1$: phương trình (2.6) sẽ trở thành $L(\hat{y}, y) = -\log(\hat{y})$. Trong trường hợp này, nếu kết quả dự đoán $\hat{y}$ càng gần về giá trị 1 thì hàm mất mát cho ra giá trị càng nhỏ và ngược lại, đáp ứng đúng như yêu cầu chúng ta mong muốn.
- Với việc $\hat{y}$ bằng các giá trị đầu mút (0 và 1) khi và chỉ khi đầu vào của hàm sigmoid là vô cùng, điều không xảy ra với các phép toán tự nhiên nên ta có thể thấy hàm mất mát luôn cho giá trị có ý nghĩa.

Tiếp theo, chúng ta sẽ giải thích tại sao cross-entropy có thể giúp mô hình hồi quy logistic tìm được điểm tối ưu toàn cục bằng cách chứng minh hàm mất mát dạng cross-entropy cho mô hình hồi quy logistic là hàm lồi. Ta có hàm logistic của mô hình hồi quy logistic:

$$h(x^{(i)}) = g(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}} \qquad (x^{(i)}, w \in R^D, b \in R) \tag{2.7}$$

Và hàm mất mát của mô hình hồi quy logistic:

$$J(w) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} log(h(x^{(i)}) + (1 - y^{(i)}) log(1 - h(x^{(i)})] \tag{2.8}$$

Tính các *đạo hàm riêng* (*partial derivative*) của hàm mất mát:

$$\frac{\partial J}{\partial h^{(i)}} = -\frac{1}{m}\frac{y^{(i)}(1 - h(x^{(i)})) + (y^{(i)} - 1)h(x^{(i)})}{h(x^{(i)})(1 - h(x^{(i)}))} \tag{2.9}$$

$$\frac{\partial h(x^{(i)})}{\partial w^T x^{(i)} + b} = g(w^T x + b)(1 - g(w^T x + b)) = h(x^{(i)})(1 - h(x^{(i)})) \tag{2.10}$$

$$\begin{aligned}\frac{\partial J}{\partial w} &= \sum_{i=1}^{m}\left[\frac{\partial J}{\partial h^{(i)}}\frac{\partial h(x^{(i)})}{\partial w^{x^{(i)}}+b}\frac{\partial w^{x^{(i)}}+b}{w}\right] \\ &= -\frac{1}{m}\sum_{i=1}^{m}\left[\frac{y^{(i)}(1-h(x^{(i)}))+(y^{(i)}-1)h(x^{(i)})}{h(x^{(i)})(1-h(x^{(i)}))}h(x^{(i)})(1-h(x^{(i)}))\circ x^{(i)}\right] \\ &= \frac{1}{m}\sum_{i=1}^{m}\left[(h(x^{(i)})-y^{(i)})\circ x^{(i)}\right]\end{aligned} \tag{2.11}$$

Ta có $\mathbf{X} = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ ... \\ x^{(m)} \end{bmatrix} \in R^{m\times D}$, $\mathbf{H} = \begin{bmatrix} h(x^{(1)}) \\ h(x^{(2)}) \\ ... \\ h(x^{(m)}) \end{bmatrix} \in R^{m}$, $\mathbf{Y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ ... \\ y^{(m)} \end{bmatrix} \in R^{m}$, khi đó

Ta có các công thức đạo hàm theo w và đạo hàm bậc 2 như sau:

$$\frac{\partial J}{\partial w} = \frac{1}{m}\mathbf{X}(\mathbf{H}-\mathbf{Y}) \tag{2.12}$$

$$\begin{aligned}\frac{\partial^2 J}{\partial w} &= \frac{1}{m}\frac{\partial \mathbf{X}(\mathbf{H}-\mathbf{Y})}{\partial w} \\ &= \frac{1}{m}\mathbf{X}\left(\mathbf{H}\circ(1-\mathbf{H})\circ\mathbf{X}\right)\end{aligned} \tag{2.13}$$

Ta có $H\circ(1-H) = \begin{bmatrix} \lambda_1^2 \\ \lambda_2^2 \\ ... \\ \lambda_m^2 \end{bmatrix}$ trong đó $\lambda_i = \sqrt{h(x^{(i)})(1-h(x^{(i)}))}0 \quad \forall i = 1...m$, khi đó $\forall \mathbf{z} \in R^D$

$$\frac{1}{m}\mathbf{z}^T\mathbf{X}\left(\mathbf{H}\circ(1-\mathbf{H})\circ\mathbf{X}\right)\mathbf{z} = \frac{1}{m}\left\|\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ ... \\ \lambda_m \end{bmatrix}\circ\mathbf{X}\mathbf{z}\right\|^2 \tag{2.14}$$

$\Leftrightarrow \frac{\partial^2 J}{\partial w}$ là một ma trận nửa xác định dương (PSD)
$\Rightarrow J(w)$ là một hàm lồi.

Phương trình (2.6) là hàm mất mát của một điểm dữ liệu đầu vào. Vậy với tập dữ liệu đầu vào có kích thước là m chúng ta sẽ có được phương trình tổng quát của *hàm chi phí* (*cost function*) trong mô hình hồi quy logistic như phương trình (2.15):

$$J(W, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \tag{2.15}$$

2.4 Sử dụng gradient descent để tìm giá trị tối ưu

Nhắc lại về hàm chi phí của hồi quy logistic:

$$J(w, b) = \frac{-1}{m} \sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \tag{2.16}$$

với:

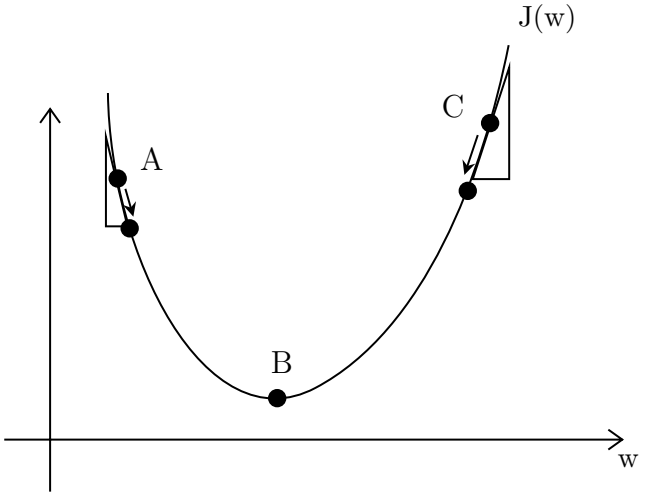
$$\hat{\mathbf{y}} = \sigma(\mathbf{w^T x} + \mathbf{b}) \tag{2.17}$$

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.18}$$

Hàm chi phí này cho ta biết được mức độ sai khác giữa giá trị đầu ra nên ta tính được từ dữ liệu đầu vào ($\hat{y}$) và giá trị đầu ra thật sự của dữ liệu (y) dựa trên giá trị của w và b. Vì vậy, nếu giá trị của chi phí càng nhỏ, thì dữ liệu mà ta tính toán được sẽ càng giống với dữ liệu thật sự. Bài toán bây giờ của chúng ta chính là tìm giá trị của w và b sao cho giá trị của hàm chi phí là nhỏ nhất.

Gradient decent (Lemaréchal, 2012) là một phương pháp được sử dụng rộng rãi để tìm kiếm giá trị tối ưu của hàm chi phí. Với một hàm khả vi và là hàm lồi, ta có thể sử dụng gradient decent để tìm được giá trị gần tối ưu của hàm đó.

Để hiểu rõ hơn về gradient decent, ta sẽ lấy một ví dụ đơn giản về việc sử dụng phương pháp này để tìm giá trị tối ưu của hàm số.



Hình 2.6: Đồ thị minh hoạ kỹ thuật gradient decent.

Hình 2.6 là một đồ thị biểu diễn hàm số $J(w)$ có B là điểm tối ưu. Để đến được vị trí B, đầu tiên ta sẽ chọn một điểm bất kì thuộc đồ thị. Sau đó, ta thay đổi vị trí của điểm đó dựa trên giá trị đạo hàm. Việc cập nhật sẽ được lặp lại cho đến khi điểm ta chọn chạy đến vị trí B. Khi đó, ta đã tìm được giá trị tối ưu của hàm $J(w)$

Công thức của gradient decent là:

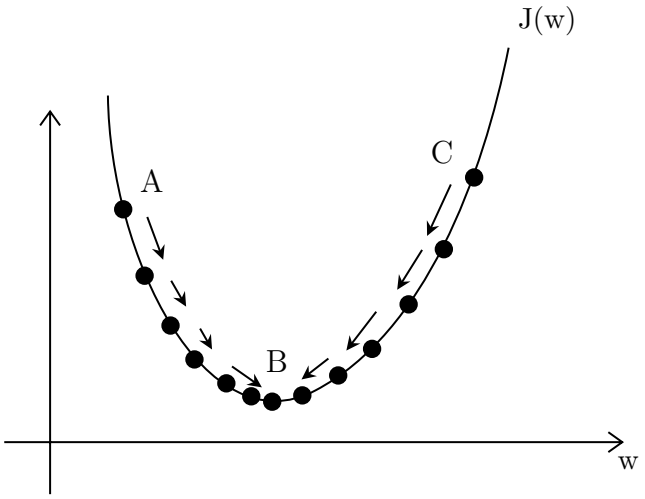
$$w = w - \alpha \frac{\delta J(w)}{\delta w} \tag{2.19}$$

với α được gọi là *tốc độ học* (*learning rate* Murphy, 2012), dùng để xác định độ dài bước đi mỗi lần cập nhật và giá trị α luôn lớn hơn 0.

Đặc điểm của phương pháp này là quá trình *đi ngược chiều đạo hàm*. Nếu ta lấy điểm khởi đầu là A, thì khi đó, đạo hàm tại điểm A sẽ là âm (do $\Delta J(w_A)$ âm còn Δw_A dương). Giá trị w được cập nhật sẽ tăng lên (do $-\alpha \frac{\delta J(w_A)}{\delta w_A} > 0$). Ta có thể thấy điểm A sẽ dời về phía bên phải và "di chuyển" gần hơn đến giá trị tối ưu B.

Trường hợp khác, nếu ta lấy điểm bắt đầu là điểm C, đạo hàm tại điểm C sẽ có giá trị là dương (do $\Delta J(w_C)$ dương và Δw_C dương). Giá trị w sẽ giảm xuống (do $-\alpha \frac{\delta J(w_C)}{\delta w_C} < 0$). Điều đó làm cho điểm C sẽ "di chuyển" về phía bên trái và đến gần hơn với giá trị tối ưu B.

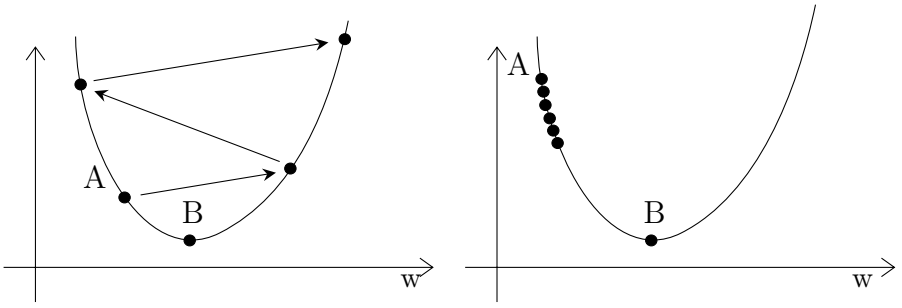
Với số lần lặp vừa đủ và chọn giá trị tốc độ học vừa phải, từ 1 điểm bất kì được chọn trên hàm số, ta đều có thể di chuyển về điểm tối ưu của hàm số.



Hình 2.7: Kết quả khi sử dụng gradient decent để tìm giá trị tối ưu

Có thể thấy, khi sử dụng gradient decent, ta sẽ "tiến về" được giá trị tối ưu của hàm số dù điểm khởi đầu của ta có thể nằm ở vị trí nào. Như Hình 2.7, từ điểm A hay C của đồ thị, với số lần lặp vừa đủ và tốc độ học vừa phải, ta sẽ luôn "tiến về" điểm tối ưu B của hàm số.

Để sử dụng gradient decent tìm giá trị tối ưu, ta phải lựa chọn tốc độ học α không quá lớn hay quá nhỏ. Nếu lựa chọn giá trị α quá lớn, quá trình lặp khi sử dụng gradient decent sẽ không tiến về giá trị tối ưu nữa mà sẽ "chạy" ra xa điểm tối ưu. Còn nếu giá trị α quá nhỏ, số lần lặp để tới được điểm tối ưu sẽ lớn hơn nhiều. Điều này làm cho thời gian chạy để tìm được điểm tối ưu sẽ lâu hơn rất nhiều.



(a) α quá lớn (b) α quá nhỏ

Hình 2.8: gradient decent ứng với các giá trị α

Với Hình 2.8a, đây là khi ta lựa chọn giá trị α quá lớn. Với điểm khởi đầu là A, tại đây khi tính đạo hàm và cập nhật hướng về phía bên phải là chính xác, nhưng khi ta thiết lập giá trị α quá lớn, giá trị mới cập nhật sẽ không phải gần hơn đến điểm tối ưu mà còn bước vượt qua điểm tối ưu này. Nếu quá trình trên được lặp lại với α quá lớn, thì càng ngày ra sẽ đi ra xa điểm tối ưu và sẽ không tìm được kết quả mong đợi.

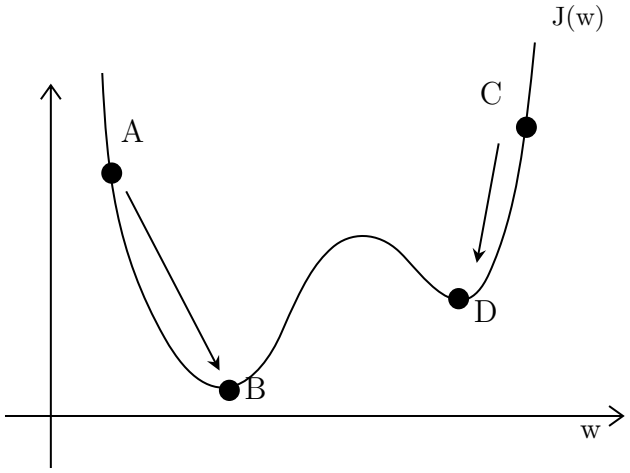
Với Hình 2.8b, đây là khi ta chọn giá trị α quá nhỏ. Với điểm khởi đầu ngẫu nhiên ta chọn là A, ta vẫn di chuyển về vị trí tối ưu nhưng bởi vì giá trị α quá nhỏ, mỗi lần lặp ta chỉ có thể đi về phía tối ưu một khoảng quá ngắn. Vì vậy để về điểm tối ưu, ta phải lặp đi lặp lại quá nhiều bước. Kết quả là thời gian ta đến được điểm cần tìm quá lâu cũng như quá tốn tài nguyên tính toán.

Vì thế, để giải thuật gradient decent chạy tốt, ta cần phải lựa chọn giá trị α cho phù hợp để có thể tìm được giá trị tối ưu một cách nhanh nhất và chính xác.

Gradient decent cũng chỉ có thể tìm được điểm tối ưu khi hàm số thoả mãn những yêu cầu nhất định - nó phải là khả vi và phải là hàm lồi. Nếu không thoả mãn 2 yếu tố trên, giải thuật này sẽ không tìm được điểm tối ưu.

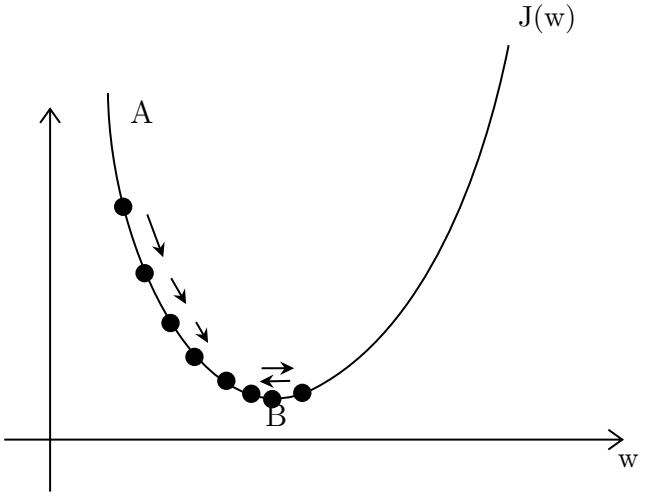
Nếu hàm số khả vi, ta có thể tính được đường đi đến điểm tối ưu của gradient decent dựa trên giá trị của đạo hàm ($w = w - \alpha\frac{\delta J(w)}{\delta w}$). Còn nếu hàm số không khả vi, là ta sẽ không thể tính được giá trị $\frac{\delta J(w)}{\delta w}$ cũng như việc tìm được điểm tối ưu là không khả thi.

Mặt khác một yếu tố cần phải được xét đến đó là hàm số phải là hàm lồi, đây là yếu tố quan trọng để xác định xem ta có thể tìm được điểm tối ưu của hàm số hay không. Vì nếu không là hàm lồi, tuỳ thuộc vào vị trí khởi tạo mà giá trị tối ưu ta thu được sau khi sử dụng gradient decent có thể là điểm tối ưu cục bộ, không phải điểm tối ưu toàn cục.



Hình 2.9: Sử dụng gradient decent với hàm số không là hàm lồi

Đồ thị trên là đồ thị của một hàm không là hàm lồi. Những hàm này ngoài điểm tối ưu toàn cục thì còn có các điểm tối ưu cục bộ. Nếu ta lấy điểm khởi đầu là điểm A, thì sau khi sử dụng gradient decent, điểm tối ưu ta tìm được vẫn là điểm tối ưu toàn cục là điểm B. Nhưng nếu điểm khởi đầu của chúng ta là điểm C, thì sau khi dùng gradient decent thì điểm tối ưu mà ta tìm được sẽ là điểm D, đây không phải là điểm tối ưu toàn cục của hàm. Vì thế, khi sử dụng gradient decent với hàm số không là hàm lồi, giá trị tối ưu ta tìm được có thể không là điểm tối ưu toàn cục. Trên thực tế, khi sử dụng gradient decent, ta chỉ có thể tìm được vị trí gần với điểm tối ưu cục bộ.



Hình 2.10: Kết quả của gradient decent thực tế

Như Hình 2.10, ta lấy điểm khởi đầu là A. Sau đó, ta sẽ sử dụng gradient decent để tìm giá trị tối ưu. Khi này, kết quả của chúng ta sẽ chạy lại gần điểm B. Nhưng, khi tiến lại gần B đến một khoảng nhất định, nếu ta cập nhật giá trị sau khi sử dụng gradient decent, kết quả là ta sẽ vượt qua giá trị tối ưu mộ khoảng rất nhỏ. Và khi ta lặp lại gradient decent một lần nữa, thì ta sẽ đi qua điểm tối ưu 1 khoảng nhỏ tiếp tục. Và cứ như thế, khi thực hiện gradient decent, kết quả mà ta nhận được sẽ dao động môt khoảng nhỏ xung quanh giá trị tối ưu B. Nhưng giá trị sai khác khi ta dùng gradient decent và giá trị tối ưu là rất nhỏ. Vì thế, kết quả ta thu được sau khi sử dụng gradient decent vẫn rất tốt khi áp dụng vào các bài toán tìm giá trị tối ưu.

Quay trở lại với bài toán tìm giá trị nhỏ nhất của hàm chi phí, hàm số của chúng ta gồm 2 biến là w và b. Vì thế, mỗi lần lặp của gradient decent, ta cần phải cập nhật giá trị của cả w và b để thu được giá trị mới nhỏ hơn của hàm chi phí. Ta thực hiện điều này bằng cách sử dụng đạo hàm từng phần hàm số với lần lượt 2 biến.

với hàm chi phí:

$$J(w,b) = \frac{-1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \tag{2.20}$$

Ta sẽ sử dụng đạo hàm từng phần với mỗi biến của hàm chi phí là w và b. Kết quả cập nhật của mỗi biến qua mỗi lần lặp của gradient decent là:

$$w = w - \alpha \frac{\delta J(w,b)}{\delta w} \tag{2.21}$$

$$b = b - \alpha \frac{\delta J(w,b)}{\delta b} \tag{2.22}$$

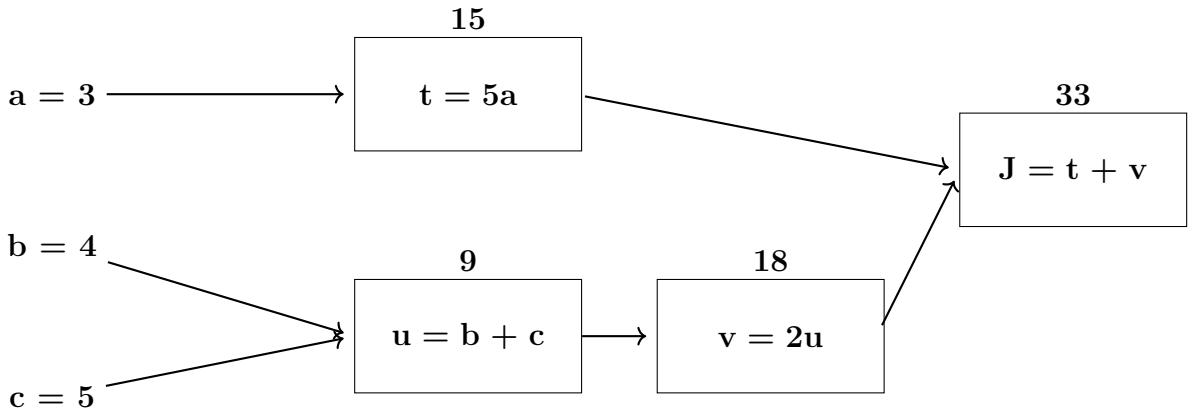
Ta cập nhật giá trị của w và b với số lần lặp vừa đủ cùng α vừa phải sẽ tìm ra được giá trị tối ưu của hàm chi phí. Để tính được giá trị của $\frac{\delta J(w,b)}{\delta w}$ và $\frac{\delta J(w,b)}{\delta b}$, ta có thể dùng *đồ thị tính toán* (*computation graph*) để tìm được các giá trị đó.

2.5 Cách tính đạo hàm riêng dưới góc nhìn đồ thị tính toán

Đồ thị tính toán là một phương pháp giúp tìm đạo hàm riêng một biến của hàm số. Hàm số được biểu diễn thành một đồ thị với các đỉnh con là kết quả phép tính của 2 đỉnh cha. Ta sẽ tính giá trị đạo hàm tại mỗi đỉnh của đồ thị bằng cách lấy giá trị xấp xỉ và lan truyền ngược giá trị về đỉnh cha cho đến biến mà ta cần tính đạo hàm. Ta sẽ xem qua ví dụ sau về việc sử dụng đồ thị tính toán để tính đạo hàm riêng.

Hình 2.11 biểu diễn một hàm số J sử dụng đồ thị tính toán. Hàm số được chia nhỏ thành từng phần và giá trị mỗi phần được tính với các giá trị a, b, c được cho trước. Hàm J có phương trình là:

$$J = 5a + 2(b+c) \tag{2.23}$$



Hình 2.11: Ví dụ sử dụng đồ thị tính toán để tính đạo hàm

Ta chia nhỏ biểu thức 2.23 trên bằng các biểu thức nhỏ hơn:

$$t = 5a \tag{2.24}$$
$$u = b + c \tag{2.25}$$
$$v = 2u \tag{2.26}$$
$$J = t + v \tag{2.27}$$

Với mỗi biểu thức trên là một đỉnh của đồ thị, ta sẽ được một đồ thị tính toán như Hình 2.11. Tiếp theo, ta sẽ tính đạo hàm tại mỗi đỉnh bằng cách xấp xỉ và lan truyền giá trị ngược lại.

Ta sẽ tính đạo hàm tại $a = 3$ bằng phương pháp xấp xỉ. Ta chọn a_1 sao cho Δa rất nhỏ (0.001). Ta lấy $a_1 = 3.001$.Từ biểu thức 2.24 và 2.27, suy ra:

$$t_1 = 5a_1 = 15.005 \tag{2.28}$$
$$J_1 = t_1 + v = 33.005 \tag{2.29}$$

Ta tính giá trị đạo hàm của J theo a:

$$\frac{\Delta t}{\Delta a} = \frac{t_1 - t}{a_1 - a} = \frac{15.005 - 15}{3.001 - 3} = 5 \tag{2.30}$$

$$\frac{\Delta J}{\Delta t} = \frac{J_1 - J}{t_1 - t} = \frac{33.005 - 33}{15.005 - 15} = 1 \tag{2.31}$$

$$\frac{\Delta J}{\Delta a} = \frac{\Delta J}{\Delta t}.\frac{\Delta t}{\Delta a} = 5x1 = 5 \tag{2.32}$$

Ta có thể xấp xỉ $\frac{\Delta J}{\Delta a}$ bằng $\frac{\delta J}{\delta a}$. Từ 2.32, giá trị đạo hàm của J tại $a = 3$ xấp xỉ:

$$\frac{\Delta J}{\Delta a} = 5 \tag{2.33}$$

Ta sẽ sử dụng cách tương tự để tính giá trị đạo hàm của J tại $b = 4$ cũng như $c = 5$. Với $b = 4$, ta chọn $b_1 = 4.001$. Từ 2.25, 2.26, 2.27 suy ra:

$$u_1 = b_1 + c \qquad = 9.001 \tag{2.34}$$

$$v_1 = 2u_1 \qquad = 18.002 \tag{2.35}$$

$$J_1 = t + v_1 \qquad = 33.002 \tag{2.36}$$

Ta tính giá trị đạo hàm của J theo v:

$$\frac{\Delta J}{\Delta v} = \frac{J_1 - J}{v_1 - v} = \frac{33.002 - 33}{18.002 - 18} = 1 \tag{2.37}$$

Sau đó, lan truyền kết quả lên để tính đạo hàm của J theo u dựa trên đạo hàm của J theo v từ 2.37 và đạo hàm của v theo u:

$$\frac{\Delta v}{\Delta u} = \frac{v_1 - v}{u_1 - u} = \frac{18.002 - 18}{9.001 - 9} = 2 \tag{2.38}$$

$$\frac{\Delta J}{\Delta u} = \frac{\Delta J}{\Delta v}\frac{\Delta v}{\Delta u} = 1x2 = 2 \tag{2.39}$$

Sau đó, lan truyền kết quả lên để tính đạo hàm của J theo b dựa trên đạo hàm của

J theo u từ 2.39 và đạo hàm của u theo b:

$$\frac{\Delta u}{\Delta b} = \frac{u_1 - u}{b_1 - b} = \frac{9.001 - 9}{4.001 - 4} = 1 \tag{2.40}$$

$$\frac{\Delta J}{\Delta b} = \frac{\Delta J}{\Delta u}\frac{\Delta u}{\Delta b} = 2x1 = 2. \tag{2.41}$$

Từ 2.41, giá trị đạo hàm của J tại $b = 4$ xấp xỉ bằng 2.

Cuối cùng, với $c = 5$, ta chọn $c_1 = 5.001$. Từ 2.25, 2.26, 2.27 suy ra:

$$u_1 = b + c_1 = 9.001 \tag{2.42}$$

$$v_1 = 2u_1 = 18.002 \tag{2.43}$$

$$J_1 = t + v_1 = 33.002 \tag{2.44}$$

Tính giá trị đạo hàm của J theo v:

$$\frac{\Delta J}{\Delta v} = \frac{J_1 - J}{v_1 - v} = \frac{33.002 - 33}{18.002 - 18} = 1 \tag{2.45}$$

Sau đó, lan truyền kết quả lên để tính đạo hàm của J theo u dựa trên đạo hàm của J theo v từ 2.45 và đạo hàm của v theo u:

$$\frac{\Delta v}{\Delta u} = \frac{v_1 - v}{u_1 - u} = \frac{18.002 - 18}{9.001 - 9} = 2 \tag{2.46}$$

$$\frac{\Delta J}{\Delta u} = \frac{\Delta J}{\Delta v}\frac{\Delta v}{\Delta u} = 1x2 = 2 \tag{2.47}$$

Sau đó, lan truyền kết quả lên để tính đạo hàm của J theo c dựa trên đạo hàm của J theo u từ 2.47 và đạo hàm của u theo c:

$$\frac{\Delta u}{\Delta c} = \frac{u_1 - u}{c_1 - c} = \frac{9.001 - 9}{5.001 - 5} = 1 \tag{2.48}$$

$$\frac{\Delta J}{\Delta c} = \frac{\Delta J}{\Delta u}\frac{\Delta u}{\Delta c} = 2x1 = 2 \tag{2.49}$$

Từ 2.49, giá trị đạo hàm của J tại $c = 5$ xấp xỉ bằng 2.

Khi tính đạo hàm ở từng bước như trên, thay vì tính xấp xỉ, có thể dùng công thức đạo hàm để tính giá trị đạo hàm trực tiếp với các hàm số khả vi. Ta sẽ sử dụng phương pháp này để tính đạo hàm cho khi thực hiện quá trình gradient decent như sau.
Hàm mất mát của hồi quy logistic với giả sử vector nhập x có 2 *đặc trưng* (*feature*) (tức là x có số chiều là 2):

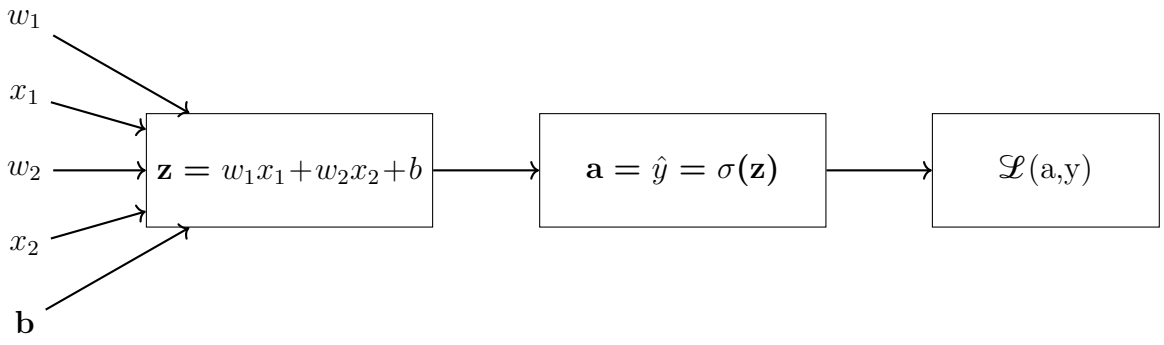
$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a)) \tag{2.50}$$

với:

$$a = \hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.51}$$

$$z = w^T x + b \tag{2.52}$$

Ta có đồ thị tính toán của hàm số trên như sau:



Hình 2.12: Sử dụng đồ thị tính toán vào hồi quy logistic

Tính đạo hàm của $\mathcal{L}$ theo a từ (2.50):

$$\frac{\partial \mathcal{L}}{\partial a} = -\left(y\frac{1}{a} + (1 - y)\frac{-1}{1 - a}\right) = \frac{1 - y}{1 - a} - \frac{y}{a} = \frac{a(1 - y) - y(1 - a)}{a(1 - a)} = \frac{a - y}{a(1 - a)} \tag{2.53}$$

Sau đó, lan truyền kết quả lên để tính đạo hàm của $\mathcal{L}$ theo z dựa trên đạo hàm của

$\mathcal{L}$ theo a từ 2.53 và đạo hàm của a theo z:

$$\frac{\partial a}{\partial z} = \frac{0(1+e^{-z}) - (-e^{-z})1}{(1+e^{-z})^2} = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}\frac{e^{-z}}{1+e^{-z}}} = \frac{1}{1+e^{-z}}\frac{1+e^{-z}-1}{1+e^{-z}} = a(1-a) \tag{2.54}$$

$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a}\frac{\partial a}{\partial z} = \frac{a-y}{a(1-a)}a(1-a) = a-y \tag{2.55}$$

Sau đó, lan truyền kết quả lên để tính đạo hàm của $\mathcal{L}$ theo w_1, w_2, b dựa trên đạo hàm của $\mathcal{L}$ theo z từ 2.55 và đạo hàm của z theo w_1, w_2, b:

$$\frac{\partial z}{\partial w_1} = x_1$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial w_1} = (a-y)x_1 \tag{2.56}$$

$$\frac{\partial z}{\partial w_2} = x_2$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial w_2} = (a-y)x_2 \tag{2.57}$$

$$\frac{\partial z}{\partial b} = 1$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial b} = (a-y) \tag{2.58}$$

Suy ra, công thức đồ thị tính toán cho hàm mất mát từ 2.56, 2.57, 2.58 là:

$$w_1 = w_1 - \alpha\frac{\partial \mathcal{L}}{\partial w_1} = w_1 - \alpha(a-y)x_1 \tag{2.59}$$

$$w_2 = w_2 - \alpha\frac{\partial \mathcal{L}}{\partial w_2} = w_2 - \alpha(a-y)x_2 \tag{2.60}$$

$$b = b - \alpha\frac{\partial \mathcal{L}}{\partial b} = b - \alpha(a-y) \tag{2.61}$$

Hàm chi phí trong trường hợp này là trung bình cộng các hàm mất mát của mỗi mẫu dữ liệu:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(a^i, y^i) \tag{2.62}$$

Từ đó, có thể tính được gradient decent của hàm chi phí bằng cách tính tổng gradient decent các hàm mất mát của dữ liệu từ 2.59, 2.60, 2.61, 2.62:

$$w_1 = w_1 - \alpha \frac{\partial J(w,b)}{\partial w_1} = w_1 - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}(a^i, y^i)}{\partial w_1} = w_1 - \frac{\alpha}{m} \sum_{i=1}^{m} (a^i - y^i) x_1^i \tag{2.63}$$

$$w_2 = w_2 - \alpha \frac{\partial J(w,b)}{\partial w_2} = w_2 - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}(a^i, y^i)}{\partial w_2} = w_2 - \frac{\alpha}{m} \sum_{i=1}^{m} (a^i - y^i) x_2^i \tag{2.64}$$

$$b = b - \alpha \frac{\partial J(w,b)}{\partial b} = b - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}(\dashv\rangle, \dagger^{\rangle})}{\partial b} = b - \frac{\alpha}{m} \sum_{i=1}^{m} (a^i - y^i) \tag{2.65}$$

Một cách tổng quát hơn, với một dữ liệu gồm n đặc trưng, ta có gradient decent là:

$$w_1 = w_1 - \frac{\alpha}{m} \sum_{i=1}^{m} (a^i - y^i) x_1^i \tag{2.66}$$

$$w_2 = w_2 - \frac{\alpha}{m} \sum_{i=1}^{m} (a^i - y^i) x_2^i \tag{2.67}$$

$$\vdots \tag{2.68}$$

$$w_n = w_n - \frac{\alpha}{m} \sum_{i=1}^{m} (a^i - y^i) x_n^i \tag{2.69}$$

$$b = b - \frac{\alpha}{m} \sum_{i=1}^{m} (a^i - y^i) \tag{2.70}$$

Phương pháp cập nhật trọng số bằng cách tính đạo hàm xuất phát từ đỉnh cuối của đồ thị tính toán cho đến các nút lá (ứng với các trọng số cần tính toán) như vậy gọi là phương pháp *lan truyền ngược* (*back propagation* Voleti, 2021).

2.6 Tổng kết phương pháp hồi quy logistic

Phương pháp hồi quy logistic sẽ tìm bộ trọng số phù hợp nhất để tối ưu hàm chi phí (tức là bộ trọng số tìm được sẽ làm cho giá trị hàm chi phí đạt giá trị nhỏ nhất với bộ dữ liệu nhập cho trước). Kỹ thuật hồi quy logistic làm được điều này bằng cách sử dụng hai kỹ thuật gradient descent và lan truyền ngược. Đây cũng là hai kỹ thuật nền tảng cho một mạng nơ-ron nhân tạo. Listing 1 là mã giả của giải thuật hồi quy logistic:

Algorithm 1: Logistic regression

$num\_iter = 500, w_1 = 0, w_2 = 0, ..., w_n = 0, b = 0, alpha = 0.1$
for $i \leftarrow 1$ **to** $range(num\_iter)$ **do**
    $J = 0, dw_1 = 0, dw_2 = 0, ..., dw_n = 0, db = 0$
    for $j \leftarrow 1$ **to** m **do**
        $z = w_1 * x_{j1} + w_2 * x_{j2} + ... + w_n * x_{jn} + b$
        $a = 1/(1 + e^(- z))$
        $J+ = -(y_j * log(a_j) + (1 - y_j) * log(1 - a_j))$
        $dz = a_j - y_j$
        $dw_1+ = dz * x_{j1}$
        $dw_2+ = dz * x_{j2}$
        ...
        $dw_n+ = dz * x_{jn}$
        $db+ = dz$
    $J/ = m$
    $dw_1/ = m$
    $dw_2/ = m$
    ...
    $dw_n/ = m$
    $db/ = m$
    $w_1 = w_1 - alpha * dw_1$
    $w_2 = w_2 - alpha * dw_2$
    ...
    $w_n = w_n - alpha * dw_n$
    $b = b - alpha * b$

Xét trên một tập dữ liệu gồm m dữ liệu và mỗi dữ liệu gồm n đặc trưng. Đầu tiên, chúng ta sẽ khai báo các biến cần thiết. Các w_1, w_2, ..., w_n lần lượt là các trọng số của mỗi đặc trưng thuộc dữ liệu. b là *bias* được thêm vào. Giá trị num_iter là số

lần lặp để tìm được giá trị tối ưu của hàm chi phí. Đây là giá trị ta có thể tuỳ chỉnh. Ta có thể gán 40, 50 hay 500 thậm chí đến 10000 tuỳ thuộc vào số lần lặp cần thiết để tìm được giá trị tối ưu của hàm chi phí. *alpha* là tốc độ học mà ta thiết lập cho gradient decent. Giá trị này không nên quá nhỏ hoặc quá lớn.

Tiếp theo, ta tính gradient decent của hàm chi phí bằng cách lặp lại việc cập nhật các trọng số với số lần lặp num_iter cho trước.

Trong mỗi lần lặp, ta sẽ khởi tạo giá trị của hàm chi phí J, các giá trị đạo hàm của J theo $w_1, w_2, ..., w_n$ là $dw_1, dw_2, ..., dw_n$ và đạo hàm của J theo b là w_b.

Ta sẽ chạy qua từng dữ liệu trong tập dữ liệu cho trước. Ta tính giá trị đầu ra. Tiếp đến, ta tính sự sai biệt giữa giá trị tính toán và giá trị thực tế dựa trên cross-entropy và thêm kết quả vào hàm chi phí. Sau khi chạy qua hết tập dữ liệu, ta tính giá trị đạo hàm của J theo từng trọng số theo công thức được suy ra từ phần trước. Cuối cùng, ta cập nhật giá trị các trọng số dựa trên đạo hàm của chúng theo gradient decent. Quá trình trên sẽ được lặp lại cho đến khi ta tìm được giá trị tối ưu của hàm chi phí và thu được bộ trọng số thoả mãn yêu cầu.

2.7 Kết chương

Qua chương này, chúng ta đã tìm hiểu quá trình hồi quy logistic được sử dụng để tìm bộ trọng số giúp tối ưu hàm lượng giá trong một bài toán học có giám sát. Để làm được điều này, kỹ thuật hồi quy logistic dựa trên hai kỹ thuật quan trọng là gradient descent và lan truyền ngược. Đây cũng là hai kỹ thuật cơ bản của một mạng nơ-ron, như sẽ được trình bày trong chương sau.

2.8 Bài tập

Cân nặng	Giới tính
2.4	Nữ
3.3	Nam
3.2	Nam
2.5	Nữ

Bảng 2.2: Dữ liệu đầu vào của bài tập

Bài tập 2.8.1. Bảng 2.2 mô tả cân nặng và giới tính của trẻ sơ sinh. Sử dụng mô hình hồi quy logistic $h\theta(x) = g(\theta_0 + \theta_1 x_1)$ để giải bài toán phân loại Nam (M) và Nữ (F) dựa theo cân nặng của họ, với x_1 là vector cân nặng những bé sơ sinh trong tập dữ liệu, g là hàm sigmoid, và θ_0, θ_1 là hai trọng số cần phải học. Giả sử $y = 0$ khi một người là nữ và $y = 1$ khi một người là nam, tốc độ học $\alpha = 0.2$ và giá trị khởi tạo của θ_0, θ_1 là 0.5, hàm mất mát là hàm cross-entropy. Với tập dữ liệu training như Bảng 2.2 và sử dụng mô hình hồi quy logistic để cập nhật trọng số θ_0 và θ_1, hãy cho biết giá trị của hai trọng số θ_0 và θ_1 sau 2 lần lặp?

Bài tập 2.8.2. Cho phương trình $J = 3a + 2(4b + 5cd)$

Hãy vẽ đồ thị tính toán của phương trình trên và tính đạo hàm của J tại từng biến a, b, c, d với $a = 2, b = 3, c = 4, d = 5$.

2.9 Phụ lục

Dưới đây là một số thuật ngữ có đề cập trong chương 2:

Bảng 2.3: Bảng các thuật ngữ

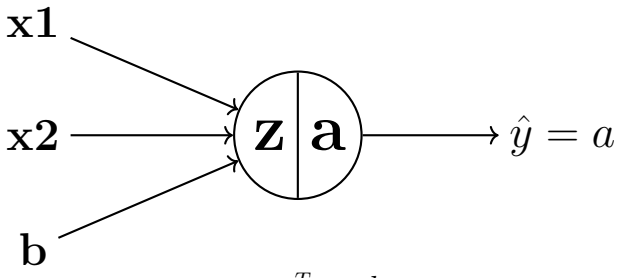
logistic regression	hồi quy logistic
linear regression	hồi quy tuyến tính
perceptron learning algorithm (PLA)	_
supervised learning	học có giám sát
unsupervised learning	học không giám sát
gradient descent	_
convex function	hàm lồi
non-convex function	hàm không lồi
local minimum	cực tiểu địa phương
global minimum	cực tiểu toàn cục
loss function	hàm mất mát
least square error (LSE)	phương pháp bình phương cực tiểu
partial derivative	đạo hàm riêng
cost function	hàm chi phí
learning rate	tốc độ học
computation graph	đồ thị tính toán
feature	đặc trưng
back propagation	lan truyền ngược

Chương 3

Mạng đa tầng và học sâu

3.1 Hồi quy logistic dưới góc nhìn mạng nơ-rôn

Như chương trước trình bày, *hồi quy logistic* (là một *phương pháp phân loại nhị phân* (binary classification method), được xây dựng bởi một hàm có khả năng nhận một giá trị bất kỳ và trả kết quả ra một con số nằm giữa 0 và 1 (hàm *sigmoid*). Điều này cũng tương tự như một *mạng nơ-ron* (neural network) một lớp. Hình 3.1 minh họa cho việc dùng hồi quy logistic như một mạng nơ-ron đơn giản.



$$z = w^T x + b$$

$$a = \sigma(z) = \frac{1}{1+e^z}$$

Hình 3.1: Một mạng nơ-ron đơn giản.

Trong Hình 3.1, ma trận x chứa các *thuộc tính* (feature) của một input, ma trận w sẽ là trọng số các thuộc tính và b là bias. Sau khi trải qua các bước tính toán, kết

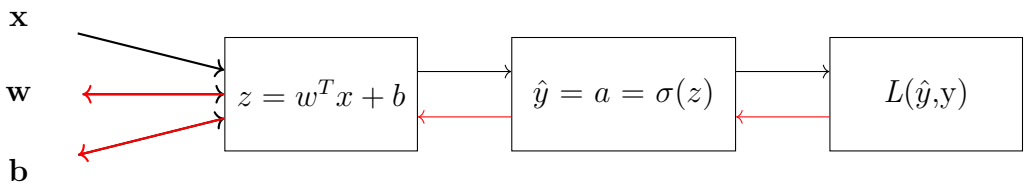
quả thu được $\hat{y}$ sẽ là xác suất mà *nhãn* (label) y nhận giá trị bằng 1 với x và w đã biết như 3.1:

$$P(y = 1|w, x) = \hat{y} \tag{3.1}$$

Như đã trình bài ở trên, kết quả của hàm σ luôn là một giá trị từ 0 đến 1. Nhưng giá trị của nhãn thì luôn là 0 hoặc 1. Sự sai biệt này dẫn đến khái niệm *hàm mất mát* (loss function). Có rất nhiều hàm có thể được dùng để làm hàm mất mát, tuy nhiên trong chương này ta sẽ chỉ dùng *hàm mất mát cross-entropy* như đã trình bày. Nhắc lại, hàm Cross-entropy được định nghĩa như 3.2.

$$L(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log (1 - \hat{y})] \tag{3.2}$$

Giá trị của hàm mất mát là kết quả của *lan truyền xuôi* (forward propagation) và đóng vai trò then chốt trong quá trình *lan truyền ngược* (backward propagation).



Hình 3.2: Lan truyền xuôi và lan truyền ngược.

Quan sát Hình 3.2, quá trình lan truyền ngược và cập nhật trọng số bằng *gradient descent* được thực hiện như sau (giả sử w có kích thước $1 \times n$):

1. Tính đạo hàm riêng phần của $L(\hat{y}, y)$ theo w_i (với $i = 1...n$).

$$\begin{aligned}
\frac{\partial L}{\partial w_i} &= \frac{\partial L}{\partial a} \times \frac{\partial a}{\partial w_i} \\
&= \frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w_i} \\
&= (\frac{1-y}{1-a} - \frac{y}{a}) \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w_i} \\
&= (\frac{1-y}{1-a} - \frac{y}{a}) \times a \times (1-a) \times \frac{\partial z}{\partial w_i} \\
&= (\frac{1-y}{1-a} - \frac{y}{a}) \times a \times (1-a) \times x_i \\
&= (a-y) \times x_i \\
&= (\hat{y}-y) \times x_i
\end{aligned} \tag{3.3}$$

2. Cập nhật lại trọng số bằng gradient descent:

$$\begin{aligned}
w_i &= w_i - \alpha \times \frac{\partial L}{\partial w_i} \\
&= w_i - \alpha \times (\hat{y}-y) \times x_i
\end{aligned} \tag{3.4}$$

Cách cập nhật trọng số được trình bày ở trên xét trên mà trên x có kích thước $1 \times n$. Trường hợp x có kích thước $m \times n$, ta sẽ áp dụng gradient descent với $\frac{\partial J}{\partial w_i}$ thay vì $\frac{\partial L}{\partial w_i}$. Trong đó, J là *cost* được tính theo công thức sau (với $y^{(i)}$ và $\hat{y}^{(i)}$ lần lượt là nhãn và kết quả dự đoán của x^i):

$$J = -\frac{1}{m} \times \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) \tag{3.5}$$

3.2 Mạng nơ-ron đa tầng (MLP) và mạng học sâu

3.2.1 Nhắc lại mạng nơ-ron đa tầng

Như chương trước đã đề cập, mạng nơ-ron đa tầng (multilayer perceptron) thực chất chỉ gồm nhiều tầng mà trong mỗi tầng các perceptron đặt chồng lên nhau, các

tầng được nối với nhau theo cơ chế *fully-connected*. Cách thức hoạt động của những tầng mạng cũng rất đơn giản bằng cách nhân ma trận. Bằng cách này, chúng ta có thể giải quyết rất nhiều bài toán. Tuy nhiên, khi để ý thì tất cả các phép tính này đều xây dựng trên hàm tuyến tính. Điều này gây ra những vấn đề sau:

- Khó khăn trong việc xây dựng những mô hình phi tuyến phức tạp.
- Khi đi qua nhiều lớp, những vấn đề về trọng số sẽ xảy ra ví dụ như đầu ra của một perceptron nào đó quá lớn hoặc quá âm sẽ ảnh hưởng rất nhiều đến độ chính xác của bài toán hoặc thậm chí là máy tính không thể biểu diễn được.
- Vì đầu ra sẽ được so sánh về ngưỡng nào đó, do đó, vấn đề về xác định xác suất sẽ rất khó thực hiện vì chúng ta chỉ thể hiện được mức độ hay nói cách khác là những giá trị rời rạc. Ví dụ như chúng ta muốn nhận diện chó hoặc mèo, thì mô hình xây dựng trên MLP chỉ cho phép ta biết đó là chó hoặc mèo chứ không cho ta biết xác suất mà mô hình nhận diện được vật thể chó là bao nhiêu phần trăm.

3.2.2 Sự tiến hóa của mạng nơ-ron đa tầng

Sau những vấn đề được nêu ra ở trên thì chúng ta thấy rằng, vấn đề đang nằm ở hàm activation và nó cần được thay đổi. Quá trình xây dựng hàm activation này sẽ được tiến hành như sau:

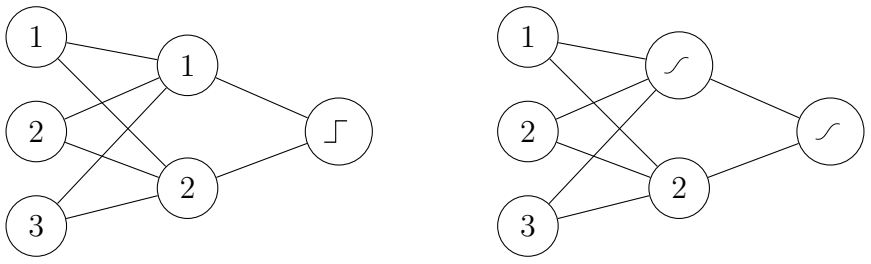
- Do việc xây dựng MLP như trên đang theo mô hình tuyến tính đối với 2 tầng gần nhau. Do đó, dễ nghĩ đến là chúng ta sẽ thiết kế một hàm activation phi tuyến. (1)
- Khi đi qua nhiều lớp, những vấn đề về trọng số sẽ xảy ra ví dụ như đầu ra của một perceptron nào đó quá lớn hoặc quá âm sẽ ảnh hưởng rất nhiều đến độ chính xác của bài toán. Để giải quyết điều này, chúng ta sẽ đi tìm hàm sao cho có thể kéo những giá trị đầu ra về 1 khoảng nào đó có thể kiểm soát được và có ý nghĩa trong xác suất thống kê như (-1,1) hoặc (0,1). (2)
- Vấn đề thể hiện độ tin cậy mà mô hình cho ra sẽ cho chúng ta ý tưởng về thiết kế hàm activation sao cho kết quả cho ra nằm trong khoảng (0,1) thể hiện cho xác suất. (3)

Từ (1),(2),(3) chúng ta sẽ thiết kế hàm activation sao cho đó là hàm phi tuyến (hàm này sẽ phải có đạo hàm) và đầu ra nên có giá trị (-1,1) hoặc (0,1).

Sau một thời gian phát triển, các mạng học sâu đã ra đời và dựa trên những hàm activation này.

3.2.3 Mạng học sâu

Bằng cách áp dụng các hàm kích hoạt *(activation)* khác nhau thay cho ngưỡng *(threshold)* và cách làm tuyến tính như MLP thì mạng học sâu đã ra đời. Chúng ta cũng dễ thấy rằng, không có sự khác biệt quá nhiều giữa MLP và mạng học sâu (như hình minh họa 3.3), hay có thể nói MLP là một tập hợp con của mạng học sâu khi chúng ta chỉ cần thay đổi hàm kích hoạt *(activation)* của từng tầng ẩn hàm kích hoạt *(hidden layers)* và tầng đầu ra *(output layer)* để thể hiện về tính phi tuyến và xác suất.



Hình 3.3: Hình a) miêu tả mạng MLP với 1 hidden layer, Hình b) miêu tả mạng học sâu với 1 hidden layer

Như Hình 3.3, chúng ta có thể thấy rằng, đi qua mỗi layer của mạng học sâu chúng ta có thể thêm vào đó các hàm activation khác nhau, hoặc thậm chí không cần thêm. Còn trong MLP thì chỉ đơn thuần là các phép tính toán ma trận tuyến tính.

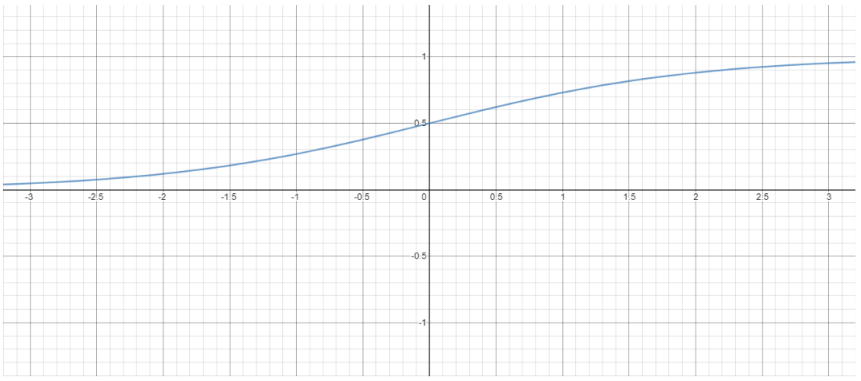
3.3 Các hàm kích hoạt (activation function)

3.3.1 Hàm sigmoid

Công thức:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.6}$$

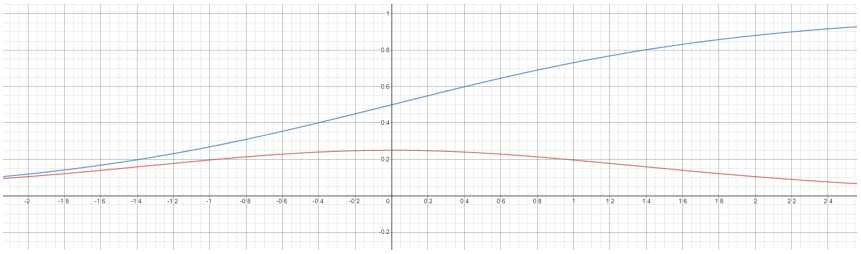
Hàm sigmoid nhận vào một giá trị thực x và trả về một giá trị trong khoảng $(0, 1)$. Nếu x là một số thực âm rất nhỏ thì kết quả của hàm sigmoid sẽ tiệm cận 0, và ngược lại nếu x là một số dương rất lớn thì kết quả sẽ tiệm cận 1. Hình 3.4 bên dưới là đồ thị biểu diễn cho hàm sigmoid.



Hình 3.4: Đồ thị của hàm sigmoid.

Như đã thấy ở phần 3.1, khi sử dụng hàm sigmoid như hàm kích hoạt *(activation)*, việc tính toán vô cùng thuận lợi do kết quả đạo hàm của sigmoid rất "đẹp". Tuy nhiên, điều này không thể che lấp những khuyết điểm nghiêm trọng của sigmoid:

1. **Hàm sigmoid bão hòa và *triệt tiêu gradient* (vanishing gradient)**

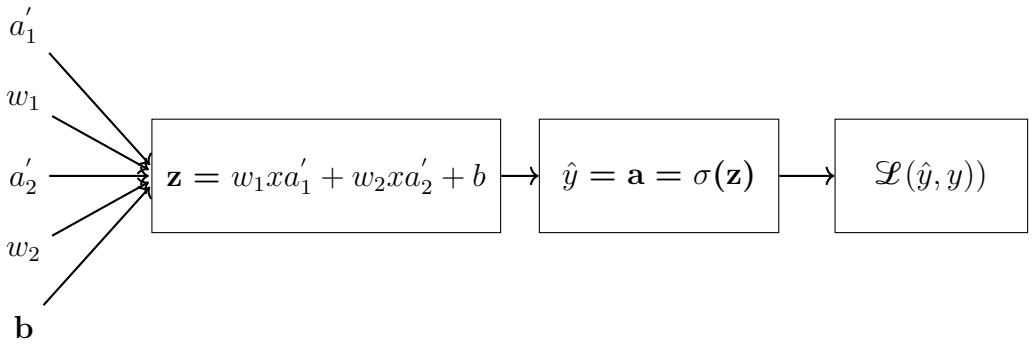


Hình 3.5: Giá trị và đạo hàm của hàm sigmoid.

Trên Hình 3.5, đường phía trên thể hiện cho giá trị của hàm sigmoid và đường phía dưới thể hiện cho giá trị của đạo hàm. Có thể nhận thấy, với những giá trị x rất lớn hoặc rất nhỏ, kết quả đạo hàm của hàm sigmoid rất gần với 0. Điều này gây ra sự triệt tiêu gradient và hạn chế khả năng học của mạng. Cụ thể, nếu mạng được khởi động bằng những trọng số quá lớn hoặc quá nhỏ, giá trị đầu vào của hàm sigmoid bị bão hòa, giá trị của đạo hàm sẽ là một giá trị

gần 0 và gradient sẽ bị triệt tiêu. Nếu mạng được khởi động bằng những trọng số "đẹp" (không quá lớn, không quá nhỏ), giá trị của đạo hàm cũng sẽ là một giá trị trong khoảng (0,0.25). Khi đi qua một mạng nhiều tầng, đạo hàm của các trọng số sẽ nhỏ dần và gradient vẫn sẽ bị triệt tiêu.

2. **Hàm sigmoid không có tính chất *zero-centered***



Hình 3.6: Một tầng ẩn của mạng neural nhiều lớp dùng hàm sigmoid.

Ở ví dụ của mạng neuron như Hình 3.6, đạo hàm riêng phần của hàm mất mát theo hai trọng số w_1 và w_2 sẽ được tính như sau:

$$\nabla w_1 = \frac{\partial L}{\partial z} \times \frac{\partial z}{\partial w_1} = \frac{\partial L}{\partial z} \times a_1' \tag{3.7}$$

$$\nabla w_2 = \frac{\partial L}{\partial z} \times \frac{\partial z}{\partial w_2} = \frac{\partial L}{\partial z} \times a_2' \tag{3.8}$$

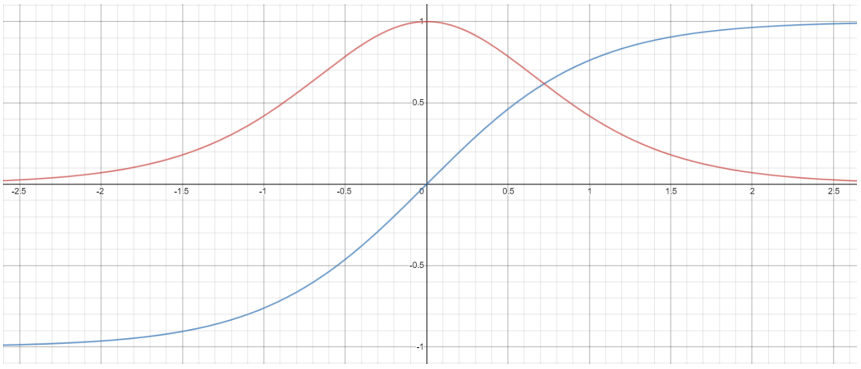
Vì a_1' và a_1' là kết quả của một hàm sigmoid trước đó, do đó luôn nhận giá trị dương và dấu của các gradient sẽ phụ thuộc vào $\frac{\partial L}{\partial z}$. Điều này có nghĩa là các gradient sẽ luôn cùng dương hoặc luôn cùng âm. Việc cập nhật trọng số sẽ chỉ xảy ra về một phía, hạn chế sự linh hoạt của mạng và gây khó khăn cho việc hội tụ.

3.3.2 Hàm tanh

Công thức:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.9}$$

Hàm tanh nhận vào một số thực và trả về một giá trị trong khoảng (-1,1). Trên Hình 3.7, đường màu xanh thể hiện giá trị của hàm tanh và đường màu đỏ thể hiện cho giá trị đạo hàm. Dễ dàng nhận thấy rằng hàm tanh cũng gặp phải vấn đề triệt tiêu gradient như hàm sigmoid. Tuy nhiên, so với sigmoid, hàm tanh có tính chất zero-centered.



Hình 3.7: Đồ thị và đồ thị đạo hàm của hàm tanh.

Hàm tanh cũng có thể được biểu diễn bằng hàm sigmoid như sau:

$$tanh(x) = 2\sigma(2x) - 1 \tag{3.10}$$

3.3.3 Hàm ReLU

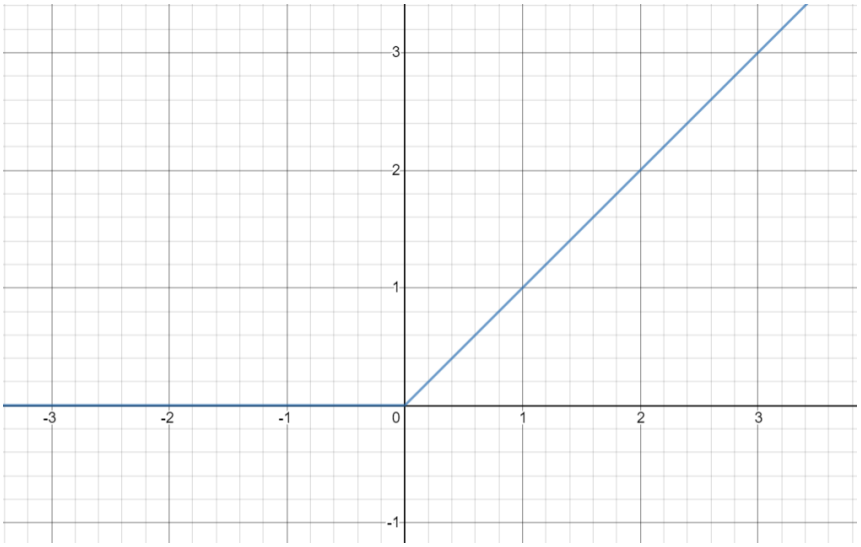
Công thức:

$$f(x) = max(0, x) \tag{3.11}$$

Quan sát công thức 3.11 và Hình 3.8, dễ dàng nhận ra cách hoạt động của hàm ReLU là lọc ra các giá trị đầu vào nhỏ hơn 0. Đạo hàm của ReLU sẽ như 3.12:

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases} \tag{3.12}$$

Như vậy, so với sigmoid và tanh, hàm ReLU sẽ không xuất hiện vấn đề triệt tiêu gradient. Tốc độ tính toán của hàm ReLU cũng sẽ nhanh hơn so với hai hàm trước



Hình 3.8: Đồ thị của hàm ReLU.

đó. Tuy nhiên ReLU cũng tồn đọng một nhược điểm, với x có giá trị nhỏ hơn 0, qua hàm ReLU sẽ thu được kết quả bằng 0. Nếu giá trị của node bị chuyển thành 0 thì sẽ không có ý nghĩa ở lớp tiếp theo và các hệ số tương ứng từ node đấy cũng không được cập nhật với gradient. Hiện tượng này gọi là *Dying ReLU*.

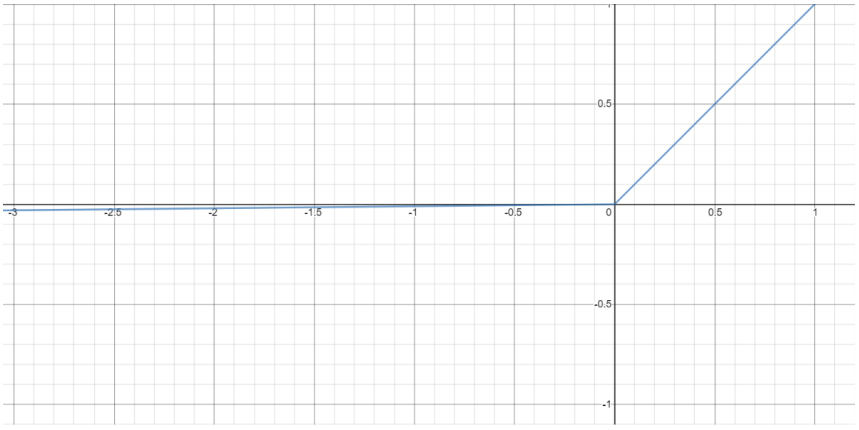
3.3.4 Hàm leaky ReLU

Công thức:

$$f(x) = max(0.01x, x) \tag{3.13}$$

Leaky ReLU là một cố gắng trong việc loại bỏ Dying ReLU. Thay vì luôn trả về giá trị bằng 0 cho các giá trị âm, leaky ReLU tạo một đường xiên có độ dốc nhỏ như Hình 3.9.

Leaky ReLU cũng có một biến thể khác là PReLU: $f(x) = max(\alpha x, x)$ với α sẽ được chọn trong quá trình học.

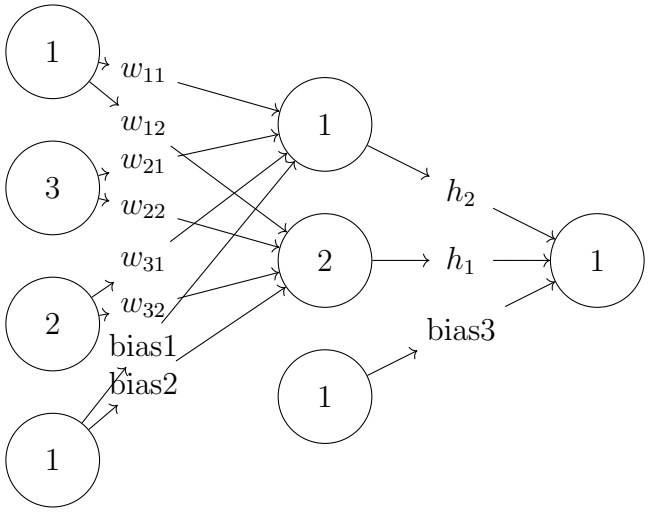


Hình 3.9: Đồ thị của hàm leaky ReLU.

3.4 Biểu diễn mạng nơ-ron như là vector và ma trận

3.4.1 Biểu diễn mạng nơ-ron

Chúng ta xem xét mạng nơ-ron gồm 3 layer như sau:



Hình 3.10: Mạng neural network gồm 3 layer.

Theo Hình 3.10, với f là hàm kích hoạt *(activation)* sẽ cho thấy:

$$\begin{aligned} h_1 &= f(in_1 * w_{11} + in_2 * w_{21} + in_3 * w_{31} + bias_1) \\ h_2 &= f(in_1 * w_{12} + in_2 * w_{22} + in_3 * w_{32} + bias_2) \\ out &= f(h_1 * w_{11} + h_2 * w_{21} + bias_3) \end{aligned} \tag{3.14}$$

Bằng một cách suy nghĩ đơn giản, có thể lưu các giá trị w_i vào một mảng hay một vector và duyệt tuần tự. Nhưng cần để ý rằng, quá trình tính toán ở mỗi perceptron sẽ được tính bằng cách lấy giá trị đầu ra của layers trước đó hoặc là giá trị input ban đầu (gọi chung là input) nhân với trọng số của nó nối với input, quá trình này sẽ được biểu diễn bằng kí hiệu toán học như sau :

$$h_i = f\left(\sum_{j=1}^{num\_input} in_j * w_{ji} + bias_i\right) \tag{3.15}$$

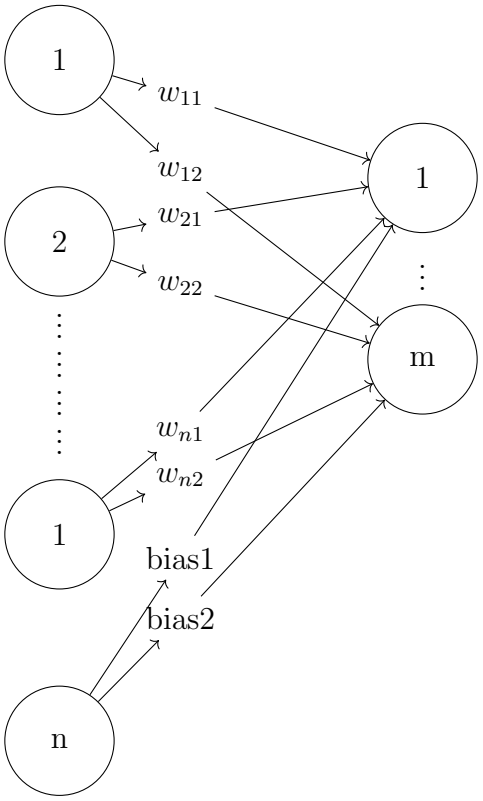
Trong đó, num_input chính là số lượng input để tính toán trong tầng hiện tại. Nếu ai đã quen với việc tính toán ma trận thì sẽ thấy công thức này giống như việc nhân 2 ma trận với nhau. Thật vậy, mỗi đầu ra của một perceptron là một giá trị vô hướng (scalar) và do mối quan hệ giữa 2 tầng liên tiếp là như nhau nên, không mất tính tổng quát, chúng ta chỉ xét riêng trường hợp với 2 tầng (như hình minh họa 3.11).

Với Hình 3.11, nếu chúng ta biểu diễn input thành 1 vector hay nói cách khác là một ma trận X_{nx1}. Ta sẽ xây dựng ma trận trọng số W tương ứng với n hàng, m cột với w_{ij} là trọng số từ input i nối với một output j.

Dựa theo công thức 3.15, chúng ta có thể thấy rằng chỉ cần lấy cột thứ i ra làm vector để nhân với vector trong ma trận cột X và áp dụng activation thì ta sẽ có kết quả của perceptron thứ i. Nếu lúc này chúng ta xét h_{mx1} là ma trận của các perceptron đầu ra thì ta sẽ có công thức như sau:

$$h = f(W^T * X + b) \tag{3.16}$$

Cách biểu diễn mạng nơ-ron dưới dạng ma trận như thế này rất hữu ích trong việc



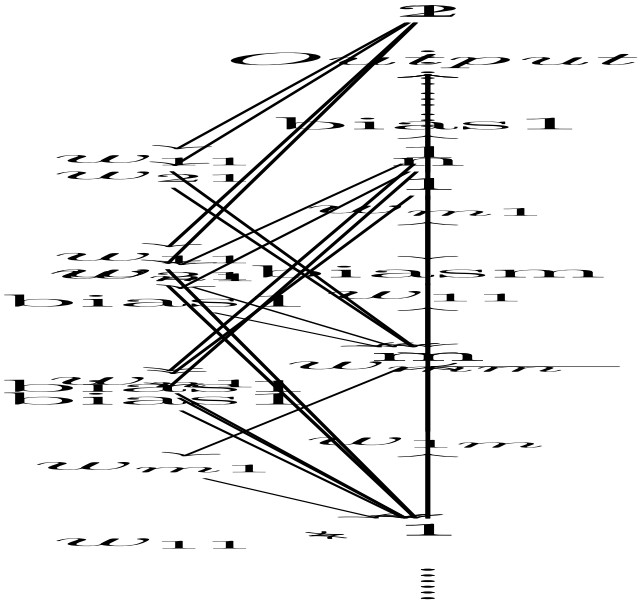
Hình 3.11: Hai layer liên tiếp nhau.

tính toán khi thực hiện các phép lan truyền xuôi và lan truyền ngược.

3.4.2 Cách biểu diễn lan truyền xuôi và lan truyền ngược

Lan truyền xuôi:

Lan truyền xuôi (forward propagation) là quá trình tính toán thông qua các tầng ẩn mà dữ liệu đầu vào được cho ra kết quả trong mô hình. Ta kí hiệu mỗi tầng i trong mạng được cấu tạo từ 3 phần bao gồm input $X^{(i)}$, ma trận trọng số $W^{(i)}$ và hàm activation $f^{(i)}$, trong đó, output của tầng này sẽ là input của tầng tiếp theo. Xem xét hình minh họa 3.12 dưới đây cho quá trình lan truyền xuôi.



Hình 3.12: Lan truyền xuôi

Như chúng ta thấy trong Hình 3.12 mạng gồm 4 layers (1 input layer, 2 hidden layer và 1 output layer).

$$\begin{aligned} X^{(1)} &= f^{(1)}(W^{(1)T} * Input + b^{(1)}) \\ X^{(2)} &= f^{(2)}(W^{(2)T} * X^{(1)} + b^{(2)}) \\ Output &= f^{(3)}(W^{(3)T} * X^{(2)} + b^{(3)}) \end{aligned} \tag{3.17}$$

Với cách tính 3.17, thì chúng ta có thể mở rộng một cách tông quát với 1 input layer, n hidden layer và 1 output layer như sau:

$$\begin{aligned} X^{(1)} &= f^{(1)}(W^{(1)T} * Input + b^{(1)}) \\ X^{(i)} &= f^{(i)}(W^{(i)T} * X^{(i-1)} + b^{(i)}), i = \overline{2, n} \\ Output &= f^{(n)}(W^{(n)T} * X^{(n)} + b^{(n+1)}) \end{aligned} \tag{3.18}$$

Lan truyền ngược

Lan truyền ngược (backward propagation hay back propagation) là quá trình cập nhật lại cái trọng số để tìm ra bộ trọng số phù hợp để làm mô hình cho bài toán cần

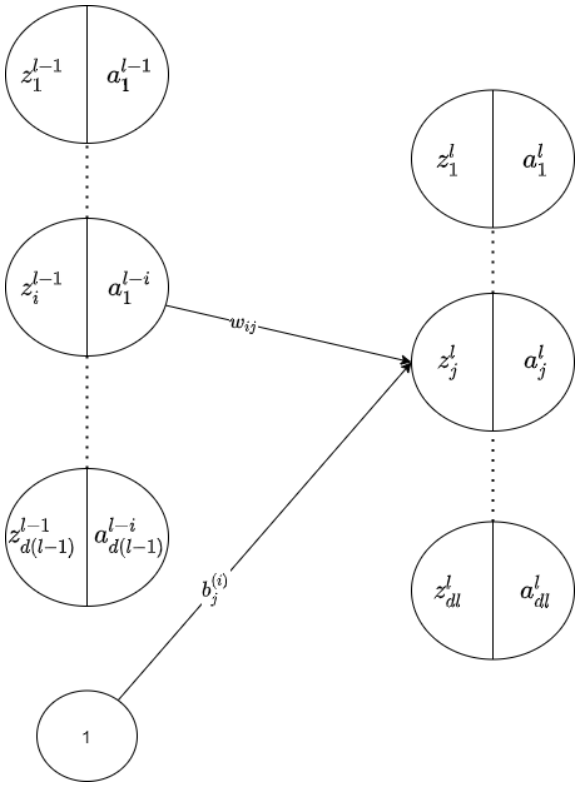
giải quyết. Khác với lan truyền xuôi khi sử dụng hàm activation, lan truyền ngược dựa trên việc tính đạo hàm để cho ra kết quả.

Để cập nhật được trọng số, thì chúng ta cần có hàm loss giả sử được định nghĩa là J và để dễ cho quá trình tính toán, chúng ta sẽ kí hiệu thêm biểu thức sau:

$$z^{(i)} = W^{(i)T} * X^{(i)} + b^{(i)} \tag{3.19}$$

$$a^{(i)} = f(z^{(i)}) \tag{3.20}$$

Các kí hiệu này được biểu diễn như hình sau. Để đỡ rối khi nhìn, xin phép chỉ kí hiệu tượng trưng các đường cần thiết, thực tế sẽ mỗi perceptron ở layer trước phải nối hết với các perceptron layer sau.



Hình 3.13: Đồ thị minh họa lan truyền ngược.

Tiếp theo, chúng ta sẽ có được gradient của các trọng số ở từng layer l như sau:

$$\frac{\partial J}{\partial w_{ij}^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} * \frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}} = e_j^{(l)} * a_i^{(l-1)} \tag{3.21}$$

với

$$\begin{aligned} e_j^{(l)} &= \frac{\partial J}{\partial z_j^{(l)}} = \frac{\partial J}{\partial a_j^{(l)}} * \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \\ &= \left(\sum_{k=1}^{d^{(l+1)}} \frac{\partial J}{\partial z_k^{(l+1)}} * \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} \right) f'(z_j^{(l)}) \\ &= (w_{j:}^{(l+1)} * e^{(l+1)}) f'(z_j^{(l)}) \end{aligned} \tag{3.22}$$

Trong đó $e^{(l+1)} = [e_1^{(l+1)}, e_2^{(l+1)}, ..., e_{d(l+1)}^{(l+1)}]^T$ và $w_{j:}^{(l+1)}$ là toàn bộ phần tử hàng thứ j của ma trận $W^{(l+1)}$. Tổng xuất hiện trong công thức trên 3.22 thể hiện việc $a_j^{(l)}$ đóng góp vào việc tính các $z_k^{(l+1)}$. Với cách làm tương tự, ta có:

$$\frac{\partial J}{\partial b_j^{(l)}} = e_j^{(l)} \tag{3.23}$$

Từ đây chúng ta có thể mở rộng ra cho cách biểu diễn bằng ma trận như các công thức sau:

$$\begin{aligned} \frac{\partial J}{\partial W^{(l)}} &= a^{(l-1)} e^{(l)T} \\ e^{(l-1)} &= W^{(l)T} e^{(l)} f'(z^{(l)}) \\ \frac{\partial J}{\partial b^{(l)}} &= e^{(l)} \end{aligned} \tag{3.24}$$

3.4.3 Vì sao lại biểu diễn bằng ma trận

Để giải quyết cho câu hỏi này, hãy thử hiện thực các tính toán trong mạng neural mà không dùng đến ma trận với ngôn ngữ minh họa là Python:

```
J = 0
```

```
for i = 1 to m:
    z[i] = b
    for j in range(len(w)):
        z[i] += w[j]*x[i][j]
    a[i] = Sigmoid(z[i])
    J += -[y[i]*log(a[i]) + (1-y[i])*log(1-y[i])]
    dz[i] = a[i] - y[i]
    for j in range(len(w)):
        dw[j] += x[i][j]*dz[i]
J = J / m
for j in range(len(w)):
    w[j] = w[j] - alpha*dw[j]/m
```

Và hiện thực các tính toán có sử dụng ma trận, cũng với ngôn ngữ này:

```
import numpy as np

Z = np.dot(w.T,X) + b
A = Sigmoid(Z) #1
dZ = A - Y
dW = (1/m) * X * dZ.T
dB = (1/m) * np.sum(dZ)
W = W - alpha * dW
b = b - alpha * dB
```

Có thể thấy, việc hỗ trợ tính toán bằng ma trận giúp ích rất nhiều trong quá trình hiện thực, đặc biệt cách tính toán bằng ma trận như vậy có thể thực hiện song song trên những kiến trúc máy tính hỗ trợ đa luồng. Đó cũng là lý do tại sao những ngôn ngữ có hỗ trợ tính toán ma trận như Python được sử dụng phổ biến trong hiện thực mạng nơ-ron.

Một lưu ý nữa được rút ra từ ví dụ trên là các nút ở cùng một tầng thì phải có cùng một hàm activation để việc áp dụng hàm activation (dòng 1) cũng ma trận hóa và trở nên dễ dàng hơn.

3.4.4 Kỷ nguyên mới của mạng nơ-ron

Sau khi đã đi qua những lý thuyết của mạng nơ-ron thì chúng ta đã thấy được sức mạnh của phương pháp này. Tuy nhiên, đã có thời gian mạng nơ-ronkhông được sử dụng nhiều và chìm trong màn đêm. Vấn đề nằm ở chỗ, việc tính toán trên các mạng nơ-ron với các quá trình tính toán ma trận như trên đòi hỏi quá nhiều tài nguyên tính toán và vượt quá khả năng xử lý của phần cứng máy tính dương thời. Tuy nhiên, sau khi phần cứng phát triển vượt bậc thì đã mở ra kỉ nguyên mới cho mạng nơ-ron với sự ra đời của GPU (Graphics Processing Unit).

Để giải thích việcnày, chúng ta quay lại việc lan truyền xuôi và lan truyền ngược sử dụng ma trận. Có thể nói rằng việc biểu diễn mạng nơ-ron bằng ma trận là cách tốt nhất trong hiện tại. Do đó, kiến trúc phần cứng nào hỗ trợ xử lý và tính toán trên ma trận tốt nhất sẽ có khả năng xử lý mạng nơ-ron hiệu quả nhất.

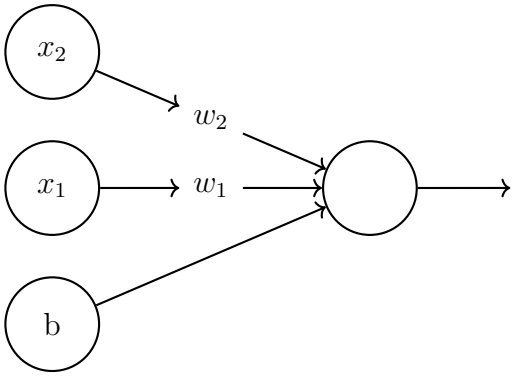
Như tên gọi của mình, GPU được dùng để hỗ trợ xử lý các dạng biểu diễn hình học trong đồ họa máy tính. Trong máy tính, các hình ảnh được biểu diễn bằng ma trận các *điểm ảnh* (pixel), vì vậy kiến trúc nội tại của GPU được xây dựng để hỗ trợ tối đa các phép tính toán trên ma trận để có thể xử ký hình ảnh tốt nhất. Thêm vào đó, GPU ngày nay lại được xử lý đa luồng và tiếp nhận thông tin gấp nhiều lần CPU để hỗ trợ đồ họa tạo hiệu ứng tối đa. Đây cũng là lý do mà chúng ta phải dùng ma trận hóa ở phần trước để tính toán khi hiện thực quá trình xử lý tính toán trên mạng nơ-ron và cũng là lý do vì sao các framework về học sâu ngày nay **chỉ cho dùng 1 hàm activation cho mỗi tầng**.

3.5 Kết chương

Qua chương này, chúng ta đã tìm hiểu và phân tích được cách thức hoạt động của mạng học sâu mà ngày nay được sử dụng ngoài thực tế. Ngoài ra, các vấn đề liên quan tới GPU và ma trận hóa cũng đã được trình bày để thấy được sự ảnh hưởng của phần cứng và sự hỗ trợ của ngôn ngữ khi chúng ta huấn luyện mạng học sâu. Từ đó chúng ta có phương pháp lựa chọn phần cứng, ngôn ngữ và thiết kế mô hình mạng hợp lý để tối ưu năng suất nhất có thể.

3.6 Bài tập

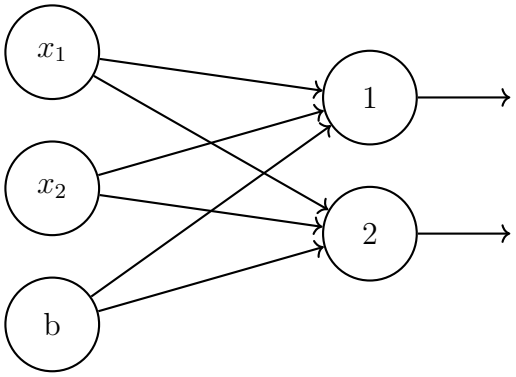
Bài tập 3.6.1. Cho mạng nơ-ron như hình vẽ sau:



Hình 3.14: Mạng neural network một output.

Biết rằng $w_1 = w_2 = w_{bias} = 0.25$, $x_1 = 0.1, x_2 = 0.2, bias = 1$ và output mong đợi là 1. Hãy tính output của mạng và các trọng số và bias mới(với hệ số học alpha là 0.1 và hàm lỗi là *Cross Entropy*) của mạng khi hàm activation là hàm *sigmoid*.

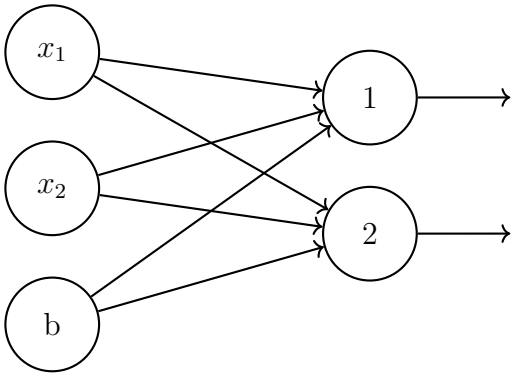
Bài tập 3.6.2. Cho mạng neural network như hình vẽ sau:



Hình 3.15: Mạng neural network hai output.

Biết rằng các trọng số và bias đều bằng 0.5, $x_1 = 0.1, x_2 = 0.2$ và output 1, 2 mong đợi lần lượt là 0, 1. Hãy dựa theo kiến thức của chương này, tìm ra các ma trận được tạo ra và dùng vectorize để tính toán thực hiện lan truyền xuôi.

Bài tập 3.6.3. Cho mạng neural network như hình vẽ sau:



Hình 3.16: Mạng neural network hai output.

Biết rằng các trọng số và bias đều bằng 0.5, $x_1 = 0.1, x_2 = 0.2$ và output 1, 2 mong đợi lần lượt là 0, 1. Hãy tính output của mạng và các trọng số và bias mới (với hệ số học alpha là 0.1 và hàm lỗi là *Cross Entropy*) của mạng với các activation sau:

1. Hàm activation của output 1, 2 lần lượt là hàm *tanh*, *sigmoid*.
2. Hàm activation output 1, 2 lần lượt là hàm *ReLU*, *sigmoid*.

3.7 Phụ lục

Dưới đây là một số thuật ngữ có đề cập trong chương 3:

Bảng 3.1: Bảng các thuật ngữ

Logistic regression	Hồi quy logistic
binary classification	phân loại nhị phân
neural network	mạng nơ-ron
feature	đặc trưng, thuộc tính
label	nhãn
multilayer perceptrons (MLP)	mạng nơ-ron đa tầng
fully-connected	kết nối đầy đủ
activation function	hàm kích hoạt
threshold	ngưỡng
input layer	tầng dữ kiện
hidden layer	tầng ẩn
output layer	tầng kết quả
vanishing gradient	triệt tiêu đạo hàm
learning rate	hệ số học, tốc độ học
zero-centered	???
pixel	điểm ảnh
GPU	Graphics Processing Unit
backward propagation	lan truyền ngược
forward propagation	lan truyền xuôi

Chương 4

Mô hình xử lý ngôn ngữ tự nhiên bằng mạng học sâu

4.1 Giới thiệu về xử lý ngôn ngữ tự nhiên

Xử lý ngôn ngữ tự nhiên (*Natural Language Processing - NLP* Guida and Mauri, 1986) là một nhánh của *Trí Tuệ Nhân Tạo* (*Artificial Intelligence - AI* Russell and Norvig, 2002), tập trung vào các ứng dụng trên ngôn ngữ của con người. Đây là một trong những chủ đề khó của Trí Tuệ Nhân Tạo vì nó liên quan đến việc phải hiểu ý nghĩa ngôn ngữ - một công cụ hoàn hảo của tư duy và giao tiếp.

Xử lý ngôn ngữ tự nhiên có thể được hiểu ngắn gọn là khả năng xử lý ngôn ngữ của con người (thường được khái quát là tự nhiên), vì ngôn ngữ con người được sử dụng trong hình ảnh, viết, nói,... Từ *xử lý* (process) trong từ "Xử lý ngôn ngữ tự nhiên" có nghĩa là: "Thay đổi một cái gì đó thành một thứ khác". Trong trường hợp này có nghĩa là "Ngôn ngữ của con người được biến thành các biểu diễn mà máy tính có thể hiểu và xử lý". Ví dụ: Những từ mà bạn đang đọc và viết dưới dạng ngôn ngữ tự nhiên (tiếng Việt bằng các ký tự latin) và được lưu trong ngôn ngữ máy tính (dạng nhị phân gồm các chuỗi số gồm 0 và 1). Tuy nhiên, xử lý ngôn ngữ tự nhiên không phải chỉ là chuyển các ký tự trong bảng chữ cái thành các bit. Nó liên quan nhiều hơn đến việc làm cho máy tính có thể hiểu, xử lý một cách tự động (hoặc làm một số tác vụ liên quan đến ngôn ngữ) dựa trên cách ngôn ngữ của con người được

biểu diễn và tổ chức.

Các ứng dụng nổi bật của xử lý ngôn ngữ tự nhiên gồm có:

- *Phân tích cảm xúc* (*Sentiment Analysis - SA* Stone et al., 1966): Phân tích cảm xúc tác giả của một đoạn văn bản. Cảm xúc có thể được nhận ra từ sự kết hợp của giọng điệu, sự lựa chọn từ ngữ và phong cách viết. Phân tích cảm xúc được dùng để hiểu sâu sắc hơn về ngữ cảnh của văn bản và áp dụng trong nhiều tình huống khác nhau. Ví dụ, việc đánh giá cảm xúc thích hay không thích về một sản phẩm tiêu dùng dựa trên hàng ngàn bình luận trên mạng xã hội có thể giúp cải thiện chiến dịch truyền thông của sản phẩm đó. Hoặc như một ví dụ khác, bạn có thể đánh giá dự báo thị trường chứng khoán thông qua đánh giá mức độ lạc quan của hàng ngàn người trên một diễn đàn tài chính để giúp hỗ trợ đầu tư. Hoặc những chính trị gia có thể đánh giá phản ứng của hàng trăm ngàn người với mỗi nội dung của họ đưa trên Twitter, Facebook.

- *Phân loại văn bản* (*Text Classification - TC* Soergel, 1985): Đây có thể được coi là một trong những ứng dụng phổ biến nhất. Trong phân loại văn bản, các từ (nghĩa, mối quan hệ các từ, ngữ cảnh) được sử dụng như là các đặc tính để dùng cho các thuật toán nhằm xác định văn bản thuộc về các lớp khác nhau. Ví dụ: Google sử dụng giải thuật phân loại văn bản để phân loại thư điện tử gửi đến có phải là thư rác hay không.

- *Nhận diện chủ thể* (*Entity Recognition - ER* Marsh and Perzanowski, 1998): Xác định và nhận diện thông tin chính (chủ thể) trong văn bản. Một chủ thể có thể là một từ hoặc một chuỗi từ mà liên tục đề cập đến cùng một thứ. Mỗi chủ thể sẽ được xác định và đưa vào các thể loại định trước. Ví dụ như xác định từ "Microsoft" trong văn bản và phân loại nó thành mục "Công ty".

- *Nhận diện chủ đề* (*Topic Modeling - TM* Blei, 2012): Thuật ngữ "chủ đề" có nghĩa là tập các từ "đi cùng với nhau". Những từ này ta sẽ gợi nhớ ra khi nghĩ đến chủ đề cụ thể. Ví dụ như chủ đề "thể thao" thì sẽ gợi nhớ ra các từ như: bóng đá, sân vận động, chạy bộ, bơi lội,....

- *Dịch máy* (*Machine Translation* Madsen, 2009): Dịch máy có thể giúp phiên dịch tự động văn bản từ ngôn ngữ này sang một ngôn ngữ khác. Có nhiều thách thức đối với dịch máy bao gồm như: sự đa dạng về ngôn ngữ, bảng chữ

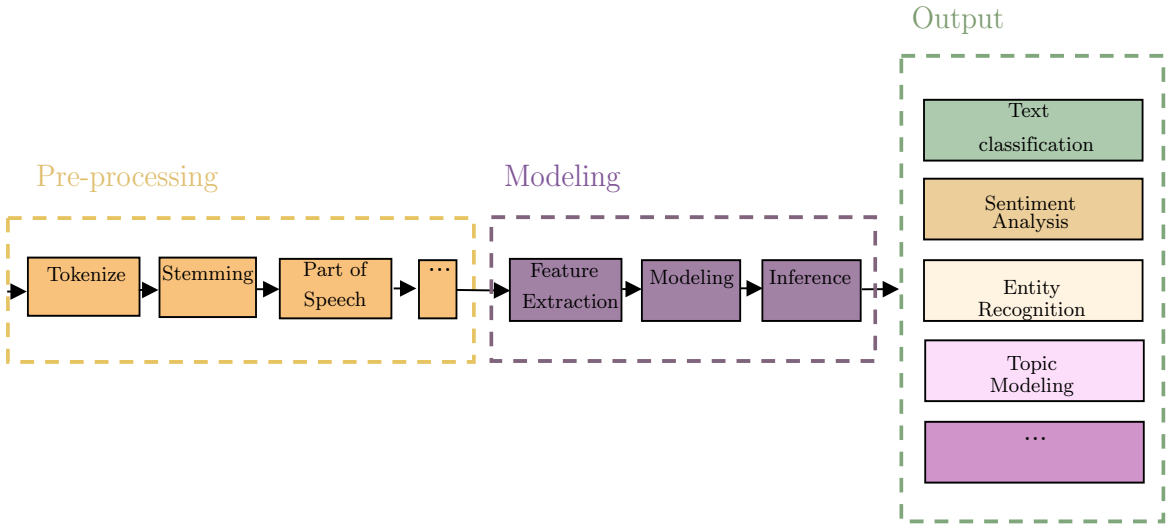
cái và ngữ pháp, khó khăn khi xác định bản dịch có chính xác hay không.

- *Trả lời tin nhắn tự động* (*Chatbot* Bradeško and Mladenić, 2012): trả lời tin nhắn tự động là dễ dàng hiểu được câu hỏi của khách hàng với những từ ngữ tự nhiên hơn mà không cần sự hỗ trợ từ con người.
- *Tự động tóm tắt văn bản* (*Automatic Summarization* Maybury, 1999): Tóm tắt văn bản là nhiệm vụ cô đọng một đoạn văn bản thành phiên bản ngắn hơn, giảm kích thước của văn bản ban đầu đồng thời bảo tồn các yếu tố thông tin chính và ý nghĩa của nội dung. Vì tóm tắt văn bản thủ công là một công việc tốn kém thời gian và thường tốn nhiều công sức, việc tự động hóa tác vụ này đang ngày càng phổ biến và do đó tạo thành động lực mạnh mẽ cho nghiên cứu học thuật.

4.1.1 Mô hình xử lý ngôn ngữ tự nhiên cổ điển

Theo phương pháp cổ điển, các tác vụ xử lý ngôn ngữ tự nhiên gồm có 2 bước chính như mô tả trong Hình 4.1 :

- *Tiền xử lý (Pre-processing)*: quá trình xử lý ban đầu văn bản để loại bỏ các thông tin thừa và chuẩn hóa các thông tin về một định dạng chung.
- *Mô hình hóa (Modeling)*: Đưa ra một mô hình học máy để thực hiện tác vụ mong muốn.

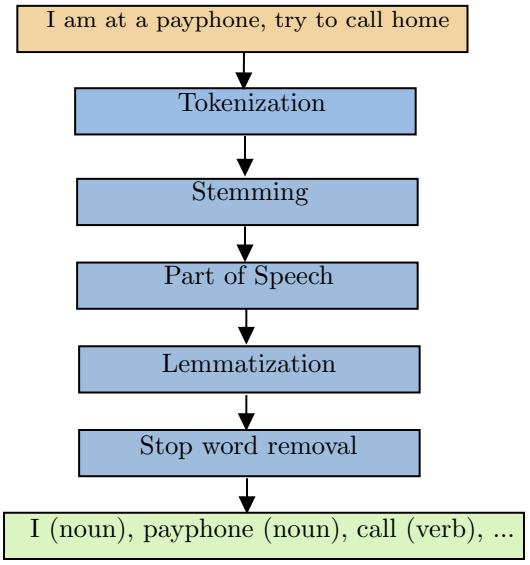


Hình 4.1: Một pipeline phổ biến cho bài toán xử lý ngôn ngữ tự nhiên theo hướng cổ điển

4.1.2 Tiền xử lý

Tiền xử lý (Preprocessing): đây là một công việc quen thuộc đối với *Khoa học dữ liệu (Data Science - DS)*. Đối với dữ liệu số, thông thường ta sẽ áp dụng mộ số quy tắc chuẩn hóa (nhằm giảm sự khác biệt giữa giá trị lớn nhất và nhỏ nhất), thay thế các giá trị không phải là dạng số (cũng như là các giá trị rỗng), phát hiện các giá trị ngoại lệ,...

Đối với dữ liệu văn bản, việc xử lý từ và cụm từ phức tạp hơn các số nguyên và số thực. Trong thực tế, dữ liệu văn bản thường được tiến hành qua vài bước tiền xử lý cơ bản hay còn gọi là *luồng tiền xử lý (Preprocessing Pipeline - PPL)*.



Hình 4.2: Một luồng tiền xử lý phổ biến

Các bước của Preprocessing thường tùy thuộc vào từng dự án và mục đích. Hình 4.2 mô tả các bước thông dụng trong tiền xử lý. Chi tiết các bước gồm có:

- *Tiền xử lý chuỗi thô (Bare String Preprocessing)*: được coi là một trong những bước chính của tiền xử lý. Đây là tác vụ sử dụng các hàm cụ thể của ngôn ngữ lập trình để sửa đổi dữ liệu đầu vào (như thay thế ký tự, tìm và thay thế các mẫu tự).
- *Tách từ (Tokenization)*: là quá trình chuyển một dãy các ký tự thành một dãy các token (token là một dãy các ký tự mang ý nghĩa cụ thể, biểu thị cho một đơn vị ngữ nghĩa trong xử lý ngôn ngữ). Đôi khi token được hiểu là một từ mặc dù cách hiểu này không hoàn toàn chính xác. Ví dụ như trong tiếng Anh các từ thường được phân tách bằng dấu cách, tuy nhiên từ "New York" vẫn chỉ được coi là một từ mặc dù nó có dấu cách ở giữa, do đó chỉ có một token trong trường hợp này. Một ví dụ khác là *I'm* được coi là 2 từ 'I' và 'am' mặc dù không có dấu cách nào. Trong trường hợp này ta có 2 token.
- *Biến đổi từ về dạng gốc (Stemming)*: đây là kỹ thuật dùng để biến đổi 1 từ về dạng gốc (được gọi là stem hoặc root form) bằng cách cực kỳ đơn giản là loại bỏ 1 số ký tự nằm ở cuối từ tạo nên biến thể của từ. Ví dụ như chúng ta thấy

các từ như walked, walking, walks chỉ khác nhau là ở những ký tự cuối cùng, bằng cách bỏ đi các hậu tố -ed, -ing hoặc -s, chúng ta sẽ được từ nguyên gốc là *walk*. Trong tiếng Việt thường không cần làm tác vụ này.

- *Phân loại từ trong câu (Part of Speech - POS)*: POS là việc phân loại các từ trong một câu (danh từ, trạng từ, tính từ hay động từ,...). Việc phân loại từ như thế này sẽ góp phần nắm được thêm ý nghĩa của câu thay vì chỉ xem nó như là tập hợp của các ký tự. Ví dụ như cùng là một từ "can" nhưng nó có thể có nghĩa là "có thể" hoặc nghĩa là "cái lon", như vậy POS có thể giúp máy tính phân biệt được điều này một cách dễ dàng tùy vào nội dung của câu.
- *Từ vựng hóa (Lemmatization)*: Làm trở lại nguyên dạng ban đầu các từ vựng bị biến đổi thể (inflection) hoặc được kết hợp (conjugatetion). Ví dụ như: biến đổi từ ate -> eat. Trong tiếng Việt cũng thường không xử lý bước này.
- *Loại bỏ các từ dừng (Stop word Removal)*: Stop word là những từ xuất hiện nhiều trong ngôn ngữ tự nhiên, tuy nhiên lại không mang nhiều ý nghĩa. Trong tiếng Việt, stop word là những từ như: để, này, kia... Tiếng Anh là những từ như: is, that, this... Mục đích của tác vụ này là loại bỏ những từ không mang lại nhiều ý nghĩa cho mô hình.

4.1.3 Mô hình hóa

Sau khi thực hiện bước tiền xử lý, ta sẽ thực hiện mô hình hóa. Mô hình hóa gồm các bước như sau để thu được kết quả.

- *Rút trích đặt trưng (Extract features)*: Trong lĩnh vực học máy, dữ liệu thông thường được biểu diễn dưới dạng *đặc trưng* (features). Việc rút trích đặc trưng là quá trình hình thành và chọn lọc các đặc trưng quan trọng để biểu diễn dữ liệu đầu vào.
- *Xây dựng mô hình (Modelling)*: mô hình là kết quả của việc áp dụng giải thuật học máy đối với dữ liệu đầu vào. Tùy vào mục đích của bài toán NLP mà ta sẽ lựa chọn giải thuật học máy phù hợp.
- *Suy diễn (Inference)*: dùng mô hình xây dựng được để áp dụng vào các dữ liệu mới.

4.2 Trích xuất đặc trưng - Feature Extraction

Vào năm 2008, nhóm nghiên cứu xử lý ngôn ngữ tự nhiên gồm giáo sư Christopher Manning và những nhà nghiên cứu khác đã viết tác phẩm mang tên *"Mở đầu về trích xuất thông tin"* (Introduction to Information Retrieval) (Manning et al., 2008) giải thích về những kỹ thuật khai thác thông tin từ các nguồn dữ liệu, đặc biệt là dữ liệu dạng text. Ta có thể dùng thuật ngữ trích xuất thông tin và trích xuất feature thay cho nhau.

Dưới đây là 2 phương pháp cơ bản để trích xuất feature được mô tả trong quyển sách nổi tiếng trên:

- Vector hóa bằng kỹ thuật đếm (Count-Vectorization - (Bag-of-words) - BOW)
- Sử dụng trọng số TF-IDF (Term Frequency - Inverse Document Frequency)

4.2.1 Phương pháp túi từ

Bằng *phương pháp túi từ* (*Bag-of-Words* Z. S. Harris, 1954), ta sẽ đưa các từ trong toàn thể văn bản thành các token và ghi nhận lại sự xuất hiện của các token này trong từng văn bản.

Cùng xem ví dụ về 3 câu sau:

1. "Hôm qua tôi học lập trình"
2. "Hôm nay tôi cũng học lập trình"
3. "Ngày mai tôi không học lập trình"

Thông qua bước tokenization ta biến đổi được thành các câu:

1. "Hôm_qua tôi học lập_trình"
2. "Hôm_nay tôi cũng học lập_trình"
3. "Ngày_mai tôi không học lập_trình"

Qua bước này "hôm qua", "hôm nay", "ngày mai" và "lập trình" được biến đổi thành từ "hôm_qua", "hôm_nay", "ngày_mai" và "lập_trình".

Ta sẽ xem mỗi câu là một văn bản riêng biệt và ta xây dựng một danh sách tất cả các từ từ 3 câu trên. Ta sẽ được các từ như sau:

- Hôm_qua
- Hôm_nay
- Ngày_mai
- tôi
- cũng
- không
- học
- lập_trình

Bước tiếp theo ta sẽ xây dựng vector. Vector sẽ chuyển các văn bản thành input cho các giải thuật học máy.

Với câu đầu tiên "Hôm_qua tôi học lập_trình", dựa vào tần suất xuất hiện từ (các từ này là duy nhất trong tập) trong câu ta xây dựng được vector như sau:

- 'Hôm_qua'= 1
- 'Hôm_nay' = 0
- 'Ngày_mai' = 0
- 'tôi' = 1
- 'cũng' = 0
- 'không' = 0
- 'học' = 1
- 'lập_trình' = 1

=> [1, 0, 0, 1, 0, 0, 1, 1]

Các câu còn lại ta sinh được vector như sau:

- Hôm_nay tôi cũng học lập_trình => [0, 1, 0, 1, 1, 0, 1, 1]

- Ngày_mai tôi không học lập_trình => [0, 0, 1, 1, 0, 1, 1, 1]

Dưới đây là ma trận ta thu được từ phương pháp BOW (Bảng 4.1):

Bảng 4.1: Bảng ma trận vector thu được từ phương pháp BOW

Câu	Hôm_qua	Hôm_nay	Ngày_mai	tôi	cũng	không	học	lập_trình
Câu 1	1	0	0	1	0	0	1	1
Câu 2	0	1	0	1	1	0	1	1
Câu 3	0	0	1	1	0	1	1	1

Số cột ở Bảng 4.1 phụ thuộc vào số từ khác nhau trong tập hợp các câu. Nếu như số lượng câu đủ lớn, ta sẽ dùng thuật ngữ *kho ngữ liệu* (*corpus* Wołk and Marasek, 2014) và mỗi từ trong kho ngữ liệu sẽ được gọi là *từ vựng (vocabulary)*.

4.2.2 Phương pháp TF-IDF

Tổng quan phương pháp TF-IDF

TF-IDF là từ viết tắt của thuật ngữ tiếng Anh "term frequency – inverse document frequency", TF-IDF là trọng số của một từ trong văn bản thu được qua thống kê thể hiện mức độ quan trọng của từ này trong một văn bản, mà bản thân văn bản đang xét nằm trong một tập hợp các văn bản (trích *"Khai thác những bộ dữ liệu lớn"* (Mining of Massive Datasets) của Anand Rajaraman và Jeffrey Ullman (Leskovec et al., 2014). Ưu điểm của TF-IDF so với phương pháp túi từ là TF-IDF có thể biểu diễn từ với các trọng số khác nhau dựa vào độ quan trọng của từ đó trong câu.

Ví dụ ta có một câu như sau:

"Today is a beautiful day"

Cảm xúc trong câu này là tích cực và được thể hiện qua tính từ *beautiful*. Do vậy, ta thấy từ *beatiful* có tầm quan trọng lớn trong việc hiểu được cảm xúc của tác giả. Với phương pháp túi từ, tất cả các từ đều có cùng tần suất xuất hiện và bằng 1. Tầm quan trọng của từng từ được thể hiện thông qua tần suất. Như vậy, các từ đều quan trọng như nhau và ta không thể nhấn mạnh được từ *beautiful*. Nhưng nếu sử dụng phương pháp TF-IDF cùng với một lượng lớn dữ liệu thống kê, hệ thống học máy có thể nhận ra tầm quan trọng của từ *beautiful*.

Mặt khác, trong một tài liệu có 200 từ, từ 'a', 'the' xuất hiện rất nhiều lần (20 lần từ 'a', 15 lần từ 'the'). Những từ này xuất hiện nhiều lần không có nghĩa là chúng quan trọng. TF-IDF sinh ra để giải quyết bài toán như vậy.

Tổng quát, phương pháp TF-IDF có các ưu và khuyết điểm sau.

Ưu điểm:

- Dễ tính toán
- Đây là cách đơn giản để trích xuất các mô tả trong văn bản
- Dễ dàng tính toán sự giống nhau giữa 2 văn bản

Nhược điểm:

- TF-IDF dựa trên phương pháp túi từ, vì thế nó không thể thể hiện tầm quan trọng của vị trí từ trong văn bản, ngữ nghĩa, sự tương quan giữa các từ trong các văn bản khác nhau

Cách tính của phương pháp TF-IDF

TF-IDF gồm 2 trọng số TF và IDF.

TF - Term Frequency : dùng để ước lượng tần xuất xuất hiện của từ trong văn bản. Tuy nhiên với mỗi văn bản thì có độ dài khác nhau, vì thế số lần xuất hiện của từ có thể nhiều hơn. Vì vậy số lần xuất hiện của từ sẽ được chia độ dài của văn bản (tổng số từ trong văn bản đó)

$$TF(t,d) = \frac{\text{(số lần từ t xuất hiện trong văn bản d)}}{\text{(tổng số từ trong văn bản d)}}$$

IDF - Inverse Document Frequency: dùng để ước lượng mức độ quan trọng của từ đó như thế nào . Khi tính tần số xuất hiện TF thì các từ đều được coi là quan trọng như nhau. Tuy nhiên có một số từ thường được được sử dụng nhiều nhưng không quan trọng để thể hiện ý nghĩa của đoạn văn, ví dụ:

- Từ nối: và, nhưng, tuy nhiên, vì thế, vì vậy,...
- Giới từ: ở, trong, trên,...
- Từ chỉ định: ấy, đó, nhỉ,...

Vì vậy ta cần giảm đi mức độ quan trọng của những từ đó bằng cách sử dụng IDF

$$IDF(t, D) = log\left(\frac{\text{Tổng số văn bản trong tập mẫu D}}{\text{Số văn bản có chứa từ}}\right)$$

Từ đó ta có công thức TF-IDF

$$\text{TF-IDF}(t, d, D) = TF(t, d) \times IDF(t, D)$$

Mỗi tài liệu sẽ được biểu diễn bằng một vector TF-IDF, vector này sẽ số chiều là K với K độ dài tất cả các từ có trong tất cả các văn bản của tập dữ liệu.

Để hiểu rõ hơn về cách xây dựng vector TF-IDF, chúng ta cùng theo dõi ví dụ sau. Giả sử ban đầu ta có ba văn bản như sau:

- Document 1: "Hôm_qua tôi học lập_trình"
- Document 2: "Hôm_nay tôi cũng học lập_trình"
- Document 3: "Ngày_mai tôi không học lập_trình"

Đầu tiên ta sẽ xây dựng bảng từ vựng trong kho ngữ liệu. Bảng 4.2 là bảng ta thu được khi xây dựng bảng từ vựng.

Bảng 4.2: Bảng từ vựng trong kho ngữ liệu

Word	Count
Hôm_qua	1
Hôm_nay	1
Ngày_mai	1
tôi	3
cũng	1
không	1
học	3
lập_trình	3

Tiếp theo ta sẽ tính TF cho mỗi từ ứng với mỗi văn bản trong kho ngữ liệu. Bảng 4.3 là kết quả tính giá trị TF cho mỗi từ vựng ứng với mỗi văn bản.

Bước kế tiếp ta cần tính giá trị IDF cho mỗi từ vựng. Bảng 4.4 liệt kê kết quả tính IDF cho mỗi từ vựng.

Bảng 4.3: Bảng tính TF cho các tài liệu

Words/Document	Document 1	Document 2	Document 3
Hôm_qua	0.25	0	0
Hôm_nay	0	0.2	0
Ngày_mai	0	0	0.2
tôi	0.25	0.2	0.2
cũng	0	0.2	0
không	0	0	0.2
học	0.25	0.2	0.2
lập_trình	0.25	0.2	0.2

Bảng 4.4: Bảng tính IDF cho các văn bản

Words	IDF value
Hôm_qua	log(3/1) = 0.48
Hôm_nay	log(3/1) = 0.48
Ngày_mai	log(3/1) = 0.48
tôi	log(3/3) = 0
cũng	log(3/1) = 0.48
không	log(3/1) = 0.48
học	log(3/3) = 0
lập_trình	log(3/3) = 0

Cuối cùng ta dựa vào giá trị TF và IDF ta sẽ tính được vector TF-IDF cho mỗi từ vựng dựa theo công thức TF-IDF = $TF \times IDF$. Bảng 4.5 mô tả ma trận kết quả thu được.

Bảng 4.5: Bảng tính TF-IDF

	Hôm_qua	Hôm_nay	Ngày_mai	tôi	cũng	không	học	lập_trình
Document 1	0.12	0	0	0	0	0	0	0
Document 2	0	0.1	0	0	0.1	0	0	0
Document 3	0	0	0.1	0	0	0.1	0	0

Sau khi tính TF-IDF, mỗi document đã được vector hóa bằng phương pháp này (số chiều của vector chính là số từ ta trong tất cả document, số chiều bằng 8) gồm:

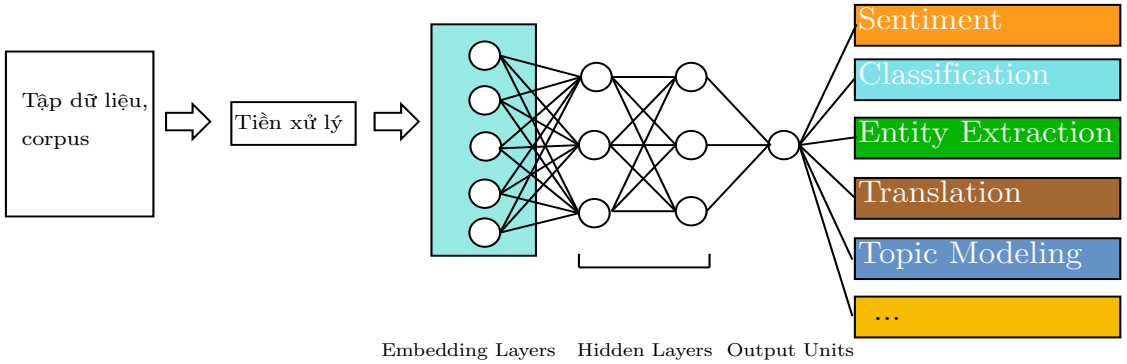
- "Hôm_qua tôi học lập_trình" => [0.12, 0, 0,0, 0, 0, 0, 0]

- "Hôm_nay tôi cũng học lập_trình" => [0, 0.1, 0, 0, 0.1, 0, 0, 0]
- "Ngày_mai tôi không học lập_trình" [0, 0, 0.1, 0, 0, 0.1, 0, 0]

Dựa vào công thức TF-IDF ta cũng nhận ra một hạn chế của phương pháp này là chúng không biểu diễn được quan hệ giữa các từ trong văn bản. Ví dụ như khi chúng ta biểu diễn câu "Học lập_trình hôm_qua tôi" bằng phương pháp TF-IDF sẽ cho ra cùng vector [0.12, 0, 0,0, 0, 0, 0, 0] với câu "Hôm_qua tôi học lập_trình".

4.3 Mô hình xử lý ngôn ngữ tự nhiên bằng mạng học sâu

Với sự trỗi dậy gần đây của các mô hình học sâu, các hướng tiếp cận cổ điển của việc xử lý ngôn ngữ tự nhiên như Hình 4.1 đã dần được thay thế bằng các *mô hình học sâu* (*Deep Learning - DL* Schmidhuber, 2015). Hiện nay, đã có rất nhiều mô hình học sâu cùng các kĩ thuật phức tạp được đề ra nhằm để giải quyết các bài toán khác nhau, nhưng về căn bản thì việc sử dụng mô hình học sâu để giải quyết một bài toán xử lí ngôn ngữ tự nhiên có thể được mô tả như Hình 4.3.



Hình 4.3: Mô hình Deep Learning được sử dụng cho NLP

Từ Hình 4.3, ta có thể thấy, để giải quyết một bài toán xử lí ngôn ngữ tự nhiên bằng kiến trúc mạng học sâu thì ta sẽ thực hiên một số bước cơ bản như sau:

- Cũng như mô hình cổ điển, dữ liệu được dùng để huấn luyện cần phải qua bước tiền xử lý như tokenization, biến đổi từ về dạng gốc, từ vựng hóa, loại bỏ các từ vựng...
- Sau đó, do mạng học sâu chỉ tương tác trên các ma trận, vector và con số như đã trình bày ở các chương trước nên sẽ cần có một bước để chuyển các từ này thành các vector. Đó chính là mục tiêu chính của tầng *nhúng* (Embedding Layer).
- Phần chính của hệ thống sẽ là một kiến trúc học sâu. Tùy thuộc vào bài toán

cụ thể như Sentiment Analysis, Text Classification hay Topic Modeling mà ta sẽ có kiến trúc học sâu phù hợp.

Như vậy khi ta áp dụng mô hình trên vào một bài toán nhất định, ta có thể thấy là một cách rất tự nhiên rằng sau quá trình huấn luyện kết thúc, mạng học sâu có thể tự động học, sau đó đã rút trích được các đặc trưng cần thiết và biến đổi chúng thành các trọng số ở các tầng ẩn. Đây cũng chính là lý do giải thích cho ta thấy được rằng kiến trúc mạng học sâu có thể được sử dụng để thay thế cho mô hình xử lý ngôn ngữ tự nhiên cổ điển.

Quay lại với tầng nhúng, có rất nhiều cách để có thể biến đổi dữ liệu đầu vào thành các vector. Với 3 tài liệu ở ví dụ đầu tiên đã nêu ra, ta sẽ có một tập từ điển có 8 từ được sắp xếp theo bảng chữ cái. Tức là, mỗi tài liệu ứng với 1 vector 1 x 8 (do có 8 từ), giá trị của các phần tử trong vector này chính là số lần xuất hiện của từ đó trong tài liệu. Ví dụ, với tài liệu là "Hôm_qua tôi đi học" sẽ tương đương với vector là [0 1 1 0 1 0 0 1].

Tuy nhiên, nhờ vào sức mạnh tính toán của mạng học sâu, ta vẫn có cách biểu điễn để giúp cho tập tài liệu của ta thể hiện nhiều thông tin hơn. Cụ thể hơn là thay vì biểu diễn cả tài liệu chỉ thông qua một vector thì ta sẽ biểu diễn từng từ trong tài liệu thành từng vector. Điều này có nghĩa với cách tiếp cận này, sau khi ta áp dung cho tập tài liệu "Hôm_qua tôi đi học" thì ta sẽ tạo ra được một ma trận 4 x 8 (do có 4 từ, mỗi từ được thể hiện bằng một vector có kích thước là 8).

Quá trình biến đổi từ thành các vector này còn được biết đến với một cái tên khác khá phổ biến là *nhúng từ* (Word Embedding). Trong mục 4.4, ta sẽ tìm hiểu kĩ hơn về các kĩ thuật được sử dụng cho Word Embedding một cách chi tiết để sử dụng chúng hiệu quả hơn.

4.4 Các kỹ thuật biểu diễn từ

Hầu hết những giải thuật học máy, đặc biệt là những giải thuật sử dụng mạng nơ-ron như MLP, mạng học sâu không làm việc hiệu quả trên dữ liệu văn bản như với các dạng dữ liệu số khác. *Nhúng từ* (*word embedding*) là một trong những là kĩ thuật được sử dụng trong xử lý ngôn ngữ tự nhiên nhằm số hoá một từ ở dạng văn bản

mà con người hiểu, sang dạng cho phép máy tính có thể tính toán và xử lý được. Sự ra đời của kỹ thuật nhúng từ bắt nguồn từ nhu cầu muốn giảm chiều biểu diễn cho từng từ trong khi làm việc với chúng. Người nghiên cứu đóng góp cho công trình này là Yoshua Bengio, thông qua bài báo *"Mô hình nơ-ron ngôn ngữ mang tính xác suất"* (Neural Probabilistic Language Models) (Bengio et al., 2001) của ông vào năm 2000.

Kỹ thuật nhúng từ được sử dụng rộng rãi và phổ biến trong những dự án xử lý ngôn ngữ tự nhiên không chỉ vì khả năng biểu diễn văn bản dưới dạng số, mà kĩ thuật này còn cho phép nhúng (embedding) ngữ nghĩa của từ trong văn bản vào định dạng số mà máy tính có thể hiểu và xử lý hiệu quả.

Từ khi khái niệm nhúng từ ra đời, đã có rất nhiều giải thuật được trình bày như word2vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014), ELMO (Peters et al., 2018), BERT (Devlin et al., 2019), v.v. Tuy nhiên một giải thuật nhúng từ cần phải thoả mãn 2 yêu cầu dưới đây:

- Tồn tại một và chỉ một cách biểu diễn cho một từ. Nghĩa là hai từ khác nhau sẽ được số hoá thành hai dạng biểu diễn khác nhau.
- Hai từ có ngữ nghĩa tương tự nhau, khoảng cách cũng sẽ gần nhau sau khi được biểu diễn sang không gian mới.

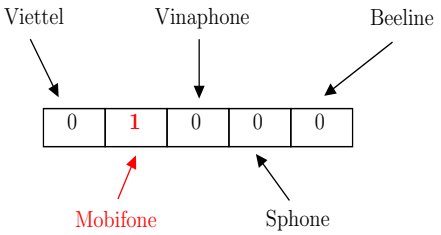
4.4.1 Biểu diễn từ bằng one-hot vector

Trước khi kỹ thuật nhúng từ ra đời, kĩ thuật được sử dụng phổ biến để số hoá một từ là sử dụng *one-hot vector*. Kĩ thuật này được áp dụng trong các mô hình học máy cổ điển trong giai đoạn tiền xử lý, nhằm biến đổi những dữ diệu dạng categorical sang numerical, từ đó giúp máy tính có thể tính toán và xử lý được hiệu quả hơn.

One-hot vector là một vector chứa dữ liệu nhị phân (0 và 1), tuy nhiên chỉ có một chiều trong vector được kích hoạt mang giá trị 1, toàn bộ những chiều còn lại đều mang giá trị 0. One-hot encoding là kĩ thuật biểu diễn một tập các giá trị rời rạc sang một tập one-hot vector. Để hiểu rõ hơn, xem xét ví dụ dưới đây.

Danh sách nhà mạng viễn thông ở Việt Nam là một tập hợp các giá trị rời rạc bao gồm: Viettel, Mobifone, Vinaphone, Sphone, Beeline. Sử dụng kĩ thuật one-hot

vector, chúng ta có thể biểu diễn giá trị Mobifone bằng một vector trong Hình 4.4.



Hình 4.4: Biểu diễn Mobifone bằng one-hot vector.

Vector này có 5 chiều tương đương với 5 nhà mạng chúng ta có. Trong đó tại chiều thứ 2, tương ứng với vị trí của Mobifone trong danh sách nhà mạng, mang giá trị 1, những chiều còn lại mạng giá trị không. Tương tự, chúng ta có thể biểu diễn các nhà mạng khác bằng các vector one-hot được trình bày trong Bảng 4.6.

	Viettel	Mobifone	Vinaphone	SPhone	Beeline
Viettel	1	0	0	0	0
Mobifone	0	1	0	0	0
Vinaphone	0	0	1	0	0
SPhone	0	0	0	1	0
Beeline	0	0	0	0	1

Bảng 4.6: Sử dụng one-hot vector để biểu diễn các nhà mạng ở Việt Nam

Chúng ta có thể dễ dàng nhận thấy, tập hợp từ trong văn bản cũng thuộc miền giá trị rời rạc. Do đó, kĩ thuật one-hot encoding có thể sử dụng nhằm biểu diễn một từ sang dạng one-hot vector.

Xem xét các câu sau đây (sau khi tokenize):

- Hôm_qua tôi học lập_trình.
- Hôm_nay tôi cũng học lập_trình
- Ngày_mai tôi không học lập_trình

Tập từ điển của chúng ta có 8 từ: "Hôm_qua", "Hôm_nay", "Ngày_mai", "Tôi", "Học", "Lập_trình" "Cũng", "Không".

Để biểu diễn sang dạng số, chúng ta sử dụng one-hot vector có 8 chiều. Số 8 tương ưng ứng với số từ trong từ điển. Mở rộng ra, khi từ điển chúng ta có n từ, chúng ta

biến đổi mỗi từ sang one-hot vector có n chiều. Đa phần các chiều của vector mang giá trị 0, chỉ duy nhất ở chiều tương ứng với vị trí của từ trong từ điển được kích hoạt mạng giá trị 1. Bảng 4.7 thể hiện việc chuyển đổi từ sang one-hot vector.

	Hôm_qua	Hôm_nay	Ngày_mai	Tôi	Học	Lập_trình	Cũng	Không
Hôm_qua	1	0	0	0	0	0	0	0
Hôm_nay	0	1	0	0	0	0	0	0
Ngày_mai	0	0	1	0	0	0	0	0
Tôi	0	0	0	1	0	0	0	0
Học	0	0	0	0	1	0	0	0
Lập_trình	0	0	0	0	0	1	0	0
Cũng	0	0	0	0	0	0	1	0
Không	0	0	0	0	0	0	0	1

Bảng 4.7: Sử dụng one-hot vector để biểu diễn các từ trong từ điển.

Dựa vào kĩ thuật one-hot encoding, chúng ta có thể biểu diễn bất kì từ nào sang dạng số mà máy tính có thể xử lý. Rất nhiều mô hình xử lý ngôn ngữ tự nhiên và học máy đã sử dụng kĩ thuật này và mang lại hiệu quả khả quan, đặc biệt khi số lượng từ trong từ điển tương đối nhỏ. Tuy kĩ thuật này đã đảm bảo được yêu cầu biểu diễn hai từ khác nhau với hai vector khác nhau, nó vẫn mang nhiều khuyết điểm cần được cải tiến:

- Giới hạn khả năng tính toán của máy tính: Hầu hết các chiều của one-hot vector mang giá trị 0, và nhiều mô hình học máy không làm việc hiệu quả trên vector có số chiều lớn (high dimensional vector) và "thưa" (sparse vector).
- Sự quá khớp xảy ra khi số lượng từ trong từ điển tăng: Với kĩ thuật này, mỗi khi chúng ta gia tăng số lượng từ trong từ điển lên n, số chiều của one-hot vector cũng tăng tương ứng vì số chiều của vector tương ứng với số lượng từ ta có trong từ điển. Trong ví dụ trên chúng ta chỉ có 8 từ trong từ điển, nhưng trong thực tế số từ trong từ điển là rất lớn, có thể đến vài ngàn, chục ngàn từ. Càng nhiều từ, càng nhiều *thông số* (*parameter*) mà mô hình học máy cần phải học. Càng nhiều thông số, chúng ta lại cần càng nhiều dữ liệu huấn luyện để xây dựng mô hình tốt, và không bị quá khớp.
- Thiếu khả năng tổng quát hoá: Qua quá trình tiến hoá của con người, ngôn ngữ được sinh ra và phát triển theo thời gian. Khi nhìn vào một từ, chúng ta

không chỉ biết ý nghĩa của mỗi từ đó mà còn hiểu được sự liên kết với những từ khác, và có khả năng tổng quát hoá ý nghĩa của một nhóm các từ liên quan. Nhờ khả năng tổng quá hoá, chúng ta có thể rút ngắn thời gian học một kiến thức mới. Ví dụ, chúng ta biết được mèo biết leo cây và ăn thịt, trong khi đó hổ và báo cũng tương tự như mèo (tổng quá hoá), nhờ vậy chúng ta có thể dự đoán được hổ vào báo cũng có thể leo cây và ăn thịt.

Khi xây dựng một mô hình học máy, chúng ta muốn mô hình cũng có khả năng tổng quát hoá như chúng ta nhằm rút ngắn thời gian huấn luyện. Quay lại ví dụ trên, khi mô hình đang được huấn luyện trên dữ liệu đầu vào là "mèo", chúng ta có dữ liệu đầu vào mới là "hổ". Nếu hiểu được "mèo" và "hổ" tương tự nhau, mô hình không cần học "hổ" từ đầu, thay vào đó mô hình có thể sử dụng lại những thông số được huấn luyện với đầu vào là "mèo". Từ đó rút ngắn thời gian và lượng dữ liệu để huấn luyện mô hình.

Với kĩ thuật one-hot encoding, mỗi từ được biểu diễn thành một vector riêng lẽ, và khoảng cách giữa các vector đều bằng nhau và bằng $\sqrt{(}2)$. Do đó, mô hình không thể sử dụng lại những "kiến thức" đã được huấn luyện cho những từ liên quan với nhau về mặt ngữ nghĩa.

4.4.2 Kỹ thuật Auto-Encoder để thu giảm số chiều

Giới thiệu

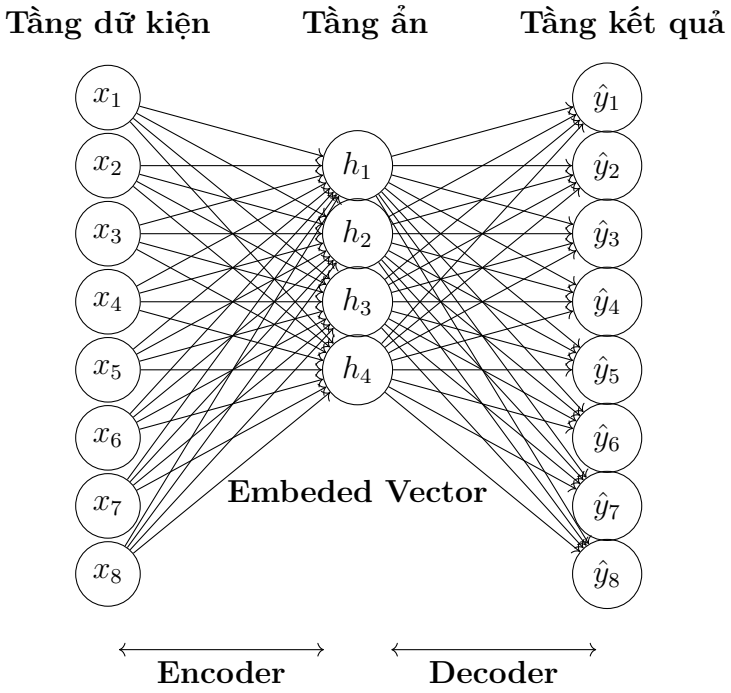
Ý tưởng về auto-encoder đã được các nhà nghiên cứu lỗi lạc đề xuất về mạng nơ-ron đề xuất từ cuối thế kỉ 20, đại diện là LeCun (LeCun et al., 1989), Hinton và Zemel (Hinton and Zemel, 1994), với ý tưởng là mô hình mạng nơ-ron, được huấn luyện dựa trên cơ chế học không giám sát, nhằm biểu diễn dữ liệu đầu vào ở dạng *vector nhiều chiều* (*high-dimensonal vector*) bằng một vector có số chiều nhỏ hơn (*low-demensial vector*). Mô hình này được sử dụng trong *kĩ thuật dữ liệu* (*data engineering*) cho phép thu giảm số chiều của dữ liệu đầu vào, nhằm giúp trực quan hoá dữ liệu; tăng khả năng tính toán của máy tính, giảm số lượng thông số của mô hình từ đó rút ngắn thời gian huấn luyện;...

Mô hình Auto-Encoder hoạt động dựa trên cơ chế *compress* và *reconstruct*. Theo đó, mô hình phải học cách biểu diễn dữ liệu đầu vào với một lượng thông tin ít hơn (compress), sao cho nó vẫn có thể tái tạo lại được dữ liệu đầu vào dựa trên lượng thông tin đó (reconstruct). Một cách hiểu đơn giản là mô hình hoạt động tương tự như cơ chế nén và giải nén file của WinRar. Mô hình được phân loại trong mô hình học không giám sát bởi vì chúng ta không cần chuẩn bị nhãn để huấn luyện mô hình. Nó sử dụng chính dữ liệu đầu vào làm nhãn, vì mục đính của mô hình là học để tái tạo lại dữ liệu ban đầu.

Kiến trúc mô hình Auto-Encoder

Kiến trúc mô hình Auto-Encoder là một mạng nơ-ron 3 lớp như Hình 4.5. Trong đó tầng ẩn có số nút nhỏ hơn rất nhiều so với số nút ở tầng dữ kiện và tầng kết quả. Auto-Encoder bao gồm các thành phần quan trọng sau:

- *Encoder*: là một ma trận $N \times k$, trong đó N là số chiều của vector đầu vào, k là số nút của tầng ẩn. Để mô hình học được cách biểu vector đầu vào bằng một vector có số chiều nhỏ hơn, mô hình được thiết kế với số nút ở tầng ẩn nhỏ hơn so với số nút ở tầng dữ kiện và kết quả ($k << N$). Ma trận làm nhiệm vụ compress dữ liệu đầu vào ở không gian nhiều chiều N sang không gian có số chiều nhỏ hơn k. Dữ liệu đầu vào sau khi đi qua ma trận Encoder, chúng ta nhận được một vector có số chiều là k, được gọi là encoded vector.



Hình 4.5: Mô hình Auto-Encoder sử dụng mạng nơ-ron 3 lớp.

- *Decoder*: ngược với với Encoder, Decoder là một ma trận $k \times N$ làm nhiệm vụ reconstruct dữ liệu đầu vào dựa trên encoded vector sinh ra từ bộ Encoder. Ở đây vector có số chiều là k sẽ được biến đổi để trở về vector ban đầu có số chiều là N. Đầu ra của bộ Decoder là một vector gần giống với vector đầu vào. Mô hình được huấn luyện để giảm khoảng cách giữa vector đầu vào và vector đầu ra nhất có thể. Mục tiêu này được thể hiện thông qua một hàm mất mát chính là khoảng cách Euclidean giữa vector đầu vào và vector đầu ra.

Quá trình huấn luyện mô hình Auto-Encoder

Xem xét các câu sau đây (sau khi tokenize):

- Hôm_qua tôi học lập_trình
- Hôm_nay tôi cũng học lập_trình
- Ngày_mai tôi không học lập_trình

Tập từ điển của chúng ta có 8 từ: "Hôm_qua", "Hôm_nay", "Ngày_mai", "Tôi", "Học", "Lập_trình", "Cũng", "Không".

Ma trận Encoder và Decoder của mô hình Auto-Ecoder trong Hình 4.5, được khởi tạo lần lượt như sau:

Ma trận Encoder 8 x 4: $\begin{bmatrix} 3 & 2 & 1 & 5 \\ 4 & 5 & 7 & 8 \\ 2 & 1 & 2 & 4 \\ 4 & 2 & 1 & 2 \\ 1 & 3 & 1 & 3 \\ 2 & 4 & 2 & 1 \\ 4 & 5 & 2 & 7 \\ 6 & 4 & 4 & 5 \end{bmatrix}$

Ma trận Decoder 4 x 8: $\begin{bmatrix} 4 & 5 & 1 & 2 & 5 & 1 & 5 & 6 \\ 5 & 6 & 8 & 1 & 9 & 0 & 2 & 4 \\ 3 & 2 & 1 & 4 & 2 & 5 & 7 & 9 \\ 7 & 5 & 7 & 9 & 2 & 1 & 3 & 1 \end{bmatrix}$

Để huấn luyện mô hình, chúng ta lần lượt lặp qua từng từ (token) trong tập *corpus (kho ngữ liệu)*. Tại mỗi bước, tương ứng với một từ, vector đầu vào và đầu ra mong đợi của mô hình chính là one-hot vector của từ đó. Quá trình huấn luyện mô hình trong mỗi bước chi tiết như sau (giả sử chúng ta bắt đầu với "Hôm_qua"):

1. Nhân vector one-hot của "Hôm_qua" với ma trận Encoder kích thước 8 x 4. Sau khi nhân, chúng ta thu được một encoded vector của "Hôm_qua". Vector này có số chiều là 4. Do tính chất đặc biệt của one-hot vector (chỉ có duy nhất một vị trí có giá trị là 1), encoded vector này cũng chính là hàng thứ 1 của ma trận Encoder, tương ứng với ví trí của "Hôm_qua" trong từ điển. Phía dưới, mô tả quá trình khi nhân one-hot vector của "Hôm_qua" với ma trận

Encoder:

$$\begin{bmatrix}\mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix} \times \begin{bmatrix}\mathbf{3} & \mathbf{2} & \mathbf{1} & \mathbf{5} \\ 4 & 5 & 7 & 8 \\ 2 & 1 & 2 & 4 \\ 4 & 2 & 1 & 2 \\ 1 & 3 & 1 & 3 \\ 2 & 4 & 2 & 1 \\ 4 & 5 & 2 & 7 \\ 6 & 4 & 4 & 5\end{bmatrix} = \begin{bmatrix}3 & 2 & 1 & 5\end{bmatrix}$$

2. Tiếp tục nhân encoded vector với ma trận Decoder, ta thu được vector đầu ra, có số chiều bằng với vector đầu vào:

$$\begin{bmatrix}3 & 2 & 1 & 5\end{bmatrix} \times \begin{bmatrix}4 & 5 & 1 & 2 & 5 & 1 & 5 & 6 \\ 5 & 6 & 8 & 1 & 9 & 0 & 2 & 4 \\ 3 & 2 & 1 & 4 & 2 & 5 & 7 & 9 \\ 7 & 5 & 7 & 9 & 2 & 1 & 3 & 1\end{bmatrix} = \begin{bmatrix}60 & 54 & 55 & 57 & 45 & 13 & 41 & 40\end{bmatrix}$$

3. Mất mát của mô hình là khoảng cách Euclidean giữa vector one-hot của "Hôm_qua" với vector đầu ra:

$$Loss = \sqrt{(1-60)^2 + (0-54)^2 + ... + (0-41)^2 + (0-40)^2} \approx 134.71$$

4. Sử dụng kĩ thuật lan truyền ngược, mô hình điều chỉnh thông số của hai ma trận Encoder và Decoder qua mỗi bước.

Tương tự như vậy, chúng ta tiếp tục lặp qua các từ còn lại trong kho ngữ liệu. Trong những bước huấn luyện đầu, vector đầu ra từ Decoder sẽ khác với vector đầu vào, do độ lỗi mô hình lúc này khá lớn. Tuy nhiên, sau quá trình huấn luyện lặp đi lặp lại trên kho ngữ liệu, mô hình sẽ hội tụ. Lúc này vector đầu ra sẽ không giống nhưng xấp xỉ với vector one-hot đầu vào.

Ví dụ, sau quá trình huấn luyện:

Ứng với vector one-hot đầu vào của từ "Hôm_nay"

$$\begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$$

Vector đầu ra của mô hình Auto-Encoder sẽ là

$$\begin{bmatrix}0.001 & 0.99 & 0.0012 & 0.00023 & 0.0004 & 0.0011 & 0.0001 & 0.00005\end{bmatrix}$$

Giả sử sau khi huấn luyện xong, ma trận Encoder và Decoder lần lượt như sau:

Ma trận Encoder 8 x 4:
$$\begin{bmatrix}0.2 & 0.1 & 0.5 & 0.2\\ 0.6 & 0.5 & 0.3 & 0.3\\ 0.4 & 0.2 & 0.8 & 0.9\\ 0.1 & 0.7 & 0.5 & 0.3\\ 0.8 & 0.4 & 0.9 & 0.1\\ 0.5 & 0.1 & 0.1 & 0.3\\ 0.1 & 0.6 & 0.1 & 0.7\\ 0.3 & 0.2 & 0.4 & 0.5\end{bmatrix}$$

Ma trận Decoder 4 x 8:
$$\begin{bmatrix}0.3 & 0.3 & 0.6 & 0.1 & 0.7 & 0.7 & 0.3 & 0.8\\ 0.2 & 0.5 & 0.4 & 0.2 & 0.7 & 0.1 & 0.5 & 0.2\\ 0.1 & 0.5 & 0.3 & 0.9 & 0.9 & 0.2 & 0.3 & 0.1\\ 0.3 & 0.5 & 0.2 & 0.5 & 0.1 & 0.2 & 0.6 & 0.1\end{bmatrix}$$

Do tính chất đã đề cập trong bước một trong mỗi bước khi huấn luyện mô hình, chúng ta có thể thấy rằng ma trận Encoder chứa toàn bộ encoded vector của các từ trong từ điển. Vector ở hàng thứ i chính là encoded vector của từ thứ i trong từ điển. Ví dụ, encoded vector của từ "Ngày_mai" tương ứng với hàng thứ 3, trong ma trận Encoder là:

$\begin{bmatrix}0.4 & 0.2 & 0.8 & 0.9\end{bmatrix}$.

Khả năng tổng quát hoá của mô hình Auto-Encoder

Do kiến trúc đặc biệt của mô hình Auto-Encoder và định nghĩa của hàm mất mát, mô hình bắt buộc phải học được cách biểu diễn dữ liệu đầu vào bằng một lượng thông tin ít hơn, súc tích hơn, theo đó tổng quát hơn, nhưng vẫn phải giữ lại được ý nghĩa cốt lõi nhằm tái tạo lại dữ liệu đầu vào. Hoạt động này giống như khi ta

đang tóm tắt nội dung của một văn bản gồm nhiều trang chỉ bằng một đoạn văn ngắn vậy. Để làm được điều này, chúng ta phải hiểu và giữ lại được những nội dung quan trọng, cốt lõi, trong khi loại bỏ những thông tin dư thừa.

Nhờ vào khả năng tổng quát này của mô hình, những dữ liệu đầu vào có tính chất tương đồng với nhau (2 vector đầu vào có khoảng cách gần nhau) thì thông tin sau khi được thu giảm số chiều cũng sẽ gần giống nhau và ngược lại. Ví dụ nếu đầu vào là "Hôm_qua", "Hôm_nay", và "Học"; thì khoảng cách giữa encoded vector của "Hôm_qua" và "Hôm_nay" sẽ gần nhau hơn, so với encoded vector của "Hôm_qua" và "Học".

Tuy nhiên, với tính chất của việc biểu diễn từ bằng one-hot vector (khoảng cách giữa các vector luôn bằng nhau và bằng $\sqrt{2}$). Sau khi được compress bởi mô hình Auto-Encoder, khoảng cách giữa cách encoded vector cũng sẽ bằng nhau cho mọi từ. Do đó, mô hình Auto-Encoder vẫn chưa giải quyết được yêu cầu thứ hai của một giải thuật nhúng từ. Chúng ta cần một kĩ thuật mạnh mẽ hơn, có thể biểu diễn được sự tương quan về mặt ngữ nghĩa giữa các từ bằng định dạng máy tính có thể hiểu được.

4.4.3 Kỹ thuật word2vec

word2vec là một trong những kỹ thuật được sử dụng phổ biến nhất trong lĩnh vực Xử lý ngôn ngữ tự nhiên. *word2vec* được tạo ra và công bố vào năm 2013 bởi một nhóm các nhà nghiên cứu dẫn đầu bởi Tomas Mikolov ở Google và đã được đăng ký bảo hộ quyền phát minh sáng chế (Mikolov et al., 2013).

Kỹ thuật word2vec đã giải quyết vấn đề tương quan ngữ nghĩa của mô hình Auto-Encoder khi biến đổi các từ trong một kho ngữ liệu thành các vector bằng cách dựa vào *thông tin ngữ cảnh* (*contextual information*) của chúng trong kho ngữ liệu đó. Bởi vậy, mô hình sẽ học để sinh ra các vector tương tự nhau cho những từ có ngữ cảnh tương tự nhau.

Thông tin ngữ cảnh của một *từ mục tiêu* (*focus word*) là một *cửa sổ* (*window*) chứa các từ ở bên trái và bên phải của từ mục tiêu, được gọi là các *từ ngữ cảnh* (*context word*). Ta nói *kích thước cửa sổ* (*window size*) $= k$ khi cửa sổ này chứa k từ bên trái và k từ bên phải của từ mục tiêu. Ta xét một ví dụ đơn giản sau:

Ví dụ 4.4.1. Giả sử kho ngữ liệu của ta gồm 3 tài liệu D1, D2, D3:

D1: *Hôm qua tôi học lập trình.*

D2: *Hôm nay tôi cũng học lập trình*

D3: *Ngày mai tôi không học lập trình.*

Sau bước tiền xử lý, kho ngữ liệu trên sẽ được biến đổi thành tập các từ phân biệt được sắp xếp theo thứ tự bảng chữ cái như sau: {*cũng, học, hôm_nay, hôm_qua, không, lập_trình, ngày_mai, tôi*}.

Với kích thước cửa sổ = 1, ta có thông tin ngữ cảnh của các từ trong tài liệu D1 như sau:

Từ mục tiêu	**Từ ngữ cảnh**
hôm_qua	tôi
tôi	hôm_qua, học
học	tôi, lập_trình
lập_trình	học

Bảng 4.8: Từ mục tiêu và ngữ cảnh tương ứng của D1 khi kích thước cửa sổ = 1

Với kích thước cửa sổ = 2, ta có thông tin ngữ cảnh của các từ trong tài liệu D1 như sau:

Từ mục tiêu	**Từ ngữ cảnh**
hôm_qua	tôi, học
tôi	hôm_qua, học, lập_trình
học	hôm_qua, tôi, lập_trình
lập_trình	tôi, học

Bảng 4.9: Từ mục tiêu và từ ngữ cảnh tương ứng của D1 khi kích thước cửa sổ = 2

Tương tự như kỹ thuật Auto-Encoder, word2vec cũng sử dụng một mạng nơ-ron ba lớp. Điểm khác biệt ở đây là đối với kỹ thuật Auto-Encoder, đầu vào và đầu ra cùng là một từ mục tiêu, còn đối với kỹ thuật word2vec ta sẽ dùng từ mục tiêu và các từ ngữ cảnh của nó để làm đầu vào và đầu ra. Tuỳ thuộc vào cách ta sử dụng từ mục tiêu để làm đầu vào hay đầu ra, mà có hai mô hình word2vec khác nhau là

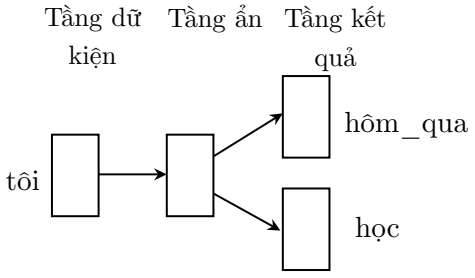
skip-gram và *CBOW* (*Continuous Bag of Words*). Ta sẽ nói rõ chi tiết hai mô hình này ở phần sau.

Skip-gram

Ở mô hình skip-gram, ta sẽ sử dụng từ mục tiêu làm đầu vào và đầu ra mong đợi là từ ngữ cảnh để huấn luyện mạng nơ-ron. Như vậy mỗi mẫu huấn luyện sẽ là một cặp (từ mục tiêu, từ ngữ cảnh) và tập dữ liệu huấn luyện của ta sẽ bao gồm tất cả các cặp từ trong kho ngữ liệu.

Xét kho ngữ liệu bao gồm ba tài liệu D1, D2, D3 như ở Ví dụ 4.4.1. Để huấn luyện mô hình skip-gram, đầu tiên ta đưa đầu vào là *"tôi"* và đầu ra mong đợi là *"hôm_ qua"*. Tiếp tục đưa đầu vào *"tôi"* và đầu ra mong đợi là từ *"học"*. Tương tự như vậy cho tất cả các cặp từ khác trong kho ngữ liệu.

Với kích thước cửa sổ = 1, ta có sơ đồ huấn luyện mô hình skip-gram như Hình 4.6:



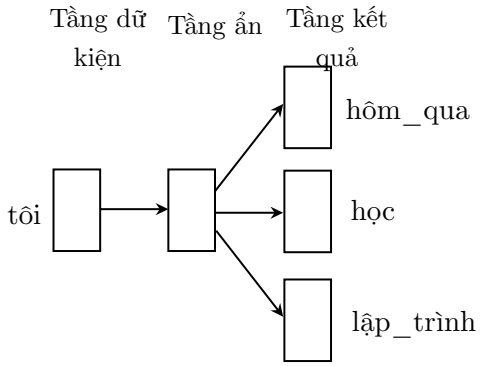
Hình 4.6: Sơ đồ huấn luyện mô hình skip-gram khi kích thước cửa sổ là 1

Với kích thước cửa sổ = 2, ta có sơ đồ huấn luyện mô hình skip-gram như Hình 4.7:

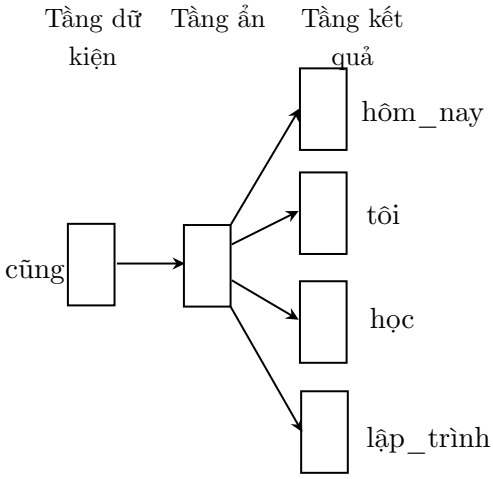
Để ý khi kích thước cửa sổ = 2, do từ mục tiêu là *"tôi"* chỉ có một từ phía bên trái và hai từ phía bên phải, nên ta chỉ có 3 cặp từ. Trong trường hợp từ mục tiêu có đầy đủ từ ngữ cảnh, ta sẽ 4 cặp từ để huấn luyện (2 từ ngữ cảnh bên trái và 2 từ ngữ cảnh bên phải), minh hoạ bằng tài liệu D2 như Hình 4.8

Trong trường hợp tổng quát, khi kích thước cửa sổ = k, thì với mỗi từ mục tiêu ta có khoảng $2k$ cặp từ để đưa vào huấn luyện mạng nơ-ron skip-gram.

Xây dựng mô hình mạng nơ-ron skip-gram Tương tự như kỹ thuật Auto-Encoder, mô hình skip-gram cũng sử dụng mạng nơ-ron 3 lớp gồm tầng dữ kiện, tầng ẩn và



Hình 4.7: Sơ đồ huấn luyện mô hình skip-gram khi kích thước cửa sổ là 2

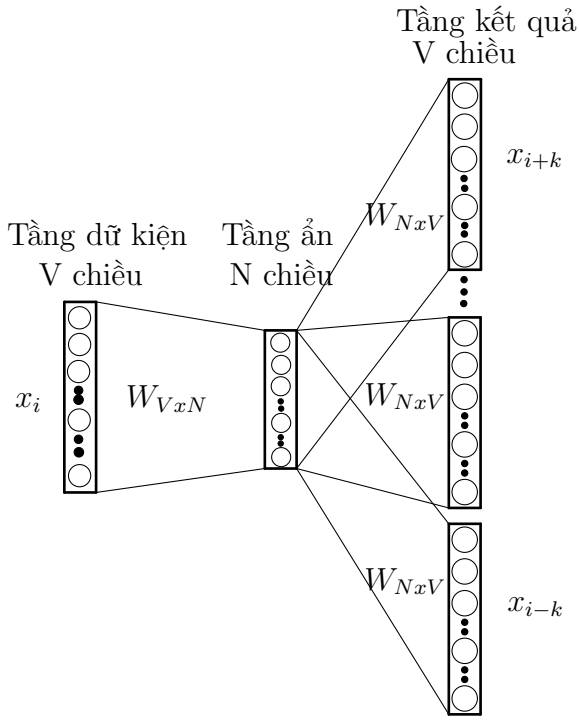


Hình 4.8: Sơ đồ huấn luyện mô hình skip-gram khi kích thước cửa sổ = 2 và sử dụng từ mục tiêu trong tài liệu D2

tầng kết quả. Để đơn giản ta xét kho ngữ liệu chỉ bao gồm 3 tài liệu D1, D2, D3 như ở Ví dụ 4.4.1. Khi đó, mạng nơ-ron sẽ có dạng như Hình 4.9.

Ở mô hình này, chúng ta không sử dụng hàm kích hoạt cho tầng ẩn như các mạng nơ-ron thông thường. Tầng kết quả sẽ sử dụng hàm kích hoạt là softmax. Ta sẽ lần lượt tìm hiểu chi tiết về các tầng này ở phần sau.

Tầng dữ kiện Tương tự như kỹ thuật Auto-Encoder, ta cũng biểu diễn một từ thành một vector one-hot để làm đầu vào của mạng nơ-ron. Kho ngữ liệu gồm D1, D2, D3 trên có 8 *từ phân biệt* (*unique word*) nên mỗi từ sẽ được biễn bằng một vector one-hot có 8 chiều, trong đó chỉ có chiều ở vị trí tương ứng với vị trí của từ trong



Hình 4.9: Mô hình skip-gram dạng tổng quát

kho ngữ liệu bằng 1, còn tất cả các chiều khác đều bằng 0. Vậy tầng dữ kiện sẽ có 8 nơ-ron tương ứng với vector one-hot 8 chiều đó. Bằng cách biểu diễn như trên, ta được các vector one-hot sau:

$cũng = \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$

$học = \begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$

$lập\_trình = \begin{bmatrix}0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$

$hôm\_nay = \begin{bmatrix}0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\end{bmatrix}$

$hôm\_qua = \begin{bmatrix}0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\end{bmatrix}$

$không = \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\end{bmatrix}$

$$ngày\_mai = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$tôi = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Tổng quát, số lượng nơ-ron của tầng dữ kiện sẽ bằng đúng số lượng các từ phân biệt trọng kho ngữ liệu và bằng số chiều của vector one-hot biểu diễn cho mỗi từ. Trong thực tế, kích thước của kho ngữ liệu có thể lên đến hàng triệu từ.

Tầng ẩn Giả sử mạng nơ-ron cần học 4 đặc trưng của các từ, tầng ẩn sẽ bao gồm 4 nơ-ron. Với tầng đầu vào của mô hình là vector one-hot 8 chiều, vậy *ma trận trọng số* (*weight matrix*) cần phải học của tầng ẩn sẽ là 8×4. Chú ý con số 4 đặc trưng của từ là một *siêu tham số* (*hyper parameter*), ta có thể điều chỉnh giá trị này để đạt được kết quả phù hợp tuỳ theo từng bài toán khác nhau.

0.3	0.5	0.2	0.01
1.2	2.4	1.5	0.2
2.5	1.1	0.3	0.55
1	0.15	0.6	2.1
1.3	1.7	2.1	0.24
2.2	1.8	0.6	0.3
1.4	0.12	0.04	2.4
1.12	0.8	0.5	1.25

Bảng 4.10: Minh hoạ ma trận trọng số của tầng ẩn sau khi huấn luyện xong

Tổng quát, số lượng nơ-ron của tầng ẩn bằng với số lượng đặc trưng mà ta muốn học ở mỗi từ trong kho ngữ liệu và cũng là số chiều của vector đầu ra của tầng ẩn. Như vậy, nếu muốn mô hình skip-gram biểu diễn các từ thành vector bao nhiêu chiều thì ta sử dụng bấy nhiêu nơ-ron ở tầng ẩn. Trong thực tế, số lượng nơ-ron của tầng ẩn thường vào khoảng vài trăm. Sau khi huấn luận xong, ta sẽ giữ lại ma trận trọng số để dùng cho mục đích biến đổi từ thành vector. Giả sử kho ngữ liệu có V từ phân biệt và ta muốn biến đổi từ thành vector có N chiều thì ma trận trọng số của mạng nơ-ron skip-gram sẽ có kích thước là $V \times N$. Khi đó mỗi từ (được biểu diễn bằng vector one-hot $1 \times V$) sẽ được biến đổi thành một vector $1 \times N$, vector này sẽ là một hàng trong ma trận trọng số.

Tầng kết quả Tiếp tục xét kho ngữ liệu D1, D2, D3 ở trên, tầng kết quả của mô

hình skip-gram sẽ có số lượng nơ-ron đúng bằng số lượng nơ-ron của tầng dữ kiện, tức là 8 nơ-ron. Tầng kết quả cũng có một ma trận trọng số để học có kích thước 4×8, để biến đổi vector 1×4 của tầng ẩn thành vector 1×8 đầu ra. Sau khi huấn luyện mạng nơ-ron xong, toàn bộ tầng đầu ra và ma trận trọng số này sẽ bị loại bỏ.

0.3	0.5	0..2	0.01	0.23	1	1.24	0.17
1.2	2.4	1.5	0.2	2.5	1.17	0.05	0.26
2.5	1.1	0.3	0.55	1.3	0.35	1.34	1.36
1	0.15	0.6	2.1	2.8	0.68	0.87	2.1

Bảng 4.11: Minh hoạ ma trận trọng số của tầng kết quả sau khi huấn luyện xong

Kết quả Giả sử sau khi huấn luyện xong ta được ma trận trọng số của tầng ẩn như ở Bảng 4.10. Dùng mạng nơ-ron skip-gram này để biến đổi thì mỗi từ thành một vector như sau:

$$cũng = \begin{bmatrix} 0.3 & 0.5 & 0.2 & 0.01 \end{bmatrix}$$

$$học = \begin{bmatrix} 1.2 & 2.4 & 1.5 & 0.2 \end{bmatrix}$$

$$lập\_trình = \begin{bmatrix} 2.5 & 1.1 & 0.3 & 0.55 \end{bmatrix}$$

$$hôm\_nay = \begin{bmatrix} 1 & 0.15 & 0.6 & 2.1 \end{bmatrix}$$

$$hôm\_qua = \begin{bmatrix} 1.3 & 1.7 & 2.1 & 0.24 \end{bmatrix}$$

$$không = \begin{bmatrix} 2.2 & 1.8 & 0.6 & 0.3 \end{bmatrix}$$

$$ngày\_mai = \begin{bmatrix} 1.4 & 0.12 & 0.04 & 2.4 \end{bmatrix}$$

$$tôi = \begin{bmatrix} 1.12 & 0.8 & 0.5 & 1.12 \end{bmatrix}$$

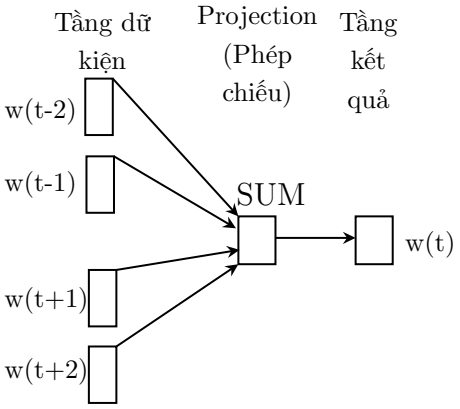
Để ý thấy rằng những từ có ngữ cảnh tương tự nhau thì sẽ cho ra các vector tương tự nhau.

Tổng kết mô hình skip-gram Mô hình skip-gram sử dụng mạng nơ-ron 3 lớp tương tự như kỹ thuật Auto-Encoder nhưng thay vì sử dụng từ mục tiêu cho cả đầu vào và đầu ra thì sử dụng các từ ngữ cảnh làm đầu ra để huấn luyện. Vì vậy, ngoài việc thu giảm số chiều của vector biểu diễn từ, thì mô hình skip-gram đã giải quyết nhược

điểm của mô hình Auto-Encoder, tức là đã giữ lại được ngữ nghĩa của các từ trong kho ngữ liệu. Các từ có ngữ cảnh tương tự nhau sẽ được biến đổi thành thành các vector tương tự nhau.

CBOW (Continuous Bag of Words)

CBOW có đầu vào là từ mục tiêu, đầu ra là các từ trong ngữ cảnh. Ý tưởng chính của mô hình CBOW là dự đoán từ mục tiêu dựa vào các từ ngữ cảnh xung quanh nó trong một phạm vi nhất định. Cho từ mục tiêu w_t tại vị trí t trong câu văn bản, khi đó đầu vào là các từ ngữ cảnh $(w_{t-m}, ..., w_{t-1}, w_{t+1}, ..., w_{t+m})$ xung quanh từ w_t trong phạm vi m.

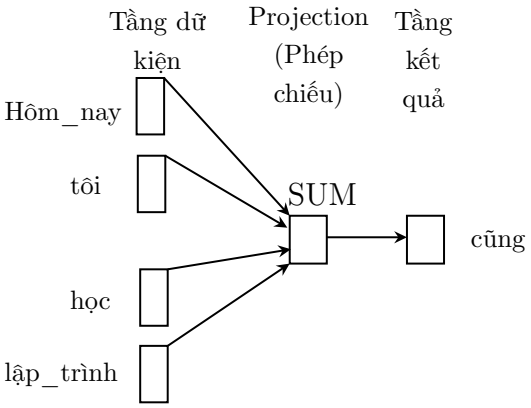


Hình 4.10: Sơ đồ minh hoạ CBOW

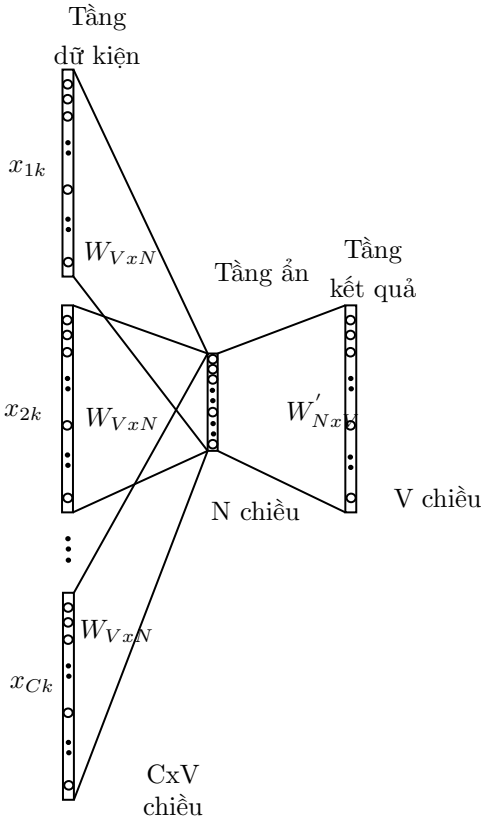
Ví dụ: **"Hôm nay tôi cũng học lập trình"**

Mô hình CBOW tổng quát được minh hoạ trong Hình 4.12 với kích thước đầu vào gồm **C** từ ngữ cảnh, **V** là kích thước của tập từ vựng và siêu tham số **N** là kích thước của tầng ẩn. Mỗi nút thuộc tầng sau được kết nối với các nút của tầng trước theo kiểu kết nối đầy đủ.

Mỗi từ đầu vào ở vị trí thứ k trong từ vựng được biểu diễn bằng một vector one-hot dưới dạng:



Hình 4.11: Sơ đồ minh hoạ CBOW với ví dụ trên



Hình 4.12: Mô hình CBOW dạng tổng quát

$$x^k = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ \vdots \\ x_V \end{bmatrix}$$

Với mỗi vector x^k chỉ có một phần tử x_k có giá trị 1, các phần tử còn lại có giá trị 0.

Ma trận trọng số giữa tầng dữ kiện và tầng ẩn W kích thước $V \times N$ được biểu diễn như sau:

$$W_{V\times N} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1N} \\ w_{21} & w_{22} & \dots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{V1} & w_{V2} & \dots & w_{VN} \end{bmatrix}$$

Mỗi hàng của ma trận **W** là một biểu diễn vector có số chiều là **N** tương ứng với một từ **w** trong tập từ vựng.

Với mỗi vector x^k, ta có:

$$W^T x^k = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1V} \\ w_{21} & w_{22} & \dots & w_{2V} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N1} & w_{N2} & \dots & w_{NV} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ \vdots \\ x_V \end{bmatrix} = \begin{bmatrix} w_{k1}x_k \\ w_{k2}x_k \\ \vdots \\ w_{kN}x_k \end{bmatrix} = \begin{bmatrix} w_{k1} \\ w_{k2} \\ \vdots \\ w_{kN} \end{bmatrix} = v_k^T \tag{4.1}$$

Ma trận h có kích thước $N \times 1$ được biểu diễn như sau:

$$\begin{aligned} h &= \frac{1}{C} \times W^T \left(x^{(1)} + x^{(2)} + \cdots + x^{(C)}\right) \\ &= \frac{1}{C} \times \left(v_1^T + v_2^T + \cdots + v_C^T\right) \\ &= \frac{1}{C} \times (v_1 + v_2 + \cdots + v_C)^T \end{aligned} \tag{4.2}$$

Từ tầng ẩn đến tầng kết quả là các trọng số được biểu diễn bằng một ma trận W' với kích thước $N \times V$ có dạng như sau:

$$W'_{N\times V} = \begin{bmatrix} w'_{11} & w'_{12} & \dots & w'_{1V} \\ w'_{21} & w'_{22} & \dots & w'_{2V} \\ \vdots & \vdots & \ddots & \vdots \\ w'_{N1} & w'_{N2} & \dots & w'_{NV} \end{bmatrix}$$

Kết quả của tầng ẩn sẽ được map với đầu ra chính là trọng số của từng từ có trong V, trọng số càng cao có nghĩa là xác suất từ đó là từ tiếp theo (*positive prediction*) càng cao. Do trọng số ở tầng kết quả có biên độ không cố định nên thường thì người ta dùng softmax ở đoạn này nhằm đổi trọng số của tầng kết quả thành phân phối xác suất.

Chúng ta sẽ tính điểm số cho mỗi từ thứ j trong V theo công thức:

$$p_j = v'^T_{wj} h$$

Trong đó v'_{wj} là vector cột thứ j của ma trận W'. Hàm $softmax$ để chuẩn hoá phân phối hậu nghiệm của mỗi tập từ vựng như sau:

$$p\left(w_O|w_{I,1}, w_{I,2}, ..., w_{I,C}\right) = y_j = \frac{exp(u_j)}{\sum_{j'}^{V} exp(u_{j'})} \tag{4.3}$$

Trong đó y_j chính là đầu ra của nút thứ j trong tầng kết quả, w_O là từ mục tiêu, $w_{I,1}, ..., w_{I,C}$ là các từ ngữ cảnh. Mục tiêu của quá trình huấn luyện là điều chỉnh các trọng số để cực đại hoá hàm phân phối bên trên đối với từ w_O và các từ ngữ cảnh $w_{I,1}, ..., w_{I,C}$ cho trước.

$$\begin{aligned} max\left(p\left(w_O|w_{I,1}, w_{I,2}, ..., w_{I,C}\right)\right) &= max\left(log\left(p\left(w_O|w_{I,1}, w_{I,2}, ..., w_{I,C}\right)\right)\right) \\ &= min\left(-log\left(p\left(w_O|w_{I,1}, w_{I,2}, ..., w_{I,C}\right)\right)\right) = min(E) \end{aligned}$$

Với E là hàm mất mát cần được cực tiểu hoá, có công thức như sau:

$$
\begin{aligned}
E &= -log\left(p\left(w_O|w_{I,1}, w_{I,2}, ..., w_{I,C}\right)\right) \\
&= -u_{j^*} + log\left(\sum_{j'=1}^{V} exp\left(u_{j'}\right)\right) \\
&= -v'^T_{wO} h + log\left(\sum_{j'=1}^{V} exp\left(v'^T_{wO} h\right)\right)
\end{aligned}
\tag{4.4}
$$

Trong đó j^* là vị trí của từ mục tiêu ở tầng kết quả. Phương trình cập nhật trọng số từ tầng ẩn đến tầng kết quả như sau:

$$
v^{new}_{wj'} = v^{old}_{wj'} - \eta \times \frac{\partial E}{\partial u_j} \times h \quad \text{với } j = 1, 2, ..., V \tag{4.5}
$$

Phương trình cập nhật trọng số từ tầng dữ kiện đến tầng ẩn như sau:

$$
v^{new}_{w_c} = v^{old}_{w_c} - \frac{1}{C} \times \eta \times \left(\frac{\partial E}{\partial h_i}\right)^T \quad \text{với } c = 1, 2, ..., C \tag{4.6}
$$

Trong đó v_{w_c} là vector đầu vào tương ứng từ thứ c trong C từ ngữ cảnh đầu vào và η là tốc độ học. Sau khi quá trình huấn luyện hoàn tất, chúng ta sẽ thu được 2 ma trận trọng số là W và W'.

Với W' có kích thước $N \times V$, ta có:

$$
\begin{gathered}
W'^T_{N \times V} \times h = z \\
y = softmax(z)
\end{gathered}
$$

Nhắc lại, mô hình CBOW cần phải được huấn luyện để điều chỉnh 2 ma trận trọng số W (từ tầng dữ kiện đến tầng ẩn) và W' (từ tầng ẩn đến tầng kết quả) sao cho vector y có giá trị xác suất lớn nhất tại vị trí của từ mục tiêu.

Chúng ta sẽ lấy ví dụ để hiểu hơn về cách thức hoạt động của CBOW, cũng với câu

"Hôm qua tôi đi học"

Với kích thước cửa sổ = 1, ta sẽ chọn từ mục tiêu (tầng kết quả) là "đi", và 2 từ ngữ cảnh (tầng dữ kiện) là "tôi" và "học".

Cho vector one-hot của từ "tôi" và "học" như sau:

$$x^{toi} = \begin{bmatrix} 0 \\ \mathbf{1} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$x^{hoc} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \mathbf{1} \\ \vdots \\ 0 \end{bmatrix}$$

Cho ma trận trọng số $\mathbf{W}$ có kích thước $V \times N$ sau khi được huấn luyện như sau:

$$W^T_{V \times N} = \begin{bmatrix} 0.1 & 2.4 & 1.6 & 1.8 & 0.5 & 0.9 & \dots & 3.2 \\ 0.5 & 2.6 & 1.4 & 2.9 & 1.5 & 2.6 & \dots & 6.1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.6 & 1.8 & 2.7 & 1.9 & 2.4 & 3.0 & \dots & 1.2 \end{bmatrix}$$

Ta có:

$$v_{toi}^T = W^T \times x^{toi} = \begin{bmatrix} 0.1 & \mathbf{2.4} & 1.6 & 1.8 & 0.5 & 0.9 & \dots & 3.2 \\ 0.5 & \mathbf{2.6} & 1.4 & 2.9 & 1.5 & 2.6 & \dots & 6.1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.6 & \mathbf{1.8} & 2.7 & 1.9 & 2.4 & 3.0 & \dots & 1.2 \end{bmatrix} \times \begin{bmatrix} 0 \\ \mathbf{1} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{2.4} \\ \mathbf{2.6} \\ \vdots \\ \mathbf{1.8} \end{bmatrix} \quad (4.7)$$

$$v_{hoc}^T = W^T \times x^{hoc} = \begin{bmatrix} 0.1 & 2.4 & 1.6 & \mathbf{1.8} & 0.5 & 0.9 & \dots & 3.2 \\ 0.5 & 2.6 & 1.4 & \mathbf{2.9} & 1.5 & 2.6 & \dots & 6.1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.6 & 1.8 & 2.7 & \mathbf{1.9} & 2.4 & 3.0 & \dots & 1.2 \end{bmatrix} \times \begin{bmatrix} 0 \\ \mathbf{1} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{1.8} \\ \mathbf{2.9} \\ \vdots \\ \mathbf{1.9} \end{bmatrix} \quad (4.8)$$

Ta tính được h như sau:

$$h = \left(\frac{v_{toi}^T + v_{hoc}^T}{2} \right)$$

Sau khi có được h, ta cũng sẽ dùng công thức tính z và y để tìm ra từ mục tiêu gần nhất với các từ đầu vào.

Giả sử ta có vector mục tiêu của từ "đi" ở tầng kết quả có kích thước $V \times 1$ như sau:

$$y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \mathbf{1} \\ \vdots \\ 0 \end{bmatrix}$$

Quá trình huấn luyện sẽ cho ra vector mục tiêu dưới dạng xác suất của từ "đi" như ví dụ dưới:

$$y_{training} = \begin{bmatrix} 0.01 \\ 0.02 \\ 0.05 \\ 0.02 \\ 0.03 \\ \mathbf{0.7} \\ \vdots \\ 0.00 \end{bmatrix}$$

Chú ý: khi hiện thực, các vector ngữ cảnh đầu vào sẽ được cộng dồn lại thành một vector có 2N số 1 (với N là số từ đứng trước/đứng sau từ mục tiêu), để cho ma trận encoder cuối cùng vẫn là một ma trận $N \times V$ như mô hình Auto-Encoder gốc.

Ưu điểm của CBOW:

- Có bản chất là xác suất cho nên việc hiện thực được cho là vượt trội hơn so với phương pháp khác nói chung.
- Sử dụng ít bộ nhớ.

Nhược điểm của CBOW:

- CBOW lấy trung bình các ngữ cảnh của 1 từ (ở công thức tính hàm h). Ví dụ: Từ "Apple" vừa có thể có nghĩa là "trái cây" lại vừa là "tên của một công ty". CBOW lấy trung bình của 2 từ tữ cảnh và đặt ở giữa.

4.5 Bài tập

Bài tập TF-IDF và Auto-Encoder

Bài tập 4.5.1. Cho corpus gồm 3 tài liệu như sau:

- Hay mình đi ăn bún cá
- Mình đi ăn phở nha
- Mình ăn sáng nha

Cho sẵn tập từ vựng (sau khi tokenize) từ tập corpus ở trên như sau:
{ hay, mình, đi, ăn, bún_cá, phở, nha, sáng }

Hãy vector hóa 3 tài liệu trên bằng phương pháp TF-IDF.

Bài tập 4.5.2. Cho mô hình Auto-Encoder như Hình 4.5. Trong đó

- Tập từ điển tương tự như bài tập 4.5.1.
- Trạng thái ma trận Encoder, và Decoder lần lượt như sau:

Ma trận Encoder 8 x 4:
$$\begin{bmatrix} 0.2 & 0.1 & 0.5 & 0.2 \\ 0.6 & 0.5 & 0.3 & 0.3 \\ 0.4 & 0.2 & 0.8 & 0.9 \\ 0.1 & 0.7 & 0.5 & 0.3 \\ 0.8 & 0.4 & 0.9 & 0.1 \\ 0.5 & 0.1 & 0.1 & 0.3 \\ 0.1 & 0.6 & 0.1 & 0.7 \\ 0.3 & 0.2 & 0.4 & 0.5 \end{bmatrix}$$

Ma trận Decoder 4 x 8:
$$\begin{bmatrix} 0.3 & 0.3 & 0.6 & 0.1 & 0.7 & 0.7 & 0.3 & 0.8 \\ 0.2 & 0.5 & 0.4 & 0.2 & 0.7 & 0.1 & 0.5 & 0.2 \\ 0.1 & 0.5 & 0.3 & 0.9 & 0.9 & 0.2 & 0.3 & 0.1 \\ 0.3 & 0.5 & 0.2 & 0.5 & 0.1 & 0.2 & 0.6 & 0.1 \end{bmatrix}$$

Cho biết vector đầu ra của mô hình của khi vector đầu vào là vector one-hot của từ "Phở"?

Bài tập vận dụng word2vec

Bài tập 4.5.3. Sử dụng *kho ngữ liệu* D1, D2, D3 ở Ví dụ 4.4.1, hãy cho biết *từ mục tiêu* và *từ ngữ cảnh* của các từ trong tài liệu D3 cho hai trường hợp *kích thước cửa sổ* bằng 1 và 2. Chú ý vẽ bảng gồm hai cột *từ mục tiêu* và *từ ngữ cảnh* tương tự như trong ví dụ.

Bài tập 4.5.4. Sử dụng *kho ngữ liệu* D1, D2, D3 ở Ví dụ 4.4.1, xét *từ mục tiêu* là *tôi* trong tài liệu D3 với *kích thước cửa sổ* $= 1$. Hãy cho biết khi từ *tôi* được đưa vào huấn luyện thì kiến trúc mạng nơ-ron skip-gram sẽ như thế nào? Giả sử vector sau khi biến đổi có số chiều như trong ví dụ. Chú ý nêu chi tiết số lượng nơ-ron ở mỗi tầng, vẽ mô hình mà nói rõ đầu vào và đầu ra cho mỗi cặp từ.

Bài tập 4.5.5. Sử dụng *kho ngữ liệu* D1, D2, D3 ở Ví dụ 4.4.1, xét *từ mục tiêu* là *học*, *hai từ ngữ cảnh* là *tôi* và *lập trình* trong tài liệu D1 với *kích thước cửa sổ* là 1. Hãy cho biết sơ đồ mô hình *CBOW* sẽ như thế nào? Giả sử vector sau khi biến đổi có số chiều như trong ví dụ.

4.6 Phụ lục

Dưới đây là một số thuật ngữ có đề cập trong chương 4:

Bảng 4.12: Bảng các thuật ngữ

Thuật ngữ	Ý nghĩa
text analyzer	phân tích văn bản
text classification	phân loại văn bản
text summarization	tóm tắt văn bản
corpus	kho ngữ liệu
word	từ
phrase	cụm từ
sentence	câu
text	văn bản
term	_
token	_
contextual information	thông tin ngữ cảnh
focus word	từ mục tiêu
context word	từ ngữ cảnh
window	cửa sổ
window size	kích thước cửa sổ
unique word	từ phân biệt
weight matrix	ma trận trọng số
hyper parameter	siêu tham số
skip-gram	một phương pháp word2vec
Continuous Bag of Word (CBOW)	một phương pháp word2vec
word embedding	kỹ thuật nhúng từ
one-hot vector	_
one-hot encoding	_
parameter	thông số
Auto-Encoder	một kỹ thuật nhúng từ
word2vec	một kỹ thuật nhúng từ
high-dimensional vector	vector nhiều chiều
low-dimensional vector	vector ít chiều

Chương 5

Mạng nơ-ron tích chập (Convolutional Neural Network)

5.1 Giới thiệu mạng nơ-ron tích chập

5.1.1 Lịch sử mạng nơ-ron tích chập

Mạng Nơ-ron tích chập (Convolutional Neural Network, còn được viết tắt là CNN hay ConvNet) là một trong những nền tảng quan trọng của chuyên ngành *Thị giác máy tính* (Computer Vision) nói riêng, hay của ngành *Học sâu* (Deep learrning) nói chung. CNN được chuyên dùng cho các bài toán liên quan đến hình ảnh như phân loại, phân tích hình ảnh (image classification) hay nhận diện khuôn mặt (facial recognition). Tác giả của kiến trúc này, không ai khác, chính là nhà nghiên cứu lỗi lạc Yann LeCun, người từng đạt giải Turing năm 2018. Ông đã trình bày ý tưởng của mình thông qua bài báo *Lan truyền ngược áp dụng cho nhận dạng chữ viết tay cho zipcode* (Backpropagation applied to handwritten zip code recognition) (LeCun et al., 1989).

Những ý tưởng sơ khai đầu tiên để hình thành nên một mạng nơ-ron tích chập như hiện nay đã bắt nguồn từ những năm 50 và 60 của thế kỷ trước. Trong một bài báo khoa học của 2 nhà nghiên cứu David H. Hubel và Torsten Wiesel về những vùng cảm thụ của vỏ não thị giác (receptive fields of visual cortex), tác giả đã nêu ra cách

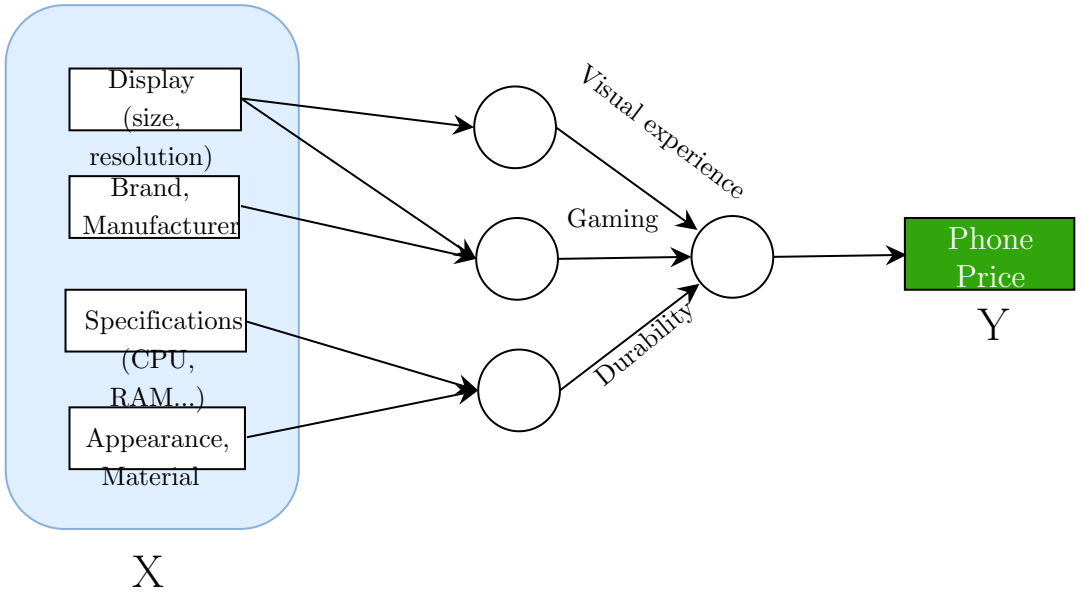
xử lý hình ảnh của các tế bào thị giác chính là nhận biết các *cạnh thẳng* (straight edges). Các tế bào thị giác được chia làm 2 loại, đó là *tế bào tối giản* (simple cell) và *tế bào phức hợp* (complex cell). 2 nhà khoa học cũng đưa ra mô hình xếp tầng (cascading model) để lý giải cách nhận dạng mẫu (pattern recognition) của 2 loại tế bào này (Hubel and Wiesel, 1959).

Đến năm 1979, nhà khoa học dữ liệu người Nhật Bản Kunihiko Fukushima đưa ra khái niệm "Neocognitron", đây được xem là mạng Nơ-ron tích chập nguyên thủy của ngành Thị giác máy tính. Dựa trên những ý tưởng của Hubel và Wiesel, các *tầng* (layer) của Neocognitron được chia làm 2 loại cơ bản: tầng S-cells được sử dụng để trích xuất các *thuộc tính cục bộ* (local feature) của hình ảnh, tương đương với simple cell (dùng để xử lý các đường thẳng cơ bản); trong khi C-cells được dùng để giảm thiểu các sai sót khi kết hợp nhiều thuộc tính cục bộ với nhau (tương đương với complex cell). Các *thuộc tính cục bộ* được tích hợp dần và phân loại ở các tầng sâu hơn. Từ đó máy tính sẽ nhận dạng được các đặc điểm khác nhau của hình ảnh trong quá trình học và có thể phân loại được hình ảnh dựa trên các *bộ lọc* khác nhau (filter). Đến nay thì những ý tưởng cơ bản của các mạng CNN hiện đại vẫn dựa trên mạng Neocognitron này (Fukushima, 1980).

Có thể thấy, những ý tưởng khởi phát của CNN đã có từ cách đây khá lâu (gần 30 năm), nhưng sự trỗi dậy của ngành Học sâu, cụ thể hơn là Thị giác máy tính chỉ thực sự bắt đầu từ những năm đầu của Thế kỷ XXI với sự phát triển của ngành công nghiệp Trò chơi điện tử. Các card đồ họa ngày càng được nâng cấp để nâng cao trải nghiệm chơi game cho người dùng, mà cốt lõi chính là việc tăng cường khả năng tính toán của các bộ xử lý đồ họa (GPU- Graphic Processing Units). Thoạt nghe thì có vẻ CNN không dính dáng tới GPU lắm vì chúng thuộc 2 chuyên ngành khác biệt, nhưng thực tế chúng đều hoạt động dựa trên các thao tác với ma trận (matrix manipulation). Hình ảnh là một ma trận các điểm ảnh (pixel), còn video thì ghép nối nhiều hình ảnh khác nhau trong một khoảng thời gian nhất định (60 frame per second chẳng hạn) để tạo nên những chuyển động. Do tính chất này, GPU được cải thiện sao cho việc tính toán các ma trận pixel được diễn ra nhanh nhất có thể, điều này vô tình khiến các mạng nơ-ron nhân tạo được hưởng lợi rất nhiều.

5.1.2 Đặc tính cơ bản của mạng nơ-ron tích chập

Vậy tại sao mạng nơ-ron tích chập lại được sử dụng phổ biến cho các bài toán phân tích hình ảnh? Câu trả lời đến từ một bản chất tự nhiên của CNN, đó là khả năng *rút trích thuộc tính ẩn* (latent feature extraction) cực kỳ tốt. Để có thể nhận biết rõ hơn về tầm quan trọng của việc rút trích được những thuộc tính ẩn,ta hãy cùng phân tích ví dụ dưới đây về mối liên hệ giữa cách thức con người suy nghĩ để đưa ra quyết định và quá trình học của mạng nơ-ron nhân tạo.

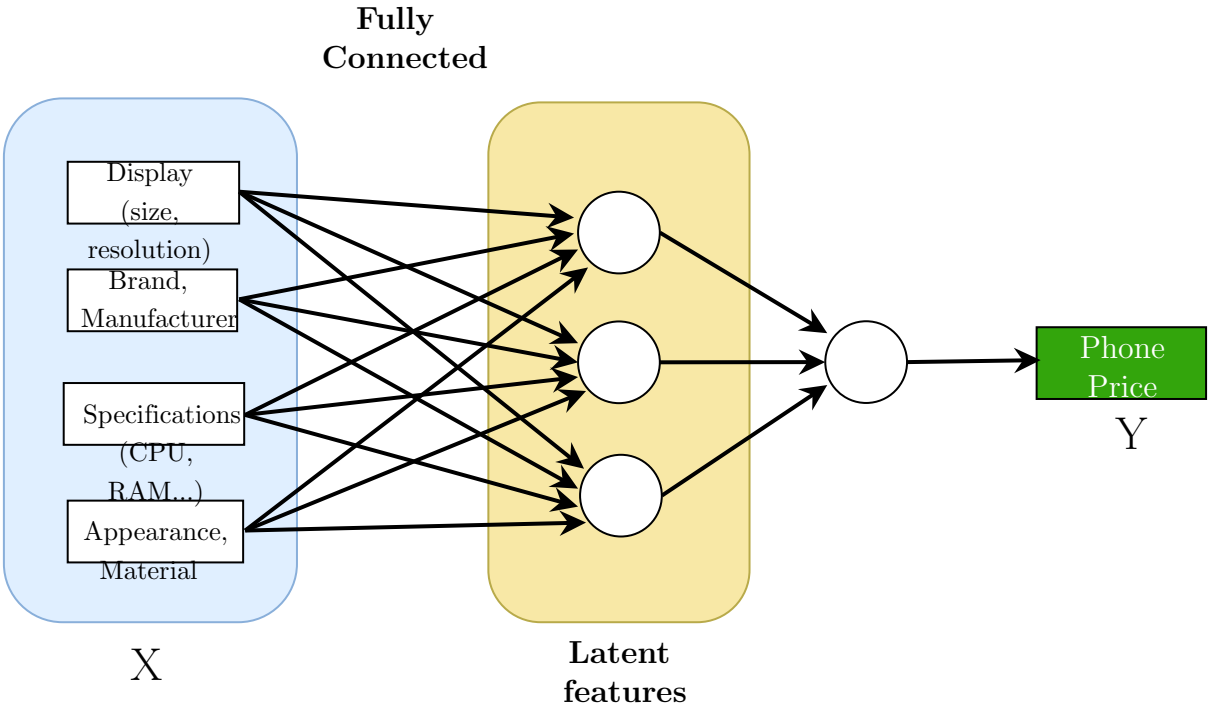


Hình 5.1: Hình ảnh minh họa phân tích của nhà phân phối điện thoại thông minh

Hình 5.1 là một minh họa về quá trình phân tích và rút trích các thuộc tính ẩn để từ đó đưa ra quyết định định giá sản phẩm của một nhà phân phối điện thoại thông minh. Có thể thấy được giá trị đầu vào chính là những thông số thường được đi kèm với mỗi dòng điện thoại như màn hình, vi xử lý, RAM, hãng sản xuất,...Tuy nhiên, các thông số này không trực tiếp quyết định số tiền mà người dùng cần bỏ ra để sở hữu chiếc điện thoại. Bản thân người dùng khi mua một sản phẩm luôn có những yêu cầu nhất định và phù hợp cho cá nhân mình, chẳng hạn như độ bền, khả năng chiến game, trải nghiệm hiển thị,...

Như vậy, đối với nhà phân phối, họ cần rút trích được các thuộc tính tiềm ẩn của sản phẩm và có ý nghĩa đối với khách hàng của họ (cũng chính là nhu cầu người

dùng) từ những thông số đầu vào tương đối vô nghĩa. Ví dụ như thông số màn hình (kích thước, độ phân giải, tấm nền) sẽ quyết định trải nghiệm thị giác; thông số màn hình cộng với thông số phần cứng như chip xử lý, bộ nhớ RAM sẽ quyết định trải nghiệm chơi game; hoặc hãng sản xuất, chất liệu và thiết kế của sản phẩm có thể ảnh hưởng đến độ bền. Từ đó, nhà phân phối đưa ra giá trị cho mỗi loại điện thoại sao cho phù hợp và bán chạy khi giới thiệu đến người dùng.



Hình 5.2: Hình ảnh minh họa quá trình học và trích xuất thuộc tính của một mạng nơ-ron

Hình 5.2 mô tả cơ bản quá trình học của một *mạng nơ-ron nhân tạo* (neural network). Từ các giá trị đầu vào, mạng nơ-ron tiến hành các thao tác xử lý để trích xuất các *thuộc tính tiềm ẩn* (latent features) của dữ liệu. Các thuộc tính này có thể tiếp tục được làm đầu vào cho các lớp sau đó cho đến khi tìm được kết quả cuối cùng.

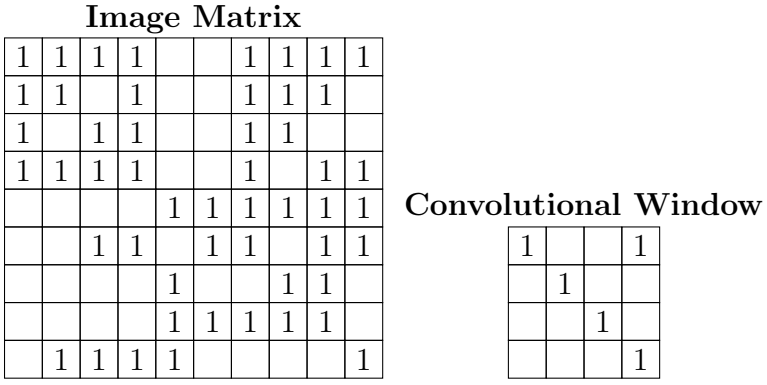
Điểm đặc biệt của các thuộc tính tiềm ẩn được trích xuất từ mạng nơ-ron nhân tạo chính là thậm chí con người cũng đôi khi cũng không thể, hoặc rất khó khăn để suy luận ra được. Một ví dụ đơn giản chính là cách quá trình trẻ nhỏ nhận biết các con vật khác nhau. Dù thực tế chúng ta không được học về một định nghĩa hay lý

thuyết cụ thể để phân biệt các loài vật, nhưng não bộ ta đã học được điều đó trong quá trình tiếp xúc trực tiếp hay gián tiếp. Do đó, khi được hỏi hãy liệt kê cụ thể các đặc điểm để phân biệt chó và mèo, đôi khi chúng ta cũng gặp không ít khó khăn để trích xuất các đặc điểm này, việc mà các mạng nơ-ron nhân tạo được thực hiện bởi máy tính lại thực hiện rất tốt.

Có nhiều mạng nơ-ron có thể đảm nhiệm tốt khả năng trích xuất thuộc tính. Tuy nhiên, đối với một số mạng nơ-ron sử dụng cơ chế *kết nối đầy đủ* (fully-connected), có một vấn đề nảy sinh chính là số *trọng số* (parameter) cần học là rất lớn bởi vì tất cả các giá trị đầu vào đều tham gia vào quá trình học của mỗi trọng số. Điều này có thể khiến cho quá trình huấn luyện diễn ra tương đối chậm. Với mạng nơ-ron tích chập, cơ chế *chia sẻ trọng số* (shared weight) sẽ giúp tiết kiệm đáng kể chi phí với trọng số được học giảm rất nhiều. Chúng ta sẽ cùng phân tích về cơ chế chia sẻ trọng số này kỹ hơn ở phần sau của chương.

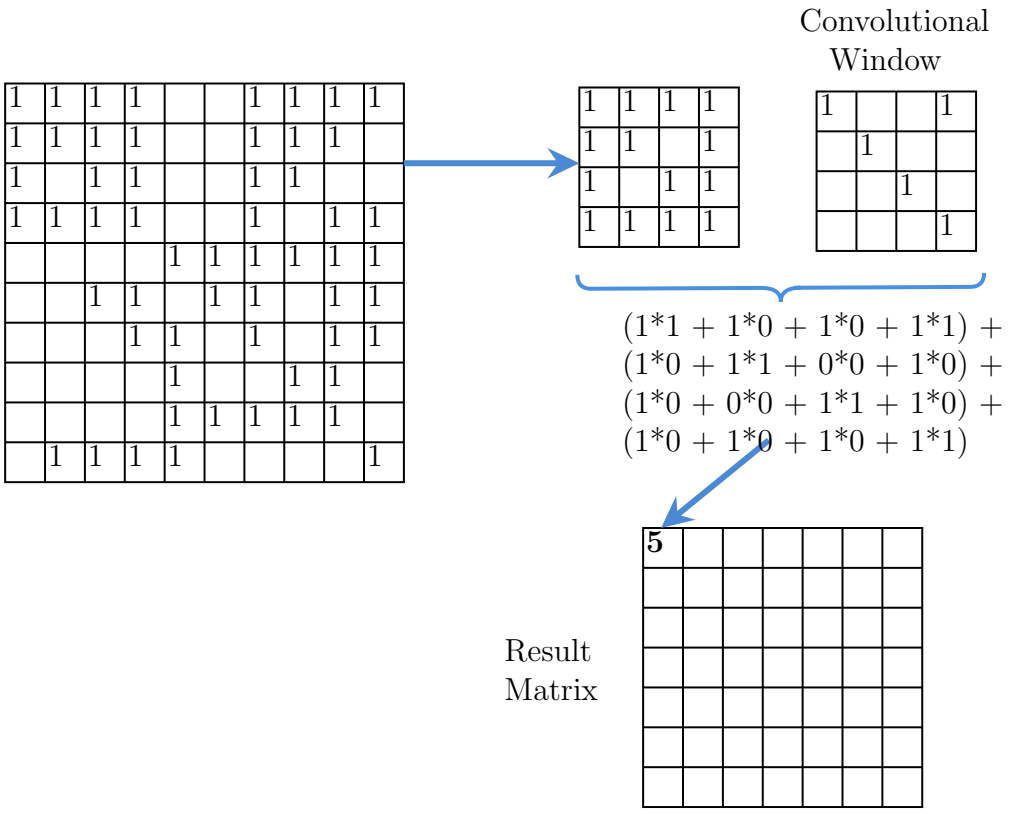
5.2 Cách thức hoạt động của mạng nơ-ron tích chập

5.2.1 Phép tích chập (Convolution)



Hình 5.3: Hình ảnh mô tả ma trận hình ảnh và cửa sổ tích chập

Đúng như cái tên của CNN, phép tính *tích chập* (convolution) chính là đặc trưng mạng nơ-ron này. Để có thể dễ dàng nắm bắt được ý tưởng của phép tích chập, ta hãy cùng phân tích ví dụ sau đây.



Hình 5.4: Hình ảnh minh họa quá trình nhân tích chập cho phần tử đầu tiên

Xét một hình ảnh được biểu diễn dưới dạng ma trận I kích thước 10x10 gồm các chữ số 1 (các vị trí không có số 1 được hiểu là 0) và một cửa sổ tích chập K (cửa sổ trượt) là ma trận 4×4 như Hình 5.3. Ta có công thức tính một phần tử của ma trận kết quả từ phép tích chập như sau:

$$S_{ij} = (I * K)_{ij} = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{5.1}$$

với i, j là vị trí dòng và cột của phần tử kết quả, m và n là kích thước của ma trận cửa sổ tích chập.

Để dễ hình dung hơn về phép tích chập, ta sẽ xét ví dụ trên bằng cách trực quan hóa thông qua hình ảnh. Ta thực hiện phép tích chập bằng cách trượt cửa sổ như sau:

- Nhân tích chập cửa sổ trượt với ma trận con 4x4 tại vị trí đầu tiên của ma trận hình ảnh. Phép nhân tích chập được thực hiện bằng cách nhân từng phân tử của ma trận hình ảnh con với phần tử tại ví trí tương ứng của cửa sổ trượt sau đó cộng các kết quả lại. Kết quả cuối cùng chính là giá trị của phần tử đầu tiên trong ma trận kết quả. Hình 5.4 mô tả cụ thể quá trình này.
- Trượt cửa sổ tích chập sang phải của ma trận hình ảnh một đơn vị cột và tiếp tục nhân tích chập để tìm được phần tử thứ 2 trong ma trận kết quả. Tiếp tục trượt sang phải cho đến khi không thể trượt thêm. Như vậy ta có được các phần tử của hàng đầu tiên cho ma trận kết quả tích chập.

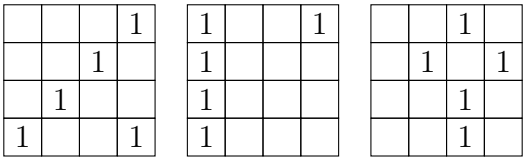
5	2	2	3	2	4	4
5	2	2	3	2	4	4
5	2	2	3	2	4	4
5	2	2	3	2	4	4
5	2	2	3	2	4	4
5	2	2	3	2	4	4
5	2	2	3	2	4	4

Hình 5.5: Hình ảnh mô tả ma trận kết quả đầy đủ sau khi nhân tích chập với cửa sổ trượt đầu tiên

- Để tính tiếp hàng thứ 2 cho ma trận kết quả, ta dời cửa sổ trượt về vị trí ban đầu và trượt cửa sổ xuống một đơn vị hàng. Tiếp tục lặp lại quá trình nhân tích chập và trượt cửa sổ. Ma trận kết quả sau khi hoàn thành quá trình tích chập sẽ có kích thước 7x7 như Hình 5.5.

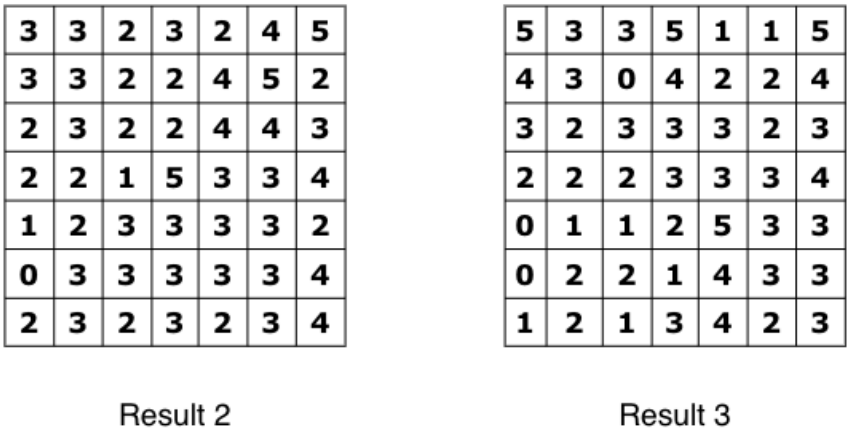
Như vậy, ta đã có thể nắm được quá trình nhân tích chập diễn ra như thế nào.

Để thực hành làm quen với phép tích chập, ta hãy tiếp tục nhân tích chập ma trận hình ảnh ban đầu cho các cửa sổ trượt 2,3 và 4 như trong Hình 5.6. Kết quả nhận được sẽ giống với 3 ma trận kết quả trong Hình 5.7.



Filter 1 **Filter 2** **Filter 3**

Hình 5.6: Hình ảnh mô tả 3 cửa sổ trượt tiếp theo



Result 2

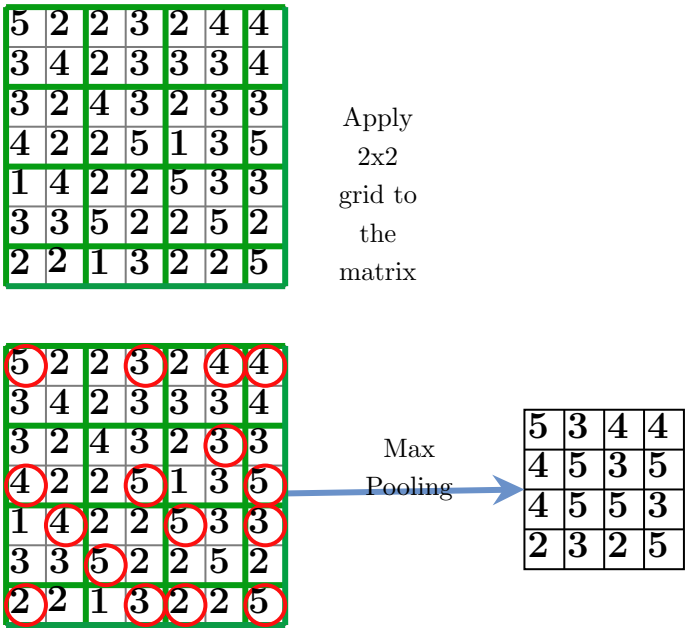
Result 3

Result 4

Hình 5.7: Hình ảnh mô tả 3 ma trận kết quả sau khi nhân tích chập từ 3 cửa sổ trượt trên

5.2.2 Phép gộp (Pooling)

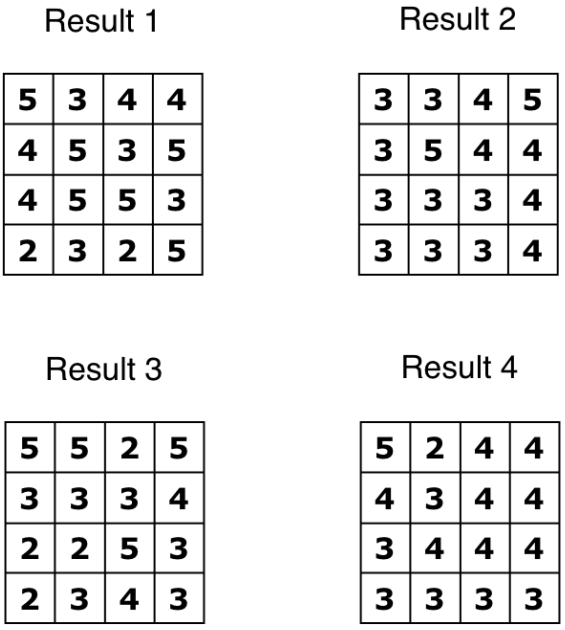
Sau khi có được các ma trận kết quả với kích thước 7x7 từ phép tính tích chập. Ta tiếp tục thực hiện *phép gộp* (pooling) để tiếp tục thu nhỏ ma trận hình ảnh. Có nhiều cách thực hiện phép gộp khác nhau như: lấy giá trị lớn nhất (max pooling), lấy giá trị trung bình (averrage pooling) hay lấy giá trị tổng (sum pooling). Trong thực tế, max pooling có khả năng khử nhiễu tốt hơn các loại pooling khác, vì vậy ta sẽ sử dụng max pooling để làm ví dụ minh họa cho phép gộp.



Hình 5.8: Hình ảnh minh họa quá trình thực hiện phép gộp max pooling

Xét ma trận kết quả được tính từ cửa sổ tích gộp đầu tiên. Ta thực hiện phép gộp max pooling như sau (Hình 5.8 mô tả quá trình thực hiện phép gộp này):

- Chia ma trận kết quả ra thành các ma trận con (grid) có kích thước 2×2. Lưu ý là do ma trận kết quả có kích thước là 7×7 nên các ma trận con nằm ở cuối hàng và cuối cột chỉ có kích thước 1×2 hoặc 2×1.
- Giá trị lớn nhất trong mỗi ma trận con (khoanh tròn màu đỏ) chính là phần tử của ma trận gộp mới. Lúc này ma trận kết quả sau khi gộp chỉ còn có kích thước 4×4, giảm đáng kể so với ma trận 10×10 ban đầu.



Hình 5.9: Hình ảnh mô tả 4 ma trận kết quả sau khi thực hiện max pooling

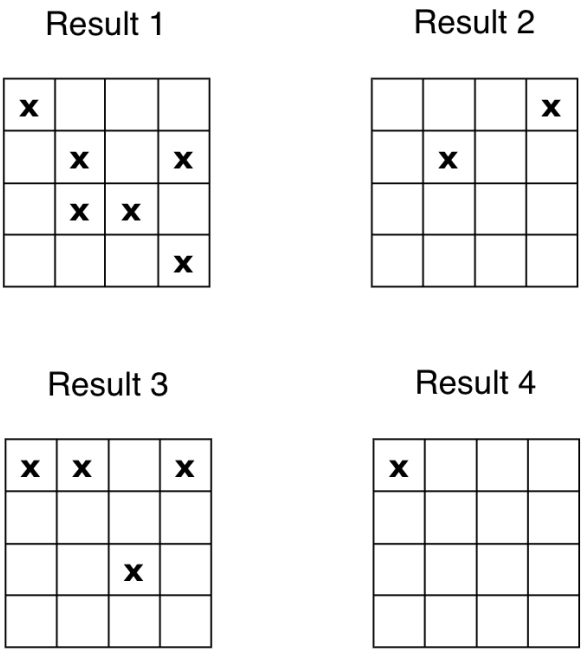
Tiếp tục thực hiện phép gộp dùng giá trị lớn nhất cho 3 ma trận kết quả tích chập còn lại, ta nhận được 4 ma trận kết quả tương ứng với 4 cửa sổ trượt ban đầu như Hình 5.9.

Như vậy, ta đã nắm bắt được ý tưởng của phép toán thứ 2 trong mạng nơ-ron tích chập đó là phép gộp (*pooling*).

5.2.3 Ý nghĩa của ma trận kết quả

Sau khi đã thực hiện *phép tích chập* (convolution) và *phép gộp* (pooling), ta có được 4 ma trận kết quả kích thước 4x4 tương ứng với 4 cửa sổ trượt (Hình 5.9). Ta nhận thấy giá trị tối đa có thể có được của mỗi phần tử là 5. Do đó, ta thực hiện thêm một biến đổi cho 4 ma trận này bằng cách chuyển các giá trị 5 thành dấu x, các giá trị nhỏ hơn 5 đều được thay bằng 0. Sau phép biến đổi này, ta có được 4 ma trận mới như Hình 5.10.

Có thể thấy, sau khi biến đổi thì các ma trận kết quả đang thể hiện được sự phân



Hình 5.10: Hình ảnh mô tả 4 ma trận kết quả mới sau khi lượt giảm các giá trị nhỏ hơn 5

bố mẫu (pattern) của các cửa sổ trượt nhỏ trên ma trận hình ảnh ban đầu. Ví dụ như cửa sổ đầu tiên thể hiện một đường chéo các số 1 (xem lại Hình 5.4) thì ma trận kết quả thể hiện phân bố đúng như bản chất ma trận hình ảnh gốc, với một đường chéo chạy dài từ đầu đến cuối. Hay như ma trận thứ 4, cửa sổ trượt thể hiện một hình tương tự mũi tên bị khuyết ở giữa và nằm lệch về bên phải (xem Hình 5.6). Ở ma trận gốc thì hình này chỉ xuất hiện đúng một lần tại góc trên cùng bên trái của hình ảnh.

Từ quan sát này, ta có thể hiểu được vì sao các cửa sổ trượt này được gọi là các *bộ lọc* (filter). Nhiệm vụ của chúng chính là tìm ra các đặc trưng của hình ảnh. Ở các lớp cao hơn, các bộ lọc đóng các vai trò khác nhau tùy theo yêu cầu của bài tóan như bộ phát hiện cạnh (edge detector), bộ phát hiện nhiễu (noise detector),...

5	3	4	4
4	5	3	5
4	5	5	3
2	3	2	5
3	3	4	5
3	5	4	4
3	3	3	4
3	3	3	4
5	5	2	5
3	3	3	4
2	2	5	3
2	3	4	3
5	2	4	4
4	3	4	4
3	4	4	4
3	3	3	3

Hình 5.11: Hình ảnh mô tả bản đồ thuộc tính từ việc ghép các ma trận kết quả

5.2.4 Bản đồ thuộc tính (Feature map)

Nếu ghép 4 ma trận kết quả ở Hình 5.9 lại với nhau, ta được một hình ảnh như Hình 5.11. Đây được gọi là *bản đồ thuộc tính* (feature map), được trích xuất sau quá trình tích chập của mạng CNN. Bản đồ thuộc tính là một đặc trưng nổi bật của mạng nơ-ron tích chập, bởi nó thể hiện được các *thuộc tính tiềm ẩn* (latent features) của hình ảnh đầu vào.

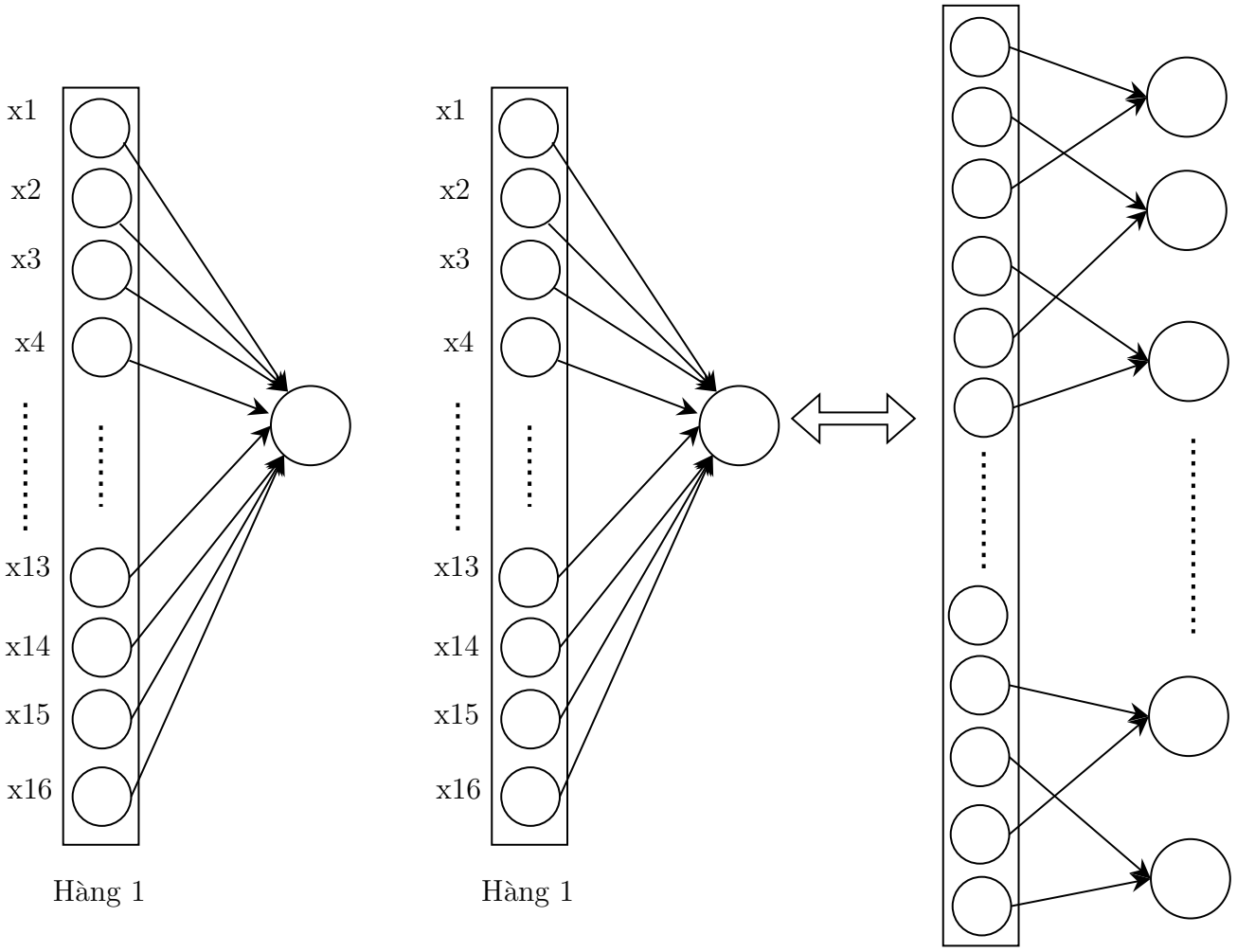
Cần lưu ý rằng *bản đồ thuộc tính* được sinh ra dựa trên các *bộ lọc* đã được định nghĩa trước (hay chính là các cửa sổ tích chập trong ví dụ trên). Do đó, điều quan trọng chính là làm sao định nghĩa được các *bộ lọc* này một cách phù hợp để áp dụng hiệu quả cho những bài toán cụ thể. Ví dụ như với bài toán nhận diện khuôn mặt người, *bộ lọc* sẽ là các bộ nhận diện mắt, mũi, miệng. Còn đối với bài toán nhận diện vạch kẻ đường, mạng nơ-ron cần học được các *bộ lọc* nhận diện đường thẳng,... Vấn đề này sẽ được bàn đến trong phần tiếp theo, khi ta cần sử dụng cơ chế học của *mạng nơ-ron nhân tạo* thông thường để có thể học được *bộ lọc*.

5.3 Mạng nơ-ron tích chập dưới góc nhìn của một mạng nơ-ron nhân tạo

Trong một mạng nơ-ron nhân tạo thông thường thì một mạng nơ-ron được cấu thành bởi các nút nơ-ron khác nhau nối tiếp nhau và qua quá trình xử lý thông tin sẽ tạo ra các nút nơ-ron ở tầng tiếp theo. Tương tự như vậy với mạng nơ-ron tích chập, ví dụ có một hình ảnh được biểu diễn dưới dạng ma trận đầu vào với kích thước 10×10 và một cửa sổ tích chập là ma trận 4×4 thì quá trình nhân tích chập tại vị trí đầu trên ma trân đầu vào 10×10 sẽ tạo ra được một phần tử của ma trận mới và đó cũng tương đương với một nút nơ ron được tạo ra sau khi thực hiện một phép biến đổi. Nút nơ-ron mới sẽ nối với 16 điểm trên ma trận đầu vào.

Hình 5.12 mô tả một ma trận đầu vào cùng với một cửa sổ và khi thực hiện phép tích chập trên một vị trí trên ma trận đầu vào sẽ tạo ra được một nút nơ-ron. Khi cho cửa sổ trượt qua hàng thứ hai sẽ thu được một nơ-ron khác và khi ta cho ma trận filter này trượt hết trên ma trận đầu vào thì sẽ tạo ra được 49 nút nơ ron khác nhau và mỗi nút nối với một ma trận 16 đơn vị.

Một đặc điểm quan trọng của mạng nơ-ron tích chập là *cơ chế chia sẻ trọng số* (shared weights). Có nghĩa là các trọng số trên mỗi bộ lọc phải giống nhau và các nơ-ron trong lớp ẩn đầu sẽ phát hiện chính xác điểm tương tự chỉ ở các vị trí khác nhau trong dữ liệu đầu vào. Việc làm này sẽ làm giảm tối đa số lượng các *tham số* (parameters), mỗi bản đồ đặc trưng sẽ giúp phát hiện thêm một vài đặc trưng khác. Với một ma trận hình ảnh đầu vào kích thước 10×10 như ở trên và 4 bộ lọc có ma trận kích thước 4×4 thì mỗi bản đồ thuộc tính cần $4 \times 4 = 16$ trọng số và số nơ-ron



Hình 5.12: Hình ảnh mô tả nhân tích chập một bộ lọc tạo ra các nơ-ron

được tạo ra ở lớp thứ hai là 49. Như vậy nếu có 4 bản đồ thuộc tính thì có $4 \times 16 = 64$ tham số. Với một mạng nơ-ron có kết nối đầy đủ thì chúng ta sẽ có $10 \times 10 \times 49 = 4900$ trọng số. Từ kết quả cho thấy sử dụng lớp tích chập sẽ cần số lượng tham số ít hơn nhiều lần so với lớp kết nối đầy đủ nhưng vẫn có thể rút ra các đặc trưng một cách hiệu quả.

Một khả năng khác của mạng nơ-ron tích chập là số tham số không phụ thuộc vào kích thước của đầu vào. Với những ma trận đầu vào có kích thước khác nhau và thông qua quá trình học theo phương pháp nơ-ron tích chập sẽ rút ra những thuộc tính ẩn mà ta có thể khó nhận thấy. Xét một ví dụ chúng ta có 10 bộ lọc và mỗi

một bộ lọc sẽ là một ma trận kích thước 3x3x3 và có một giá trị sai lệch (bias) là 1, chúng ta cần tính xem có bao nhiêu parameter sẽ được tạo ra từ việc sử dụng mạng nơ-ron tích chập? Để giải được bài toàn này thì cần tính số lượng parameters cần dùng mỗi bộ lọc rồi từ đó tính được kết quả.

- Số lượng tham số cho mỗi bộ lọc là 3x3x3 = 27
- Tổng số tham số cho mỗi bộ lọc là 27 +1 = 28
- Tổng số tham số cho 10 bộ lọc là 28 x 10 = 280

Như vậy từ ví dụ này thì cho dù dữ liệu đầu vào là bao nhiêu thì số lượng tham số được tạo ra cũng là 280, do đó số tham số có được từ mạng nơ-ron tích chập này không phụ thuộc vào kích thước đầu vào.

5.4 Triển khai một mạng CNN

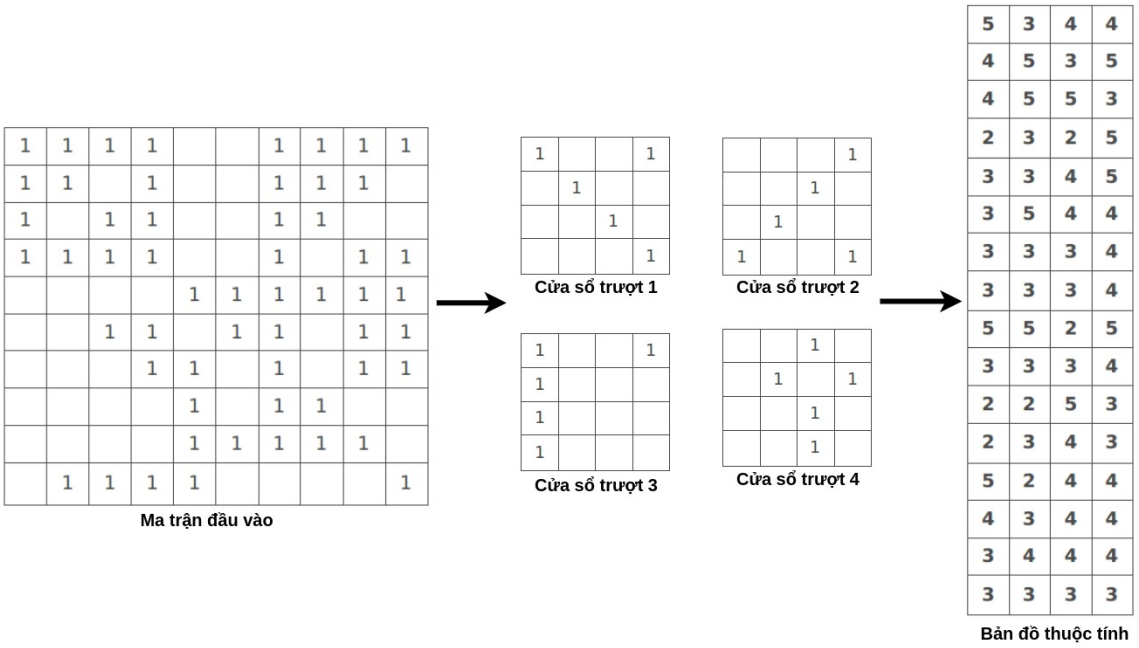
Một mạng CNN cơ bản bao gồm 3 bộ phận chính: *Tầng tích chập* (Convolution), *tầng tổng hợp* (Pooling) và tầng cuối cùng là *tầng kết nối đầy đủ* (Fully Connected).

Tầng tích chập là tầng quan trọng nhất và cũng là tầng đầu tiên của của mô hình CNN, có chức năng chính là phát hiện các đặc trưng cụ thể của input ban đầu (input ở đây có thể là một bức ảnh cây cối). Tầng này có các bộ phận chính là một ma trận đầu vào, *bản đồ thuộc tính* (Feature map) và các *cửa sổ trượt* (Convolution Filter). Tầng này bao gồm nhiều bản đồ thuộc tính đã được trích xuất ra các đặc tính cụ thể, mỗi bản đồ thuộc tính được tạo ra bằng cách quét input ban đầu qua cửa sổ trượt.

Tầng tổng hợp có mục đích làm giảm số lượng thông số mà ta phải tính toán, điều này giúp chúng ta giảm thời gian tính toán. Có 2 phương pháp tổng hợp thường gặp nhất là phép tổng hợp lớn nhất *(Max pooling)* và phép tổng hợp trung bình *(Average Pooling)*.

Cuối cùng tầng kết nối đầy đủ sẽ kết hợp các đặc điểm để ra được output của model.

Để dễ hình dung, ta sẽ mô tả mạng quá trình Convolution và Pooling ở ví dụ trước đó qua Hình 5.13 dưới đây.



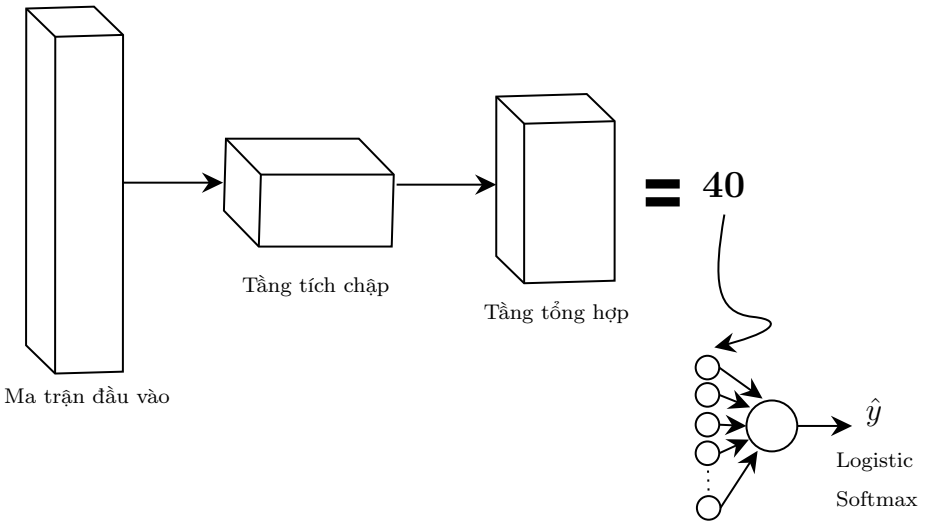
Hình 5.13: Ví dụ một quá trình Convolution và Pooling

Hình 5.13 ở trên biểu diễn ví dụ một ma trận đầu vào có kích thước 10×10, 4 cửa sổ trượt, mỗi cửa sổ trượt có kích thước $4x4$ và một bản đồ thuộc tính kết quả có kích thước 4×16. Cách tính toán bao gồm nhân ma trận đầu vào với các cửa sổ trượt, sau đó thực hiện Pooling để ra được kết quả bản đồ thuộc tính đã được trình bày ở phần trước.

Đây là ví dụ quá trình một ma trận ban đầu được thực hiện Convolution và Pooling để thu được một bản đồ thuộc tính, và quá trình này có thể lặp đi lặp lại nhiều lần nếu cần thiết. Có nghĩa là, bản đồ thuộc tính 4×16 ta có được trong ví dụ trên, có thể tiếp tục thực hiện qua các cửa sổ trượt khác, và sẽ thu được một bản đồ thuộc tính khác với cách tính toán tương tự. Quá trình này có thể lặp lại rất nhiều lần, và đến tầng cuối cùng thì ta sẽ học qua tầng Kết nối đầy đủ. Chúng ta cần chú ý rằng quá trình Convolution và Pooling phải được thực hiện ít nhất một lần và 2 tầng này luôn đi kèm với nhau, nhưng luôn cần một tầng kết nối đầy đủ sau cùng, điều này tương đương với việc triển khai một mạng Nơ-ron Network đơn giản cho một bài toán cần học.

Nhưng tại sao tầng cuối phải là fully connected? Ta có thể thấy thực ra việc học

CNN là nó học các cái trọng số của các cửa sổ trượt, nó cần học và cập nhật lại các trọng số này. Tương tự như những bài toán trước đó, cần phải có bài toán cho mạng CNN học. Việc học này sẽ được thực hiện ở tầng fully connected sau cùng, tương tự như một mạng nơ-ron cơ bản trình bày ở ví dụ trước.



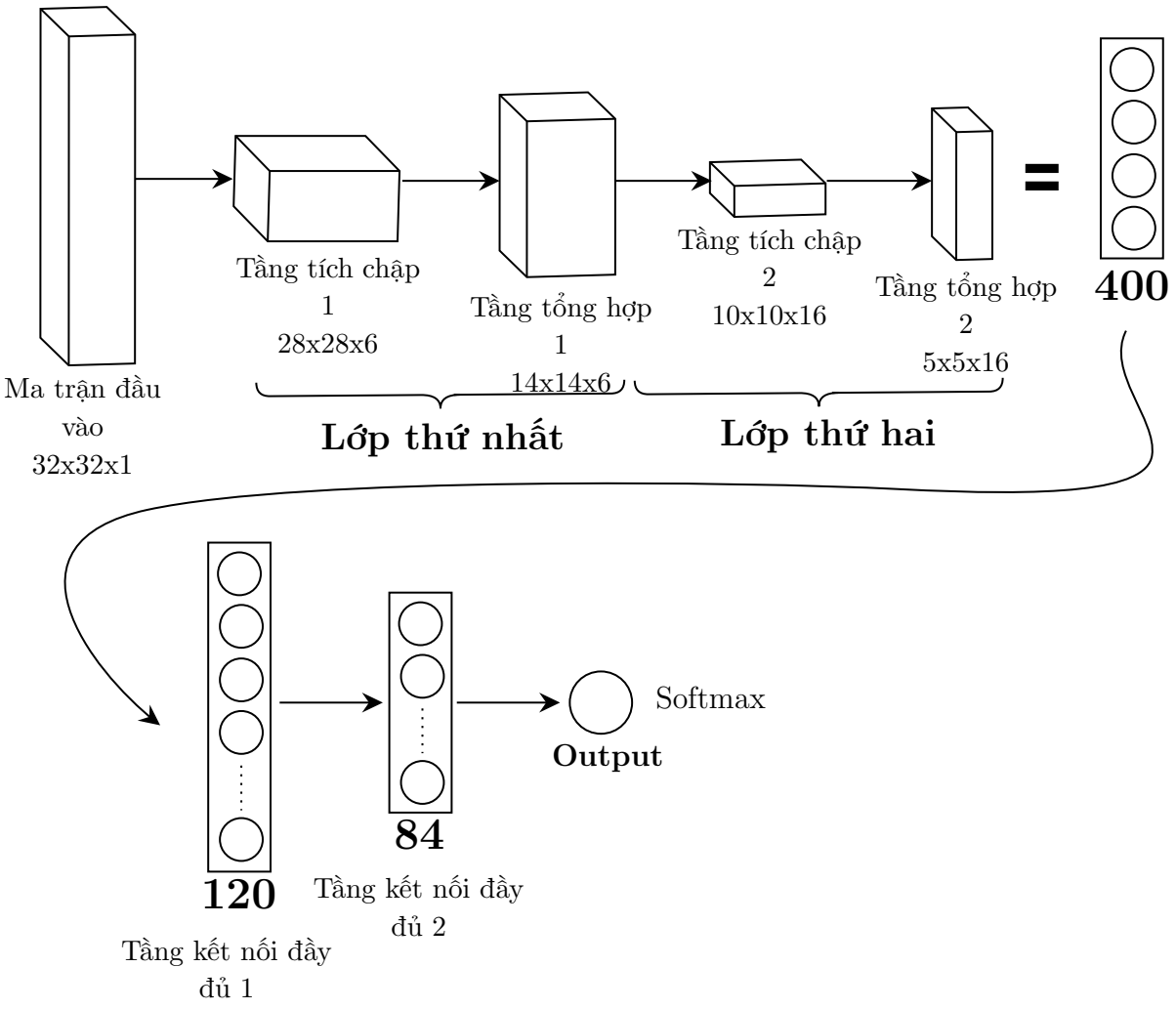
Hình 5.14: Ví dụ một mạng CNN

Hình 5.14 thể hiện ví dụ một mạng CNN, ta có một ma trận đầu vào, một lớp tích chập và một lớp tổng hợp, ta ví dụ 40 là kết quả tính toán có được sau khi qua lớp tổng hợp (một vector 40 chiều), và vector 40 chiều này sẽ được đưa vào lớp kết nối đầy đủ, sử dụng hàm logistic hoặc Softmax tùy theo mục đích, để thu được output sau cùng.

Và với kiến trúc mạng này, ta có thể ráp nhiều tầng CNN lại với nhau. Ta cũng chú ý rằng tầng kết nối đầy đủ có thể nhiều hơn 1 tầng để tăng khả năng học của bài toán cần triển khai, nhưng không nên quá nhiều vì sẽ làm mất đặc trưng của bài toán cần học, vì nếu quá nhiều sẽ dẫn đến việc tăng số lượng thông số đầu vào. Nếu số tầng fully connected quá nhiều thì chỉ riêng thông số của các tầng này cũng đã hơn các tầng trước đó, và điều này làm mất đi một ưu điểm của CNN - giảm số tham số cần học của một mạng học sâu.

Mạng LeNet5 là một trong những mạng CNN lâu đời và nổi tiếng nhất, được Yann LeCUn phát triển vào những năm 1998. Cấu trúc của mạng LeNet5 gồm 2 lớp Convolution và Maxpooling, 2 lớp Fully Connected và output là hàm Softmax. Hình

5.15 thể hiện ví dụ một mạng Lenet5.



Hình 5.15: Ví dụ một mạng Lenet-5

Một số thông tin về kiến trúc mạng Lenet5 như sau:

- Đầu vào là một ma trận có kích thước $32 \times 32 \times 1$, nghĩa là một tấm ảnh có chiều dài 32 pixel, chiều rộng ảnh là 32 pixel và số lượng kênh ảnh là 1 (ảnh đen trắng).
- Lớp thứ nhất bao gồm:
  - Tầng tích chập thứ 1: là một ma trận $5 \times 5 \times 3$, tốc độ stride là 1, đi qua

6 cửa sổ trượt có kích thước $5 \times 5 \times 1$, và output của nó là 1 ma trận có kích thước $28 \times 28 \times 6$.

  - Tầng tổng hợp thứ 1: Kích thước cửa sổ trượt là 1 ma trận 2×2, tốc độ stride là 2, số lượng cửa sổ trượt là 16, và output của nó là 1 ma trận có kích thước $14 \times 14 \times 6$.

- Lớp thứ hai bao gồm:

  - Tầng tích chập thứ 2: là một ma trận $5 \times 5 \times 6$, tốc độ stride là 1, đi qua 16 cửa sổ trượt và output của nó là 1 ma trận có kích thước $10 \times 10 \times 16$.

  - Tầng tổng hợp thứ 2: Kích thước cửa sổ trượt là 1 ma trận 2×2, tốc độ stride là 2 và output của nó là 1 ma trận có kích thước $5 \times 5 \times 16$.

- Số nút Output của lớp thứ 2 sẽ là $5 \times 5 \times 16 = 400$.
- Số nút Output của tầng kết nối đầy đủ thứ 1 sẽ là 120.
- Số nút Output của tầng kết nối đầy đủ thứ 2 sẽ là 84.

Với kiến trúc như thế này, thì việc áp dụng vào từng bài toán cụ thể sẽ được trình bày ở phần sau.

5.5 Sử dụng mạng CNN cho các bài toán NLP

5.5.1 Các cách tiếp cận cổ điển trước khi sử dụng CNN

Như đã trình bày, *Xử lý ngôn ngữ tự nhiên* (Natural Language Processing - NLP) là một lĩnh vực nghiên cứu rất phong phú và sâu sắc, với rất nhiều kỹ thuật để trích xuất thông tin từ các ngôn ngữ. *NLP* có rất nhiều ứng dụng phổ biến bao gồm *phân loại tài liệu* (Text Classification), *nhận dạng giọng nói* (Speech Recognition) và *dịch vụ dịch thuật* (Translation Services). Với các bài toán này, ta hoàn toàn có thể sử dụng phương pháp cổ điển là Bag-of-Words và TF-IDF để giải quyết. Để ví dụ, ta phân tích 3 câu sau:

- Câu 1: Bộ phim rất dở và dài.
- Câu 2: Bộ phim không dở và không dài.

- Câu 3: Bộ phim thật tệ và không vui.

Bag-of-words (BoW)

Đối với BoW, trước tiên chúng ta sẽ xây dựng một bộ từ điển bao gồm tất cả các từ khác nhau xuất hiện trong cả ba câu trên. Bộ từ điển này sẽ gồm 10 từ: [*'Bộ', 'phim', 'rất', 'dở', 'và', 'dài', 'không', 'thật', 'tệ', 'vui'*].

Bây giờ chúng ta có thể lấy từng từ này và đánh dấu sự xuất hiện của chúng trong ba câu ở trên như Hình 5.16. Điều này sẽ cung cấp cho chúng ta 3 vectơ cho 3 câu:

	1 Bộ	2 phim	3 rất	4 dở	5 và	6 dài	7 không	8 thật	9 tệ	10 vui	Tổng số từ trong câu
Câu 1	1	1	1	1	1	1	0	0	0	0	6
Câu 2	1	1	0	1	1	1	2	0	0	0	7
Câu 3	1	1	0	0	1	0	1	1	1	1	7

Hình 5.16: Bảng biểu diễn sự xuất hiện của các từ trong câu

$$\text{Vector biểu diễn cho câu 1: } \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\text{Vector biểu diễn cho câu 2: } \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 2 & 0 & 0 & 0 \end{bmatrix}$$

$$\text{Vector biểu diễn cho câu 3: } \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Trong ví dụ trên, chúng ta có thể có các vector có độ dài 10. Tuy nhiên, chúng ta bắt đầu đối mặt với các vấn đề khi bắt gặp các câu mới:

1. Nếu các câu mới chứa các từ mới, thì kích thước từ vựng của chúng ta sẽ tăng lên và do đó, độ dài của các vector cũng sẽ tăng theo.
2. Ngoài ra, các vectơ cũng sẽ chứa nhiều 0, do đó dẫn đến một ma trận thưa (đó là những gì chúng tôi muốn tránh)
3. Chúng ta chưa thể biểu diễn thông tin về ngữ pháp của các câu cũng như về thứ tự của các từ trong văn bản.

TF-IDF

Chúng ta sẽ lại sử dụng cùng một bộ từ điển mà chúng ta đã xây dựng trong mô

hình Bag-of-Words để chỉ ra cách tính toán đối với tần suất xuất hiện các từ của ba câu trên. Sau khi tính toán theo phương pháp TF-IDF, ta được bảng như sau:

Từ	Câu 1	Câu 2	Câu 3	TF 1	TF 2	TF 3	IDF	TF-IDF 1	TF-IDF 2	TF-IDF 3
Bộ	1	1	1	1/6	1/7	1/7	0.00	0.000	0.000	0.00
phim	1	1	1	1/6	1/7	1/7	0.00	0.000	0.000	0.00
rất	1	0	0	1/6	0	0	0.48	0.080	0.000	0.00
dở	1	1	0	1/6	1/7	0	0.18	0.029	0.025	0.00
và	1	1	1	1/6	1/7	1/7	0.00	0.000	0.000	0.00
dài	1	1	0	1/6	1/7	0	0.18	0.029	0.025	0.00
không	0	2	1	0	2/7	1/7	0.18	0.000	0.050	0.03
thật	0	0	1	0	0	1/7	0.48	0.000	0.000	0.07
tệ	0	0	1	0	0	1/7	0.48	0.000	0.000	0.07
vui	0	0	1	0	0	1/7	0.48	0.000	0.000	0.07

Hình 5.17: Bảng biểu diễn cách tính TF-IDF đối với ba câu trên

Từ cách tính toán của TF-IDF trong Hình 5.17 trên, ta nhận thấy các từ như *'Bộ', 'phim', 'và'* là những từ xuất hiện trong cả ba câu nhưng ít quan trọng, trong khi đó các từ còn lại quan trọng hơn nên có giá trị cao hơn. Vì vậy, những từ có giá trị cao hơn sẽ quan trọng hơn và những từ có giá trị thấp hơn thì ít quan trọng hơn. Bên cạnh đó, TF-IDF cũng cung cấp các giá trị lớn hơn cho các từ ít thường xuyên hơn và cao khi cả hai giá trị IDF và TF đều cao, nghĩa là từ này hiếm trong tất cả các tài liệu được kết hợp nhưng thường xuyên trong một tài liệu.

Từ ví dụ cho cả hai kỹ thuật Bag-of-words và TF-IDF, ta rút ra được kết luận như sau:

- Bag of Words chỉ tạo ra một tập các vectơ chứa số lần xuất hiện từ trong tài liệu (đánh giá), trong khi mô hình TF-IDF chứa thông tin về các từ quan trọng hơn và cả những từ ít quan trọng hơn.
- Các vectơ Bag of Words rất dễ để giải thích. Tuy nhiên, TF-IDF thường hoạt động tốt hơn trong các mô hình học máy.

5.5.2 Mô hình n-gram trong NLP

Mặc dù cả Bag-of-Words và TF-IDF đều được sử dụng phổ biến theo cách riêng của của từng kỹ thuật, nhưng vẫn còn một vấn đề khá lớn, trong đó việc hiểu ngữ nghĩa và ngữ cảnh của các từ chính là vấn đề được quan tâm nhất. Cụ thể, vấn đề của Bag-of-words chính là nó chỉ quan tâm đến sự xuất hiện của từ đó trong câu mà không quan tâm đến thứ tự các từ trong câu đó. Còn TF-IDF, để phát hiện được sự giống nhau về ngữ nghĩa của hai từ đồng nghĩa, hoặc để dịch một câu sang ngôn ngữ khác, nó hầu như không làm được hoặc phải đòi hỏi cần nhiều thông tin hơn về các từ trong câu đó.

Từ những sự hạn chế đó, các kỹ thuật Word Embedding như Word2Vec, Continuous Bag of Words (CBOW), Skipgram,... sẽ được sử dụng thay thế chúng để biểu diễn thông tin n-gram. Mô hình n-gram là một trong những mô hình đơn giản nhất nhưng cũng quan trọng nhất để gán xác suất cho câu và cụm từ.

Trong lĩnh vực ngôn ngữ học hoặc tính toán xác suất, n-gram là một chuỗi liên tục của n mục được chứa trong một văn bản hoặc một câu nói. Các mục này có thể là âm vị, âm tiết, chữ cái hoặc từ. Trong NLP, n-gram được hiểu như một chuỗi gồm n. Theo cách hiểu đó, 2-gram là một chuỗi gồm 2 từ như "*tôi đi*", "*đi học*", và 3-gram là một chuỗi gồm 3 từ như "*tôi đi học*".

Chúng ta bắt đầu với một xác suất có điều kiện $P(w|h)$, biểu diễn cho xác suất của một từ w với điều kiện cho trước là một chuỗi từ h. Giả sử h là câu "*Hôm nay tôi đến thăm*", và chúng ta muốn tính xác suất của từ tiếp theo trong câu là từ "*bác*", thì xác suất này sẽ được biểu diễn là:

$$P(bác|Hôm\ nay\ tôi\ đến\ thăm)$$

Một cách để ước tính xác suất này là từ số lượng lặp lại tương đối: lấy một tập văn bản rất lớn, đếm số lần chúng ta thấy sự xuất hiện của câu "*Hôm nay tôi đến thăm*", và đếm số lần mà từ tiếp theo của câu trên là "*bác*". Phương pháp này sẽ trả lời cho câu hỏi có bao nhiêu lần từ "*bác*" xuất hiện ngay phía sau câu "*Hôm nay tôi đến thăm*":

$$P(bác|Hôm\ nay\ tôi\ đến\ thăm) = \frac{C(Hôm\ nay\ tôi\ đến\ thăm\ bác)}{C(Hôm\ nay\ tôi\ đến\ thăm)} \tag{5.2}$$

Tới đây, chúng ta có thể mường tượng ra rằng cách tính này hầu như là không khả thi khi thực hiện điều này trên toàn bộ kho văn bản, đặc biệt là kho văn bản có kích thước rất lớn. Để giải quyết vấn đề này, chúng ta sẽ sử dụng mô hình n-gram như một giải pháp khắc phục. Ở đây, thay vì tính xác suất trên toàn bộ kho văn bản, chúng ta sẽ ước chừng nó chỉ với một vài từ phía trước từ được dự đoán.

Trong mô hình bigram, để xấp xỉ xác suất xuất hiện của một từ đứng sau một chuỗi từ (được biểu diễn là $P(w_n|w_1^{n-1})$), nó sẽ chỉ sử dụng xác suất có điều kiện của một từ phía trước đó (được biểu diễn là $P(w_n|w_{n-1})$). Nói cách khác, ta sẽ tính

$$P(\textit{bác}|\textit{thăm})$$

thay vì tính cho

$$P(\textit{bác}|\textit{Hôm nay tôi đến thăm})$$

Khi chúng ta sử dụng mô hình bigram để dự đoán xác suất có điều kiện của từ tiếp theo, chúng ta sẽ có xấp xỉ sau:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1}) \tag{5.3}$$

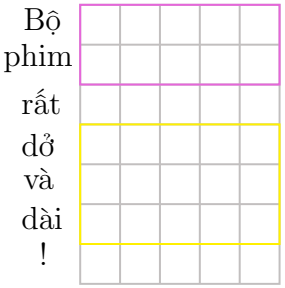
Xác suất có điều kiện này là cơ sở để chúng ta xây dựng *mô hình ngôn ngữ*, như sẽ được thảo luận tiếp theo.

5.5.3 Sử dụng CNN vào bài toán NLP với phương pháp Word Embedding

Với phương pháp Word Embedding, mỗi từ có thể biểu diễn thành một vector, từ đó một văn bản có thể giải thích thành một ma trận như hình bên dưới:

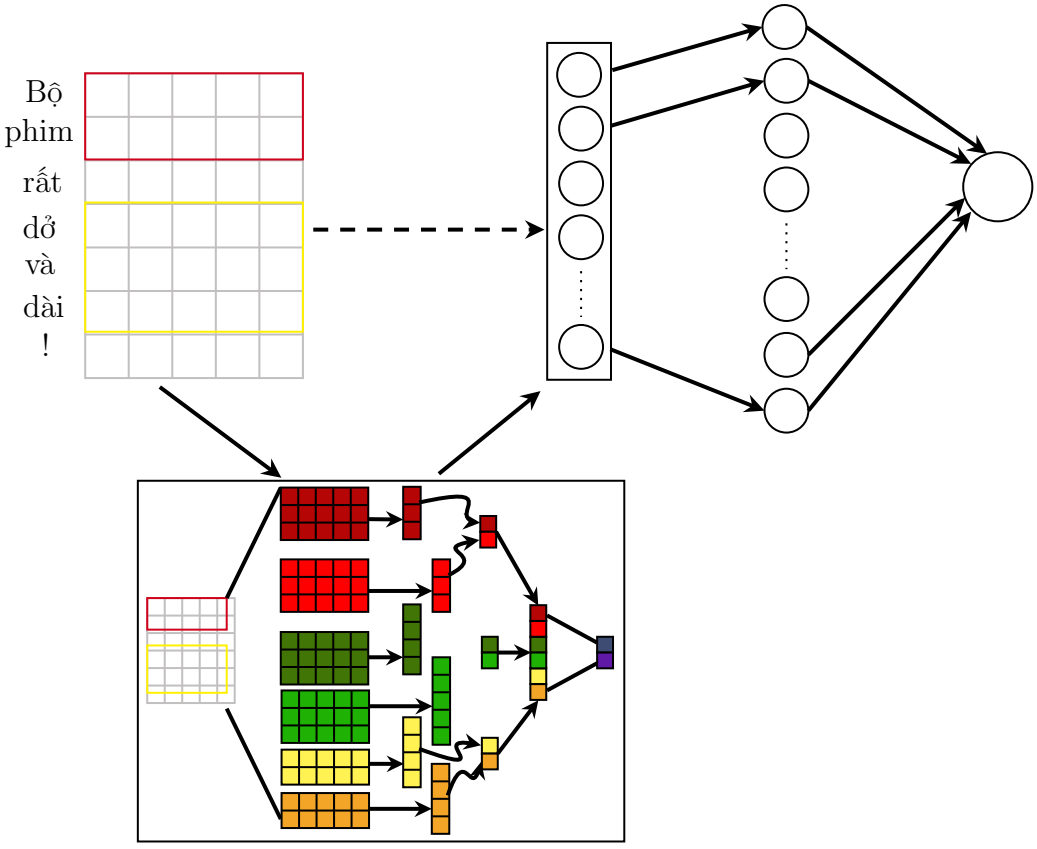
Từ Hình 5.18 trên, ta để ý rằng hình chữ nhật màu đỏ sẽ biểu diễn một bi-gram trên ma trận này. Tương tự, hình chữ nhật màu vàng biểu diễn một tri-gram. Như vậy một bộ lọc ma trận (filter) $2 \times k$ là phù hợp để biểu diễn và rút trích các latent feature của các bi-gram, tương tự một bộ lọc ma trận $3 \times k$ cho tri-gram.

Với ma trận từ được biểu diễn như Hình 5.18 ở trên, ta hoàn toàn có thể sử dụng như là input cho một mạng nơ-ron như bình thường, tuy nhiên với cách tiếp cận



Hình 5.18: Biểu diễn mỗi từ trong câu thành vector bằng Word Embedding

mới bằng các kỹ thuật Word Embedding, ta có thể sử dụng thêm tầng CNN để rút trích các latent features, được thể hiện như Hình 5.19 ở dưới, và cách biểu diễn kinh điển của một tầng CNN sẽ được trình bày trong case study tiếp theo.



Hình 5.19: Sử dụng thêm tầng CNN trong cách tiếp cận mới

5.5.4 Case study: sử dụng CNN cho bài toán phân tích cảm xúc (sentiment analysis)

Phân tích cảm xúc là quá trình phân tích, đánh giá quan điểm, cảm xúc của một người về một đối tượng nào đó qua lời nói hoặc văn bản (quan điểm có thể mang tính tiêu cực, tích cực hay bình thường,..). Sử dụng các kỹ thuật từ NLP, *sentiment analysis* sẽ xem xét biểu thức của người dùng và lần lượt liên kết cảm xúc với những gì người dùng đã cung cấp. Tuy nhiên, cũng có trường hợp tùy thuộc vào nhiều yếu tố tác động mà một câu nói sẽ mang nhiều ý nghĩa cảm xúc khác nhau. *sentiment analysis* được áp dụng rộng rãi cho việc phân tích đánh giá và phản hồi khảo sát, phương tiện truyền thông xã hội và trực tuyến và được sử dụng nhiều trong lĩnh vực dịch vụ chăm sóc khách hàng.

Trong phần này, chúng ta sẽ đi vào chi tiết một case study là sử dụng CNN cho bài toán *sentiment analysis*. Dưới đây là một sơ đồ biểu diễn kiến trúc CNN cho bài toán phân loại cảm xúc:

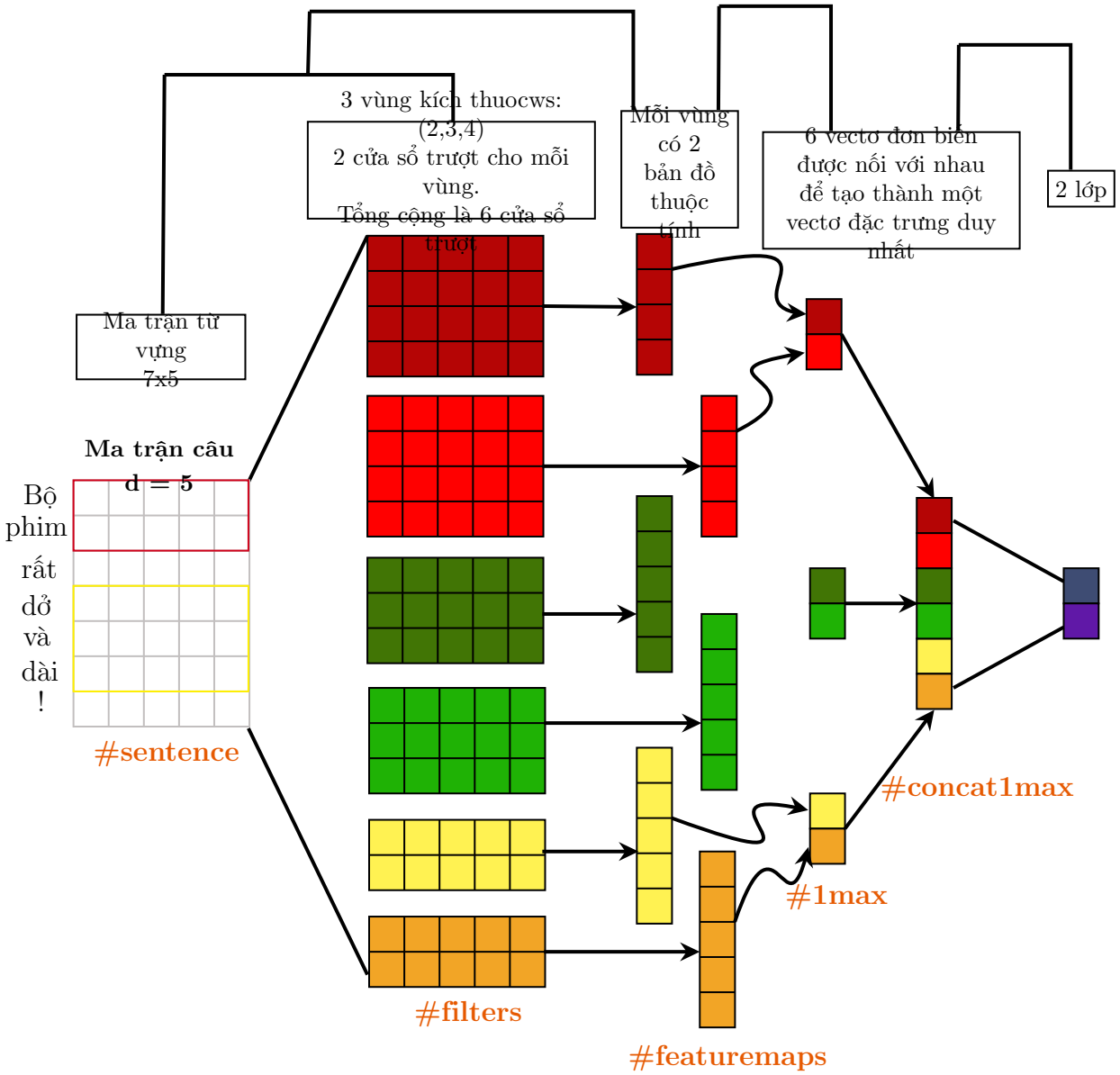
Để dễ hiểu, chúng ta sẽ chia sơ đồ trên ra thành 5 phần: #sentence, #filters, #featuremaps, #1-max và #concat-1-max.

#sentence

Ví dụ là câu *'Bộ phim rất dở và dài !'* tổng cộng có 7 từ trong câu (dấu chấm than vẫn được tính là một từ). Ở đây chúng ta chọn 5 là chiều của các vector từ. Chúng ta hãy biểu thị độ dài của câu là s và d là kích thước của vector từ, do đó chúng ta có một ma trận câu có hình dạng sxd là 7x5.

#filters

Một trong những đặc tính mong muốn của CNN là nó bảo toàn hướng không gian 2D trong thị giác máy tính. Thay vì 2 chiều, các văn bản có cấu trúc một chiều để biểu thị trình tự các từ. Vì chúng ta đã biểu diễn các từ trong câu ví dụ bằng các vector từ 5 chiều, do đó chúng ta sẽ sửa một chiều của bộ lọc để khớp với vector từ và thay đổi kích thước vùng *h (region size)*. *Region size* đề cập đến số hàng của từ trong ma trận câu sẽ được lọc. Cụ thể ở đây,chúng ta sẽ chọn kích thước cho các bộ lọc được biểu diễn bởi các ma trận có số cột là 7, số hàng sẽ tùy biến (2, 3, 4) và chúng ta sẽ áp dụng 2 bộ lọc cho từng vùng, sau colvolution đầu tiên ta sẽ có 6 bộ



Hình 5.20: Sơ đồ biểu diễn kiến trúc CNN cho bài toán phân loại cảm xúc

lọc.

#featuremaps

Đối với phần này, chúng ta sẽ sử dụng CNN để thực hiện các phép toán tích chập, các công thức tính toán này đã được trình bày ở các phần trước đó.

#1-max

Ta đã biết được rằng chiều của c phụ thuộc cả s và h, nói cách khác, nó sẽ khác nhau giữa các câu có độ dài khác nhau và các bộ lọc có kích thước vùng khác nhau. Để giải quyết vấn đề này, chúng ta sẽ sử dụng hàm gộp 1-max và trích xuất số lớn nhất từ mỗi vector c.

#concat-1-max

Sau khi thực hiện phép gộp 1-max, chúng ta chắc chắn có một vector có độ dài cố định gồm 6 phần tử (= *số bộ lọc = số bộ lọc cho mỗi kích thước vùng (2)* x *số lượng kích thước vùng được xem xét (3)*). Vector có chiều dài cố định này sau đó có thể được đưa vào một lớp softmax *(fully-connected)* để thực hiện phân loại. Sự mất mát đến từ sự phân loại này sau đó sẽ được lan truyền ngược trở lại vào các tham số sau đây như là một phần của việc học:

- Ma trận w (ma trận dùng để tạo ra ma trận o).
- Bias được thêm vào o để tạo ra ma trận c.
- Các vector từ.

Lớp softmax cuối cùng sau đó nhận vector tính năng này làm đầu vào và sử dụng nó để phân loại câu. Ở đây chúng ta giả định là phân loại nhị phân và do đó sẽ tạo ra hai trạng thái đầu ra có thể nhất.

Sau khi rút trích ra được các features, ta sẽ có vector output, và với vector này ta vẫn có thể sử dụng được cho các phương pháp học máy khác như SVM (Support Vector Machine), Naive Bayes,... Tuy nhiên, trong trường hợp này, nhiều mô hình hiện thực sử dụng một tầng fully connected (tức là qua một mạng Nơ-ron đơn giản). Vì khi này chúng ta sẽ có một mạng *end-to-end*, đầu vào và đầu ra của hệ thống đều được kết nối qua một kiến trúc mạng nơ-ron. Nhờ vậy, sau khi có kết quả Loss ở tầng cuối cùng, hệ thống có thể cập nhật toàn bộ các trọng số từ đầu bằng cơ chế backpropagation. Toàn bộ các bước đều có thể học theo cơ chế Backpropagation này, do đó trọng số của các filter tương ứng với n-gram, *trigram*, *bigram*,... đều được học và cập nhật liên tục. Phần hiện thực mọi người có thể tham khảo một số mã nguồn mở như Github.

Trong thực tế, để lựa chọn kích thước và số lượng các bộ lọc cho các bài toán NLP

áp dụng mạng CNN, chúng ta thường phải tham khảo từ các bài báo liên quan nổi tiếng trong ngành. Ở đây, chúng tôi giới thiệu một trong các bài báo tiêu biểu về việc lựa chọn các siêu tham số *(hyperparameters)*như trong ví dụ trên, đó là bài báo *'Convolutional Neural Networks for Sentence Classification'* của Yoon Kim (2014) (Kim, 2014).

Trong bài báo, tác giả đã mô tả một loạt các thí nghiệm với các mạng CNN được xây dựng trong mô hình word2vec. Mặc dù có một ít sự điều chỉnh các *hyperparameter*, nhưng đã thu được một mạng CNN đơn giản với một lớp chập thực hiện rất tốt. Kết quả của tác giả đã khẳng định cho một luận điểm rõ ràng rằng việc huấn luyện trước *(pre-training)* các vector từ không được giám sát *(unsupervised)* là một phần quan trọng trong việc ứng dụng học sâu cho NLP.

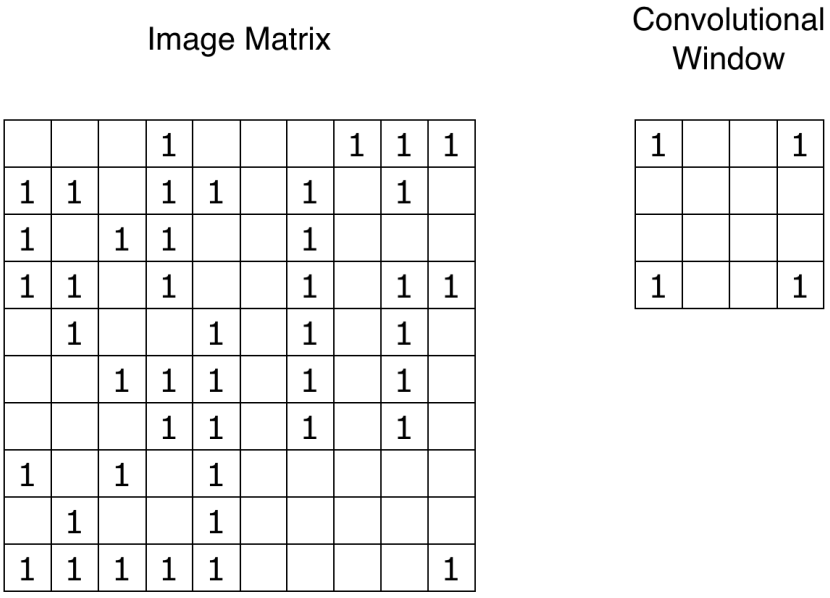
Cụ thể, qua tất cả các thực nghiệm về việc lựa chọn *hyperparameter* trong bài báo, tác giả đã sử dụng: đơn vị tuyến tính đã chỉnh sửa *(rectified linear units)*, cửa sổ bộ lọc (h) 3, 4, 5 đối với mỗi 100 feature map, tỉ lệ bỏ qua *(dropout rate)* là 0.5, ràng buộc L_2 cho các hàng của ma trận *softmax* s là 3 và mini-batch size = 50. Các giá trị này được chọn thông qua tìm kiếm dạng lưới *(grid search)* trên bộ dev SST-2 (Stanford Sentiment Treebank - đây là một phần mở rộng của thông số MR, Movie review với mỗi đánh giá tích cực/tiêu cực bằng một câu, nhưng có thêm các phân tách train/dev/test được cung cấp và các nhãn đánh giá *(very positive, positive, neutral, negative, very negative)*).

Đối với thông số chiều cho các vector từ, ở đây tác giả tin rằng việc khởi tạo vector từ với những từ thu được từ mô hình ngôn ngữ nơ-ron không giám sát *(unsupervised neural language model)* là một phương pháp phổ biến để cải thiện hiệu suất khi không có bộ dữ liệu được gán nhãn lớn. Tác giả sử dụng các vector Word2Vec có sẵn công khai được huấn luyện trên 100 tỷ từ trên Google News. Các vector này có chiều là 300 và được huấn luyện bằng cách sử dụng kiến trúc CBOW *(continuous bag-of-words)*. Các từ không có trong tập hợp *pre-trained words* sẽ được khởi tạo ngẫu nhiên.

Trong bài báo này, tác giả cũng nhấn mạnh rằng trong suốt quá trình huấn luyện, cần luôn tiếp tục kiểm tra hiệu suất trên bộ dev set và chọn trọng số chính xác cao nhất *(highest accuracy weights)* cho lần đánh giá cuối cùng.

5.6 Bài tập

Bài tập 5.6.1. Cho ma trận hình ảnh kích thước 10×10 và một cửa sổ tích chập kích thước 4×4 như Hình 5.21. Hãy cho biết ma trận kết quả cụ thể và số chiều của ma trận này sau khi thực hiện phép tích chập và phép gộp max pooling.



Hình 5.21: Ma trận hình ảnh và cửa sổ tích chập trong bài tập 5.6.1

Bài tập 5.6.2. Cho một mạng nơ ron có 3 tầng, với tầng thứ nhất là một ma trận kích thước 10×10 và có 2 cửa sổ trượt ở tầng 1 như Hình 5.22 và ở tầng thứ hai có 1 cửa sổ trượt như Hình 5.23. Hãy cho biết ở tầng thứ 3 sẽ là vector có bao nhiêu chiều?

Bài tập 5.6.3. Với cách làm cho Bài tập 5.6.2 thì ta vẫn cần ma trận ban đầu có số chiều cố định. Nếu chiều dài không cố định thì cần xử lý như thế nào?

Bài tập 5.6.4. Cho câu sau đây:

Bộ phim rất dở và dài !

One-hot vector của câu này sẽ là

Image Matrix

0	0	1	1	0	1	1	0	0	1
1	1	1	0	0	0	0	1	1	0
1	1	0	0	1	1	1	0	1	1
1	0	0	0	0	0	1	1	0	0
0	0	0	0	1	0	0	1	0	1
1	1	0	0	0	1	1	1	0	1
0	0	0	0	1	0	0	0	1	1
0	0	0	1	1	1	0	0	0	0
0	1	1	1	0	0	1	1	0	0
1	1	0	1	1	0	1	1	1	1

Convolutional Window

1	0	0	1
0	1	0	0
1	0	1	0
0	0	0	1

1	0	0	1
0	0	0	0
1	0	0	0
0	0	0	1

Hình 5.22: Ma trận hình ảnh và 2 cửa sổ tích chập ở tầng đầu tiên trong bài tập 5.6.2

Convolutional Window

1	0	0	1
0	1	1	0
0	0	1	0
1	0	0	1

Hình 5.23: Cửa sổ tích chập cho tầng thứ 2 trong Bài tập 5.6.2

Ma trận Embedding của câu này sẽ là $\begin{bmatrix} 8 & 2 & 1 & 9 \\ 2 & 4 & 7 & 0 \\ 1 & 3 & 5 & 8 \\ 0 & 1 & 9 & 4 \\ 7 & 8 & 1 & 2 \\ 1 & 5 & 8 & 9 \\ 5 & 1 & 5 & 2 \end{bmatrix}$

	Bộ	phim	rất	dở	và	dài	!
Bộ	1	0	0	0	0	0	0
phim	0	1	0	0	0	0	0
rất	0	0	1	0	0	0	0
dở	0	0	0	1	0	0	0
và	0	0	0	0	1	0	0
dài	0	0	0	0	0	1	0
!	0	0	0	0	0	0	1

Hãy cho biết ma trận biểu diễn câu này như thế nào ?

Bài tập 5.6.5. Cho ma trận ban đầu là $\begin{bmatrix} 6 & 5 & 9 & 1 \\ 3 & 4 & 7 & 1 \\ 1 & 7 & 5 & 1 \\ 0 & 7 & 9 & 0 \\ 4 & 2 & 2 & 4 \\ 7 & 5 & 9 & 8 \\ 1 & 1 & 1 & 0 \end{bmatrix}$

Ma trận 2-gram là $\begin{bmatrix} 1 & 5 & 9 & 3 \\ 3 & 8 & 0 & 2 \end{bmatrix}$

Ma trận 3-gram là $\begin{bmatrix} 0 & 7 & 9 & 4 \\ 2 & 9 & 7 & 5 \\ 3 & 3 & 0 & 7 \end{bmatrix}$

Sử dụng Max pooling, hãy cho biết bản đồ thuộc tính sau cùng được tạo ra.

5.7 Phụ lục

Dưới đây là một số thuật ngữ có đề cập trong chương 5:

Bảng 5.1: Bảng các thuật ngữ

Thuật ngữ	Ý nghĩa
Convolutional Neural Network	mạng nơ-ron tích chập
latent feature extraction	rút trích thuộc tính ẩn
fully connected	kết nối đầy đủ
shared weight	chia sẻ trọng số
convolution	phép tích chập
pooling	phép gộp
max pooling	phép gộp lấy giá trị lớn nhất
average pooling	phép gộp lấy giá trị trung bình
sum pooling	phép gộp lấy giá trị tổng
distribution pattern	phân bố mẫu
filter	bộ lọc
feature map	bản đồ thuộc tính
sentiment analysis	phân tích cảm xúc
language model	mô hình ngôn ngữ

Chương 6

Mạng nơ-ron truy hồi

6.1 Giới thiệu về dữ liệu chuỗi

6.1.1 Dữ liệu chuỗi là gì

Sequence data là một chuỗi dữ liệu gồm nhiều phần tử có trật tự. Khác với kiểu dữ liệu truyền thống khi các mẫu là phi thứ tự và độc lập lẫn nhau, với sequence data, thứ tự xuất hiện trước sau của các mẫu dữ liệu cũng mang thông tin nhất định. Một số ví dụ cho dữ liệu có thứ tự như:

- Dữ liệu về chuỗi văn bản: Trong ngôn ngữ tự nhiên, thứ tự xuất hiện của các từ đóng vai trò rất quan trọng trong việc truyền tải ý nghĩa của một câu hay đoạn văn nào đó.

  Lấy ví dụ với một câu tiếng Anh "I like apples but I hate bananas", tập từ xuất hiện trong câu bao gồm { "I", "likes", "apple", "but", "I", "hate", "bananas" }.

  Giả sử bỏ qua thông tin về thứ tự của câu, ta có các câu cùng tập từ nhưng khác thứ tự như "I like bananas but I hate apples" hoàn toàn trái ngược nghĩa của câu ban đầu, hay trong trường hợp khác như "I bananas hate but I apples like" lại hoàn toàn vô nghĩa.

  Vì vậy, thông tin về thứ tự là vô cùng quan trọng xét riêng trong ngữ cảnh dữ

liệu chuỗi văn bản.

- Dữ liệu về time-series:

  Giả sử ta có dữ liệu về nhiệt độ đúng 12h trưa mỗi ngày liên tục trong vòng một năm. Ngoài thông tin về mẫu dữ liệu đó (bao nhiêu độ), thông tin về thời gian mẫu dữ liệu được thu thập cũng không kém phần quan trọng. Dựa vào thời gian, ta có thể rút ra được các kiến thức như vào giai đoạn nào trong năm nhiệt độ cao/thấp, hay nếu như nắng nóng liên tục trong một tháng thì nhiệt độ hôm nay thế nào,...

  Time-series data là một trường hợp cụ thể của sequence data khi các mẫu dữ liệu được gắn thêm cả thông tin về thời gian.

- Dữ liệu về audio/video:

  Ở mặt dữ liệu lưu trữ trên máy tính, audio là một chuỗi các số đại diện cho các tham số của đoạn audio tại các thời điểm về tần số, biên độ, ... Video lại được hiểu là một chuỗi các hình ảnh (frame) liên tiếp nhau. Xét với bộ não con người, để hiểu được một đoạn âm thanh hay xem một đoạn video nào đó, ta cần tiếp nhận chúng (qua tai nghe, mắt nhìn) một cách tuần tự. Ta không thể nghe và hiểu được một bài hát bị xáo trộn lời, hay xem một bộ phim từ đoạn cuối trở ngược lại đầu. Thứ tự xuất hiện của dữ liệu trong audio và video là quan trọng. Audio và video cũng có thể được liệt vào dạng time-series data.

Một đặc điểm đặc biệt của sequence data so với dữ liệu dạng mẫu phi trật tự cũ, đó là một chuỗi dữ liệu có thể có độ dài bất định. Một câu nói có thể có 2 hay 3 từ, nhưng cũng có thể chứa hàng chục từ. Hay một đoạn video có thể kéo dài 10 giây, nhưng cũng có bộ phim kéo dài hàng giờ đồng hồ. Tính bất định về độ dài là một trong những đặc trưng của dữ liệu dạng sequence. Với dữ liệu dạng độ dài xác định, nhiều kĩ thuật như PCA hay LDA có thể được áp dụng để thu giảm số chiều với ít mất mát dữ liệu. Tuy nhiên, vì bản chất có thứ tự, dữ liệu dạng sequence khó có thể được thu giảm về độ dài xác định mà vẫn giữ được thông tin hoàn chỉnh cũng như bản chất của dãy ban đầu.

Dữ liệu dạng sequence không quá khó tìm. Và lẽ dĩ nhiên, các ứng dụng về phân tích dữ liệu nói chung và machine learning nói riêng cũng đã được nghiên cứu và để

xuất để xử lý kiểu dữ liệu này. Một số bài toán trên sequence data có thể kể đến như:

- **Dạng one-to-many:**

  Với dữ liệu đầu vào cố định, dữ liệu đầu ra dạng sequence.

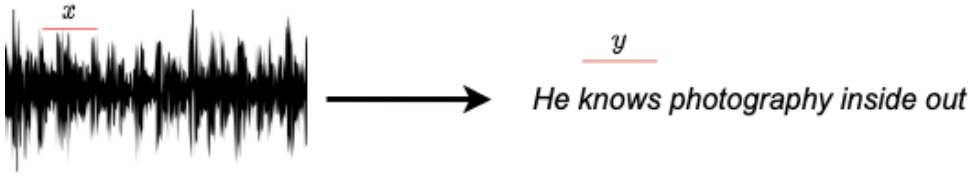
  Bài toán sinh nhạc tự động: Chỉ với một thông tin đầu vào (nốt nhạc đầu tiên, kiểu nhạc, etc), yêu cầu đặt ra là sinh một sequence dữ liệu tiếp theo là các nốt nhạc phù hợp:

Hình 6.1: Sinh nhạc tự động

- **Dạng sequence-to-sequence, hay many-to-many bất đối xứng:**

  Dữ liệu đầu vào và đầu ra đều là dạng sequence, tuy nhiên có độ dài khác nhau và không có sự bắt cặp input-output.

  Trong bài toán nhận diện giọng nói, yêu cầu đặt ra là chuyển dữ liệu từ dạng audio sang dạng text. Ta có một sequence đầu vào (audio) và một sequence đầu ra (chuỗi văn bản) với độ dài khác nhau. Bài toán speech recognition được liệt vào dạng sequence-to-sequence.



Hình 6.2: Bài toán nhận diện giọng nói

Một ví dụ khác thường gặp là bài toán dịch máy:

Cho đầu vào là một chuỗi văn bản ở ngôn ngữ A, hệ thống phải trả dữ liệu đầu ra là một chuỗi văn bản ở ngôn ngữ B nào đó. Rõ ràng rằng cũng một

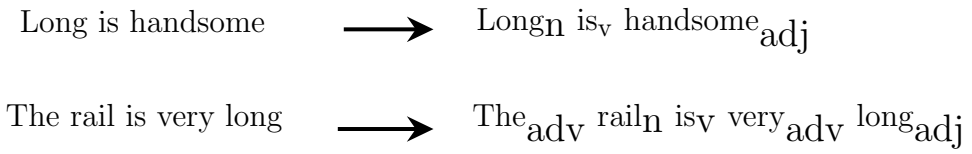
câu nói, nhưng ở các ngôn ngữ khác nhau sẽ có cách biểu diễn khác nhau (độ dài câu, số từ, ngữ pháp, ...). Bài toán Machine translation cũng được liệt vào dạng many-to-many bất đối xứng. Cần phân định rõ bài toán dịch máy sequence-to-sequence và word-by-word, word-by-word translation chỉ đơn giản dịch từng từ ở câu cũ và ghép lại thành câu mới.

Hình 6.3: Dịch máy

- **Dạng many-to-many đối xứng:**

  Dữ liệu đầu vào và đầu ra đều là dạng sequence, có độ dài như nhau và có sự bắt cặp tương ứng input-output ở từng bước.

  Xét bài toán nhận diện loại từ trong một câu. Vì mỗi từ chỉ thuộc về một loại duy nhất, nên ta sẽ có các cặp input-output đối xứng tương ứng. Hơn nữa, loại từ của một câu cũng phụ thuộc vào các từ đứng xung quanh đó, hay thứ tự xuất hiện của các từ trong câu. Vì vậy, ta liệt bài toán này vào dạng many-to-many đối xứng.



Hình 6.4: Phân loại từ trong một câu

- **Dạng many-to-one:**

  Dữ liệu đầu vào dạng sequence, đầu ra dạng cố định. Bài toán phân loại hành động là một trường hợp cụ thể.

  Giả sử, xét đoạn video một người đang đánh golf, input sẽ là sequence nhiều ảnh có thứ tự là từng frame của đoạn videos đó, output kết quả hành động đã đánh golf của vận động viên.

Hình 6.5: Bài toán nhận diện hành động qua video

6.1.2 Một số vấn đề của mạng nơ-ron thông thường với dữ liệu chuỗi

Các kiến trúc neural network trước (bao gồm MLP và CNN) được xếp vào nhóm feed-forward neural networks. Chúng chỉ đưa ra kết quả dự báo đầu ra chỉ dựa trên dữ liệu đầu vào. Mô hình sẽ hoạt động tốt trong trường hợp giả thiết này là đúng, ví dụ:

- Dự báo giới tính của người (đầu ra) chỉ dựa vào ảnh gương mặt của người đó (đầu vào).
- Dự báo bệnh ung thư não (đầu ra) chỉ dựa vào ảnh chụp cắt lớp não (đầu vào).

Với dữ liệu dạng sequence, đầu ra y_i tại một vị trí bất kì i ngoài bị ảnh hưởng bởi các dữ liệu đầu vào x_i, còn có thể bị ảnh hưởng bởi thứ tự xuất hiện của chúng trong sequence. Một số vị dụ cho trường hợp này:

- Dự đoán hôm nay trời có mưa hay không? Ngoài việc dựa vào nhiệt độ, độ ẩm, trời nắng/râm, ta cũng có thể dựa vào dữ liệu từ các ngày hôm trước có mưa hay không (có thể xảy ra trường hợp mưa liên tục kéo dài).

- Dự đoán giá cổ phiếu cuối phiên giao dịch hôm nay? Giá có thể dựa vào tình trạng giao dịch, tình hình biến động của thị trường, và lẽ dĩ nhiên cũng dựa vào giá của những ngày hôm trước (giá hôm trước cao kéo theo giá hôm nay cũng cao).

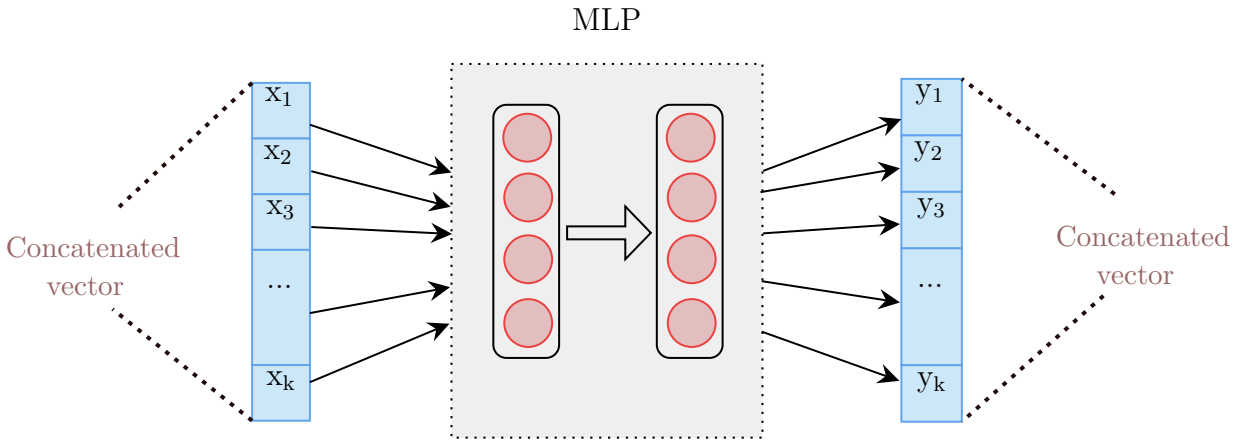
Dữ liệu dạng sequence đôi khi còn nằm ở dạng chuỗi dữ liệu thuần túy không dựa trên quan sát:

- Dự đoán và đề xuất từ tiếp theo trong chuỗi từ đang nhập từ bàn phím (phổ biến trong bàn phím điện thoại) chỉ dựa vào chuỗi từ đã nhập trước đó.
- Dự đoán kết quả sổ xố hôm nay dựa vào kết quả của những ngày hôm trước.

Các mạng feed-forward với dạng dữ liệu đầu vào cố định không thể giải quyết bài toán với dữ liệu sequence động (độ dài tùy ý). Tuy nhiên, dữ liệu sequence có thể được biến đổi (tách nhỏ) để sử dụng trên các kiến trúc chỉ dựa vào dữ liệu đầu vào bằng cách lấy một lượng cố định k mẫu các dữ liệu liên tiếp trước đó làm đầu vào. Các model tiếp cận theo hướng này được xếp vào nhóm "auto regressive models". Về tổng quát, một auto regressive model bậc k (k^{th}-order) tổng quát có thể được hiểu là một ánh xạ:

$$\hat{y}_i = f(x_i, y_{i-1}, y_{i-2}, ..., y_{i-k}) \tag{6.1}$$

với x_i là quan sát hiện có, y_i là giá trị sequence ở thời điểm i, $\hat{y}_i$ là giá trị dự đoán tại thời điểm i, k là bậc của model.



Hình 6.6: Áp dụng MLP cho bài toán dữ liệu sequence theo dạng regressive model

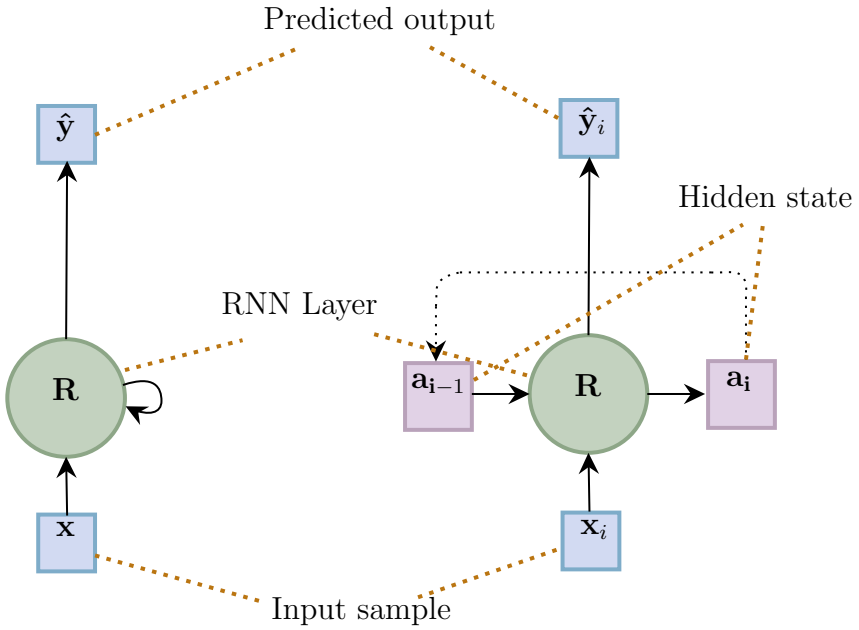
Với hướng tiếp cận như trên, ta tạm thời giải quyết được vấn đề về độ dài dữ liệu bất định. Tuy nhiên, vẫn còn rất nhiều vấn đề khiến MLP không phù hợp cho dữ liệu dạng sequence

- Mô hình chưa có khả năng ghi nhớ trạng thái từ các dữ liệu trước đó để hỗ trợ ra quyết định, vẫn phụ thuộc hoàn toàn vào đầu vào. Trong trường hợp cần ghi nhớ dữ liệu ở mức lớn hơn bậc k của mô hình, feed-forward network sẽ không thể đưa ra dự đoán chính xác. Thật vậy, cách mà mô hình regressive model nêu trên lấy dữ liệu từ các mẫu trước đó trong sequence hoàn toàn là do giả định của người thiết kế mạng.
- Số lượng tham số là rất lớn: khi sử dụng fully-connected layer, số lượng tham số của một lớp sẽ bằng tích của chiều đầu vào lẫn chiều đầu ra. Lượng tham số lớn sẽ làm ảnh hưởng tới tính chính quy hóa (regularization), khiến mô hình không học hiệu quả mà chỉ overfit trên tập training. Ngoài ra, số lượng tham số quá lớn còn khiến mô hình chiếm rất nhiều tài nguyên cả về bộ nhớ lẫn thời gian huấn luyện.
- Mô hình không có khả năng chia sẻ trọng số giữa các bước, ngoài việc sử dụng một lượng lớn tham số dùng riêng cho từng đầu vào. Vì mỗi đầu vào $\mathbf{x}_i$ đều mang tính chất là một điểm dữ liệu trong sequence, ta cần một cách chung tổng quát để xử lý chúng thay vì xử lý mỗi điểm dữ liệu theo một cách riêng (tức phải qua cùng một cách tính toán).

Nhắc lại về CNN cho bài toán xử lý ảnh, vấn đề của MLP khi giải quyết bài toán liên quan đến ảnh cũng tương tự như với dữ liệu dạng sequence. Bằng cách thêm giả định (hypothesis) các điểm dữ liệu lân cận sẽ có mối tương quan với nhau nhiều hơn các điểm ở xa (kiến thức thực tiễn từ con người), CNN đã tách việc tính toán trên toàn bộ ảnh thành việc tính toán trên từng vùng cửa sổ nhỏ hơn với cùng một bộ tham số. Tương tự, ở dữ liệu dạng sequence, với giả định thông tin về thứ tự của dữ liệu cũng như cách tính toán trên từng mẫu dữ liệu phải tương đương, người ta đã đưa ra kiến trúc về *mạng nơ-ron truy hồi* (recurrent neural network - RNN).

6.2 Giới thiệu về RNN

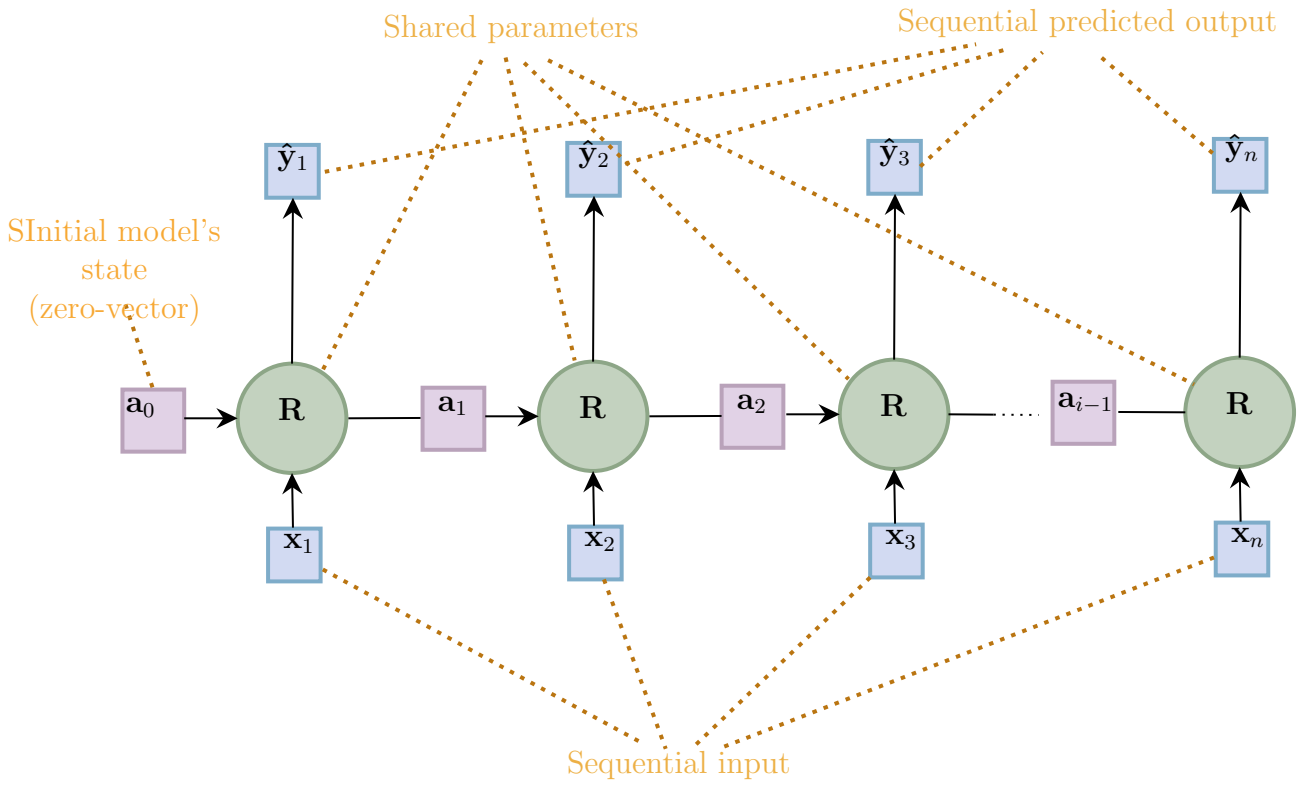
Năm 1986, Rumelhart và các cộng sự của mình đã công bố một bài nghiên cứu mang tên *"Học biểu diễn nội bộ bằng lan truyền lỗi"* (Learning internal representations by error propagation) (Rumelhart et al., 1986) . Trong bài nghiên cứu có trình bày một kiến trúc mạng kinh điển để xử lý dữ liệu dạng chuỗi, có tên là Recurrent Neural Network (RNN). Recurrent Neural Networks (RNNs), hay mạng neural hồi quy, là một kiến trúc neural network mà ở đó, trạng thái của mô hình tại bước liền trước sẽ được dùng làm đầu vào của bước hiện tại. Nói cách khác, một mô hình RNN sẽ hoạt động theo dạng ánh xạ dữ liệu đầu vào và trạng thái trước đó của mô hình thành dữ liệu đầu ra.



Hình 6.7: Kiến trúc của một lớp RNN. *Bên trái:* biểu diễn với đường nối vòng. *Bên phải:* Biểu diễn với hidden state

Với mạng neural truyền thống dạng feed-forward, dữ liệu chỉ được truyền theo một chiều, tính toán và đưa ra kết quả gói gọn trong một pha dựa hoàn toàn vào dữ liệu input (không tồn tại đường nối vòng). Ta cần làm rõ rằng đường nối vòng ở đây không tạo ra một chu kì tính toán lặp lại vô hạn, mà chỉ dùng dữ liệu từ pha tính toán trước làm đầu vào cho pha hiện tại. Cụ thể hơn, ta "gỡ" đường nối vòng

(unfold) để đưa ra kiến trúc tính toán cụ thể hơn khi truyền dữ liệu sequence vào mạng:

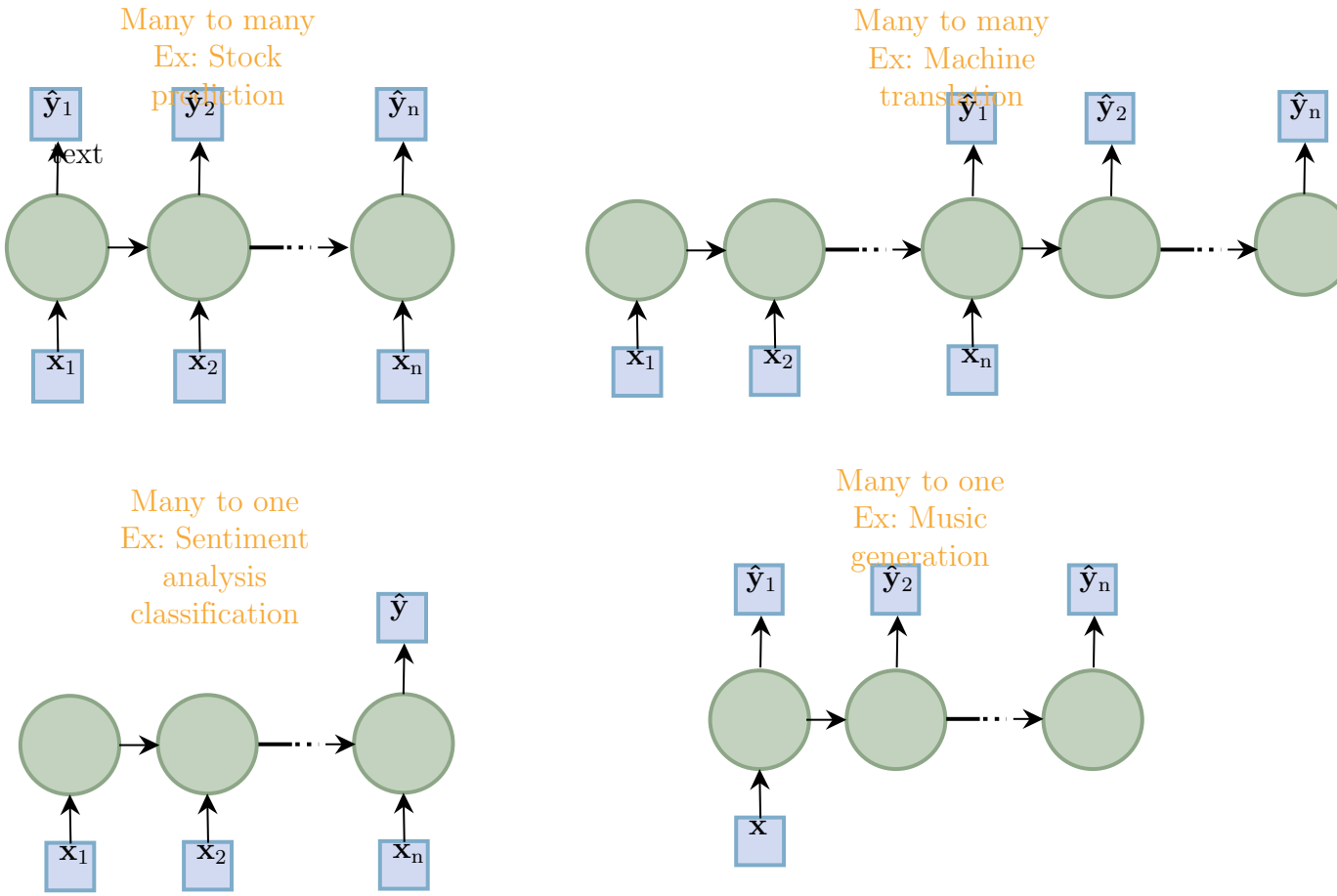


Hình 6.8: Luồng tính toán cụ thể trên sequence của RNN

Với RNN, việc thông tin từ lần truyền trước được lưu lại (tại hidden state) cho lần tính toán hiện tại, tạo ra một đường nối vòng trên đồ thị tính toán. Nói cách khác, hidden state đóng vai trò như một bộ nhớ của lớp RNN. Điều này đặc biệt hữu dụng khi xử lý dữ liệu dạng sequence, khi ta có thể áp dụng cùng một logic tính toán (cùng cách tính và bộ trọng số) cho từng điểm dữ liệu trong chuỗi, giữ được thông tin từ các điểm trước đó, đồng thời vẫn giữ được thông tin về thứ tự của sequence.

Năm 1982, Hopfield đã đưa ra kiến trúc Hopfield network là một dạng mạng neural với khả năng lưu giữ lại trạng thái cũ trước đó để hỗ trợ cho lần tính toán hiện tại. Hopfield network là một dạng đặc biệt của RNN. Trọng số trên mạng Hopfield được cập nhật bằng phương pháp tối ưu lồi (khác với gradient descent ở RNN hiện đại). RNN chính thức được đưa ra dựa trên công trình của David Rumelhart năm 1986.

Hiển nhiên ta không thể áp dụng cùng một kiến trúc RNN nêu trên cho mọi bài toán với dữ liệu sequence. Một số mẫu biến thể thiết kế (design patterns) từ kiến trúc trên để áp dụng cho từng loại bài toán:



Hình 6.9: Các mẫu thiết kế RNN cho từng loại bài toán

6.3 Cơ chế lan truyền xuôi của RNN

Để tìm hiểu rõ hơn cách hoạt động cụ thể của RNN, mục này sẽ trình bày cơ chế để lan truyền một sequence dữ liệu qua mạng. Ta xét kiến trúc mạng cụ thể cơ bản của RNN, được biểu diễn dưới dạng công trức truy hồi như sau (lưu ý vector được

biểu diễn dưới dạng cột):

$$\boldsymbol{a}_0 = 0 \tag{6.2}$$

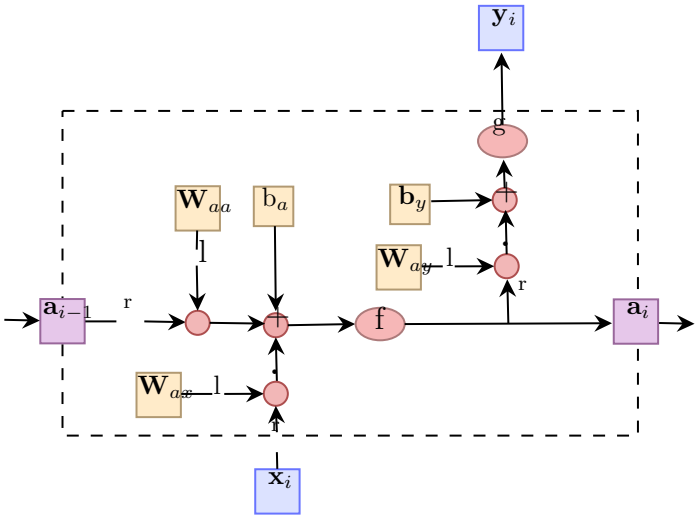
$$\boldsymbol{a}_i = f(\mathbf{W}_{ax} \cdot \mathbf{x}_i + \mathbf{W}_{aa} \cdot \mathbf{a}_{i-1} + \mathbf{b}_a) \tag{6.3}$$

$$\boldsymbol{\hat{y}_i} = g(\mathbf{W}_{ya} \cdot \mathbf{a}_i + \mathbf{b}_y) \tag{6.4}$$

Các biến số và tham số trong công thức truy hồi trên được định nghĩa là:

- $\boldsymbol{a}_i$ là hidden state - trạng thái ẩn của mô hình. $\boldsymbol{a}_0 = 0$ là quy ước cho trạng thái bắt đầu.
- $\boldsymbol{\hat{y}_i}$ là giá trị dự đoán đầu ra ứng với bước i.
- $\boldsymbol{x}_i$ là giá trị đầu vào tại bước i.
- f và g là các hàm activation phù hợp. Thông thường, ta chọn $f(x) = \tanh(x)$ hoặc $f(x) = \text{ReLU}(x)$ và $g(x) = \sigma(x)$ hoặc $g(x) = \text{Softmax(x)}$, tùy vào cách thiết kế và yêu cầu bài toán.
- $\mathbf{W}$ và $\mathbf{b}$ là các trọng số của mô hình, cụ thể:
  - $\mathbf{W}_{ax}$ là ma trận của phép ánh xạ tuyến tính biến đổi $\mathbf{x}$ thành một thành phần trong $\mathbf{a}$
  - $\mathbf{W}_{aa}$ là ma trận của phép ánh xạ tuyến tính biến đổi trạng thái $\mathbf{a}$ cũ thành một thành phần trong trạng thái $\mathbf{a}$ mới.
  - $\mathbf{b}_a$ là bias của phép biến đổi tìm $\mathbf{a}$
  - $\mathbf{W}_{ya}$ là ma trận của phép ánh xạ tuyến tính biến đổi trạng thái hiện tại $\mathbf{a}$ thành kết quả đầu ra $\hat{\mathbf{y}}$
  - $\mathbf{b}_a$ là bias của phép biến đổi tìm $\hat{\mathbf{y}}$

Ở dạng đồ thị tính toán, một nút RNN cơ bản có thể được biểu diễn bằng:



Hình 6.10: Đồ thị tính toán cụ thể của một nút trong RNN

Về lý thuyết, với mỗi nút đầu vào của sequence, RNN sẽ đều cho ra một kết quả đầu ra. Tuy nhiên, chọn nút đầu ra nào làm kết quả và tối ưu như thế nào là tùy vào mục tiêu của bài toán. Lấy ví dụ trong các trường hợp sau:

- Với bài toán dự đoán giá cổ phiếu (dạng many-to-many đối xứng): Giả sử chuỗi sequence là thông tin input của mạng theo từng ngày và đầu ra của mạng là giá cổ phiếu cho ngày hôm đó. Với cách thiết kế many-to-many, mỗi $\hat{\mathbf{y}}$ đều tương ứng với một ngày nào đó trong sequence. Vì vậy, mọi đầu ra tại từng bước đều mang ý nghĩa. Ta cần tính toán hàm loss dựa trên tất cả giá trị đầu ra của mạng.
- Với bài toán sentiment classification (dạng many-to-one): xét trường hợp đơn giản nhất là gắn một nhãn tốt/xấu cho một câu, với dữ liệu đầu vào là từng từ trong câu đó, đầu ra là kết quả dữ đoán câu đó tốt hay xấu. Vì mỗi câu chỉ mang duy nhất một nhãn, ta nhận thấy rằng, chỉ có kết quả dự đoán của mô hình khi toàn bộ câu đã được truyền vào mới mang ý nghĩa.
  - Câu "This movie is not bad" khi được truyền toàn bộ vào model, model có thể gắn nhãn "tốt" cho câu (phim không tệ là phim hay)
  - Tuy nhiên, cùng với câu trên, khi chỉ lấy đoạn "This movie is not", model lại có thể gắn nhãn "xấu" cho câu (từ "not" mang nghĩa phủ định, có thể

là tiêu cực)

Lẽ dĩ nhiên, qua mỗi bước truyền, mạng đều cho ra một kết quả đánh giá mức tốt/xấu của câu đến điểm đó. Tuy nhiên, xét riêng trong ngữ cảnh bài toán này, ta quan tâm tới nhãn của toàn câu chứ không quan tâm nhãn của từng đoạn ngắn trong câu. Vì vậy, chỉ có output ở nút cuối cùng là có ý nghĩa.

Việc xác định cụ thể ta cần chọn đầu ra nào là rất quan trọng, vì mô hình sẽ dựa vào đúng đầu ra có ý nghĩa đó để cập nhật trọng số. Việc lan truyền của RNN có thể được hiểu thông qua hiện thực cụ thể ở đoạn mã sau bằng ngôn ngữ Python:

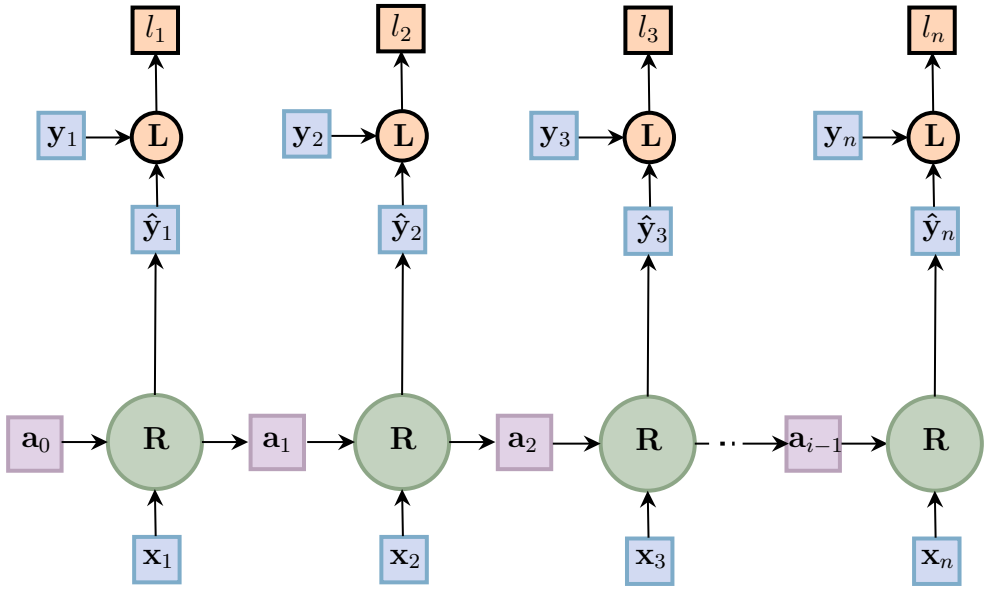
```
import numpy as np

dim_x = 5  # Input dimension
dim_y = 5  # Output dimension
dim_a = 3  # Hidden state dimension

n = 10  # Input sequence length

X = np.random.randn(n, dim_x)

W_ax = np.random.rand(dim_a, dim_x)  # Shape(W_ax) = (3, 5)
W_aa = np.random.rand(dim_a, dim_a)  # Shape(W_aa) = (3, 3)
b_a = np.random.rand(dim_a)          # Shape(b_a)  = (3,)
W_ya = np.random.rand(dim_y, dim_a)  # Shape(W_ya) = (5, 3)
b_y = np.random.rand(dim_y)          # Shape(b_y)  = (5,)

a = np.zeros(dim_a)  # Initial state = zeros
y = []

# Forwarding process
for i in range(n):
    xi = X[i].T  # Transform into column-vector
    # Follow equation 6.3 and 6.4 with f is tanh and g is softmax
    a = np.tanh(np.dot(W_ax, xi) + np.dot(W_aa, a) + b_a)
    yi = np.dot(W_ya, a) + b_y
    yi = np.exp(yi) / np.sum(np.exp(yi))
    y.append(yi)
```

6.4 Cơ chế lan truyền ngược của RNN

Nhắc lại về các bước huấn luyện một mạng feed-forward với gradient descent:

1. Bước forward: Đưa dữ liệu đầu vào $\mathbf{x}$ qua mạng, thu được kết quả đầu ra $\hat{\mathbf{y}}$
2. Tính toán hàm mất mát $\mathcal{L}(\hat{\mathbf{y}})$ cho bài toán học không giám sát hoặc $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$ cho bài toán học có giám sát
3. Bước backward: Tính đạo hàm của hàm mất mát $\Delta w = \delta\mathcal{L}/\delta w$ cho từng tham số w trong mạng
4. Bước cập nhật: Cập nhật các tham số theo tốc độ học α hiện tại: $w \leftarrow w - \alpha\Delta w$

Quá trình này cũng tương tự với RNN. Tuy nhiên, vì tính đặc thù với đường nối vòng, một số vấn đề sẽ nảy sinh khiến ta cách tối ưu thông thường gặp phải nhiều vấn đề. Ta xét một mạng RNN với hàm mất mát dạng cũ và bài toán học có giám sát như Hình 6.11:



Hình 6.11: RNN với hàm mất mát

Với kiến trúc truyền tuần tự từng mẫu, tại mỗi thời điểm, chúng ta đều tính toán được một giá trị đầu ra. Hàm mất mát truyền thống được tính dựa trên một giá trị đầu ra dự báo $\hat{\mathbf{y}}$ và một giá trị nhãn thực $\mathbf{y}$, nên với hướng tiếp cận này, chúng ta

có nhiều hơn một giá trị hàm mất mát tại mỗi bước. Tuy nhiên, để thực hiện tính toán đạo hàm truyền ngược, ta cần một giá trị mất mát duy nhất. Vì vậy, ta cần một hàm mất mát mới, nhận giá trị đầu vào là tất cả các đầu ra của mạng $\hat{\mathbf{y}}_i$ và tất cả các nhãn ở từng bước $\mathbf{y}$, và cho ra một con số duy nhất là hàm mất mát của toàn bộ quá trình. Một số hướng tiếp cận để giải quyết vấn đề này như:

- Với bài toán dạng many-to-one: Ta chỉ quan tâm đến một giá trị đầu ra cuối cùng của mạng. Vì vậy, ta chỉ cần tối ưu nhãn cuối cùng của một pha: $\mathcal{L} = \mathcal{L}(\hat{\mathbf{y}}_n, \mathbf{y}_n)$
- Với các bài toán còn lại: khi có nhiều hơn một đầu ra, ta có thể chọn một hàm để tích hợp các giá trị l_i lại thành một giá trị duy nhất (tổng, trung bình, cực đại, etc) tùy vào trường hợp bài toán cụ thể.

Để nhìn nhận quá trình một cách rõ ràng hơn, vì tất cả các nút RNN đều chia sẻ trọng số trong quá trình đưa từng mẫu dữ liệu qua mạng, ta tách trọng số của mô hình ra thành các khối dữ liệu. Khi đó, mô hình bao gồm cả hàm mất mát sẽ có dạng như Hình 6.12:

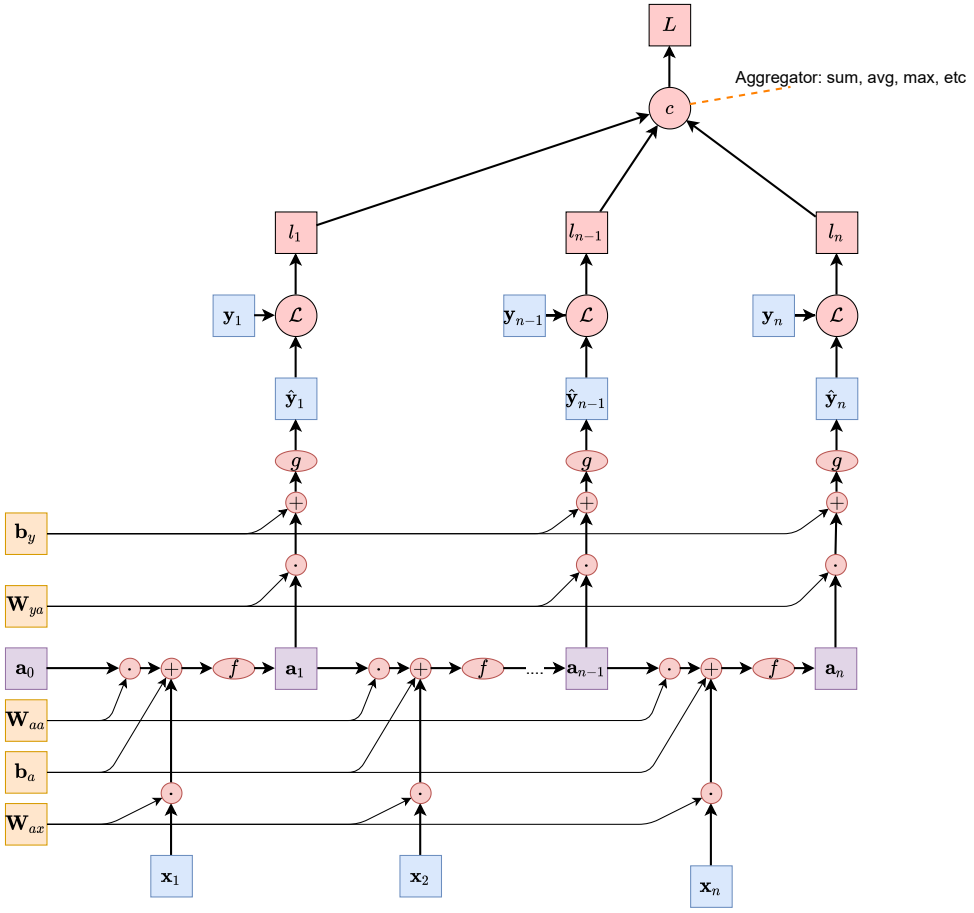
với các thành phần mới:

- $l_i = \mathcal{L}(\hat{\boldsymbol{y}}_i, \boldsymbol{y}_i)$ là giá trị hàm loss cho đầu ra thứ i
- $L = c(l_1, l_2, ..., l_n)$ là giá trị hàm mất mát tổng; c là một hàm tích hợp nhiều giá trị mất mát (tổng, cực đại,...)

Với sự hỗ trợ của các framework tính toán hiện đại, việc tính toán đạo hàm hầu hết đã được tự động hóa. Tuy nhiên, để nắm rõ quy trình hoạt động của RNN cũng như đánh giá khả năng hoạt động của kiến trúc, phần này sẽ trình bày cụ thể cách tính đạo hàm trong RNN.

Ta tính lần lượt các giá trị $\Delta w = \frac{\partial L}{\partial w}$ cho từng tham số w trong mạng, cụ thể là từng bộ tham số $\boldsymbol{W}_{ya}, \boldsymbol{b}_y, \boldsymbol{W}_{aa}, \boldsymbol{W}_{ax}, \boldsymbol{b}_a$:

$$\Delta \boldsymbol{W}_{ya} = \frac{\partial L}{\partial \boldsymbol{W}_{ya}} = \sum_{i=1}^{n} \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\boldsymbol{y}}_i} \cdot \frac{\partial \hat{\boldsymbol{y}}_i}{\partial \boldsymbol{W}_{ya}} \right) = \sum_{i=1}^{n} \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\boldsymbol{y}}_i} \cdot g'(\boldsymbol{W}_{ya} \cdot \boldsymbol{a}_i + \boldsymbol{b}_y) \cdot \frac{\partial (\boldsymbol{W}_{ya} \cdot \boldsymbol{a}_i)}{\partial \boldsymbol{W}_{ya}} \right) \tag{6.5}$$



Hình 6.12: RNN với hàm mất mát và trọng số cụ thể

$$\Delta \boldsymbol{b}_y = \frac{\partial L}{\partial \boldsymbol{b}_y} = \sum_{i=1}^{n} \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\boldsymbol{y}}_i} \cdot \frac{\partial \hat{\boldsymbol{y}}_i}{\partial \boldsymbol{b}_y} \right) = \sum_{i=1}^{n} \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\boldsymbol{y}}_i} \cdot g'(\boldsymbol{W}_{ya} \cdot \boldsymbol{a}_i + \boldsymbol{b}_y) \right) \tag{6.6}$$

$$\Delta \boldsymbol{W}_{aa} = \frac{\partial L}{\partial \boldsymbol{W}_{aa}} = \sum_{i=1}^{n} \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\boldsymbol{y}}_i} \cdot \frac{\partial \hat{\boldsymbol{y}}_i}{\partial \boldsymbol{a}_i} \cdot \frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{W}_{aa}} \right) = \sum_{i=1}^{n} \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\boldsymbol{y}}_i} \cdot \boldsymbol{W}_{ya} \cdot \mathrm{P}_i \right) \tag{6.7}$$

với

$$\mathrm{P}_i = \frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{W}_{aa}} = f_i' \cdot (\boldsymbol{W}_{aa} + \boldsymbol{I}) \cdot \frac{\partial \boldsymbol{a}_{i-1}}{\partial \boldsymbol{b}_a} = f_i' \cdot (\boldsymbol{W}_{aa} + \boldsymbol{I}) \cdot \mathrm{P}_{i-1} \quad ; \quad \mathrm{P}_0 = 0 \tag{6.8}$$

$$\Delta \boldsymbol{b}_a = \frac{\partial L}{\partial \boldsymbol{W}_{ya}} = \sum_{i=1}^{n} \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\boldsymbol{y}}_i} \cdot \frac{\partial \hat{\boldsymbol{y}}_i}{\partial \boldsymbol{a}_i} \cdot \frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{b}_a} \right) = \sum_{i=1}^{n} \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\boldsymbol{y}}_i} \cdot \boldsymbol{W}_{ya} \cdot \mathrm{Q}_i \right) \tag{6.9}$$

với

$$\mathrm{Q}_i = \frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{b}_a} = f_i' \cdot \left(\boldsymbol{W}_{aa} \cdot \frac{\partial \boldsymbol{a}_{i-1}}{\partial \boldsymbol{b}_a} + \boldsymbol{I} \right) = f_i' \cdot \left(\boldsymbol{W}_{aa} \cdot \mathrm{Q}_{i-1} + \boldsymbol{I} \right) \quad ; \quad \mathrm{Q}_0 = 0 \tag{6.10}$$

$$\Delta \boldsymbol{W}_{ax} = \frac{\partial L}{\partial \boldsymbol{W}_{ax}} = \sum_{i=1}^{n} \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\boldsymbol{y}}_i} \cdot \frac{\partial \hat{\boldsymbol{y}}_i}{\partial \boldsymbol{a}_i} \cdot \frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{W}_{ax}} \right) = \sum_{i=1}^{n} \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\boldsymbol{y}}_i} \cdot \boldsymbol{W}_{ya} \cdot \mathrm{R}_i \right) \tag{6.11}$$

với

$$\mathrm{R}_i = \frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{W}_{ax}} = f_i' \cdot \left(\frac{\partial (\boldsymbol{W}_{ax} \cdot \boldsymbol{x}_i)}{\partial \boldsymbol{W}_{ax}} + \boldsymbol{W}_{aa} \cdot \frac{\partial \boldsymbol{a}_{i-1}}{\partial \boldsymbol{W}_{ax}} \right) = f_i' \cdot \left(\frac{\partial (\boldsymbol{W}_{ax} \cdot \boldsymbol{x}_i)}{\partial \boldsymbol{W}_{ax}} + \boldsymbol{W}_{aa} \cdot R_{i-1} \right) \quad ; \quad \mathrm{R}_0 = 0 \tag{6.12}$$

Bạn đọc có thể tự chứng minh các biểu thức đạo hàm ở trên (Dựa vào đồ thị tính toán, đạo hàm của một nút dữ liệu bất kì b theo một nút a bằng tổng đạo hàm theo chain rule trên tất cả các đường đi từ a đến b).
Với RNN, trên đồ thị tính toán, có nhiều hơn một đường đi từ một nút tham số bất kì tới nút hàm mục tiêu $\mathcal{L}$. Có thể nhận xét rằng đường nối vòng trong RNN làm việc tính toán đạo hàm thủ công trở nên phức tạp hơn so với trên mạng feed-forward truyền thống.

Nạng RNN với quá trình lan truyền ngược như vậy sẽ xảy ra hiện tượng đạo hàm suy biến. Phụ lục C sẽ trình bày rõ việc này.

6.5 Ưu và nhược điểm của RNN

RNN được thiết kế đặc thù để giải quyết dữ liệu dạng sequence. Một số ưu điểm của RNN so với mạng feed-forward kiểu truyền thống có thể kể đến như:

- Khả năng xử lý dữ liệu với độ dài bất định
- Kích thước mô hình là cố định và độc lập với độ dài đầu vào
- Việc tính toán có tận dụng được thông tin từ quá khứ
- Sử dụng chung một bộ trong số cho toàn bộ sequence, tức áp dụng cùng một logic tính toán trên từng điểm dữ liệu trong sequence.
- Chia sẻ trọng số trong quá trình tính toán còn giúp làm giảm số lượng tham số, cải thiện tính tổng quát hóa (regularization) và tránh sự quá khớp

Mặc dù vậy RNN (cơ bản) vẫn gặp phải một số vấn đề nhất định:

- Tính toán tuần tự: Kết quả của lần tính toán hiện tại dựa vào kết quả của lần tính toán trước. Vì vậy, dữ liệu từ các lần truyền không thể được thực hiện song song, dẫn đến việc không tận dụng được GPU
- Có khả năng bị "quên" dữ liệu: Dù RNN có được thiết kế để lưu trữ thông tin từ quá khứ, tuy nhiên với các thiết kế cơ bản, chúng vẫn chỉ lưu được một lượng thông tin từ một số hữu hạn các lần lặp trước đó. Lý do cho sự "quên" này là do việc nhân lặp đi lặp lại của một giá trị trong chuỗi chain-rule ở quá trình lan truyền ngược dẫn đến việc đạo hàm quá nhỏ hoặc quá lớn. Để giải quyết vấn đề này, người ta đã đưa ra kiến trúc LSTM với khả năng chọn lọc thông tin để giữ/bỏ trong quá trình lan truyền thuận, và tránh việc đạo hàm quá nhỏ hoặc quá lớn trong quá trình lan truyền ngược
- RNN cơ bản chỉ có thể nhìn vào "quá khứ", tức dự đoán hiện tại dựa trên thông tin đã có trong quá khứ. Trong thực tế, có nhiều trường hợp, mẫu dự đoán hiện tại còn cần cả thông tin về các mẫu trong tương lai. Biến thể Bi-directional RNN có thể giúp giải quyết bài toán này.

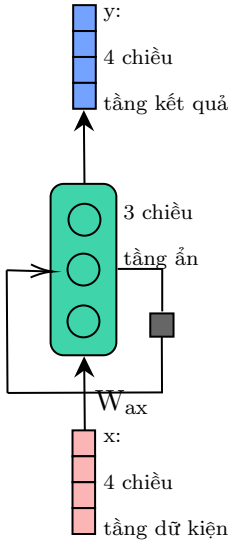
6.6 Ví dụ minh họa mạng RNN

Sau đây chúng ta sẽ làm một ví dụ minh họa cho việc sử dụng RNN trong các mô hình tự động sinh văn bản. RNN cho phép ta dự đoán xác suất xuất hiện của chữ mới dựa vào các chữ đã có liền trước đó theo cơ chế các đầu ra của cụm này sẽ là đầu vào của cụm tiếp theo cho tới khi ta được câu (hoặc từ, tùy theo mô hình cụ thể) hoàn chỉnh. Giả sử ta có 1 kho ngữ liệu gồm các chữ *("H", "E", "L", "O")* và ta muốn huấn luyện mạng RNN sao cho sau khi đọc được chuỗi "HE", máy sẽ cho dự đoán chữ tiếp theo là "L" (để có từ "HELLO").

Đầu tiên ta sẽ mã hóa các chữ cái trong kho ngữ liệu dưới dạng one-hot encoding. Vì kho ngữ liệu chỉ có 4 chữ cái nên one-hot vector có số chiều là 4:

$$H \longrightarrow \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \quad E \longrightarrow \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \quad L \longrightarrow \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \quad O \longrightarrow \begin{pmatrix}0\\0\\0\\1\end{pmatrix},$$

Giả sử mạng RNN được xây dựng có: 3 là số nút trong 1 lớp ẩn, 4 là số chiều của one-hot vector đầu vào, khi đó số chiều của các ma trận W_{ax}, W_{aa}, W_{ya} lần lượt là: (3, 4), (3, 3), (4, 3). Sơ đồ tính toán lặp được rút gọn như Hình 6.13:



Hình 6.13: Sơ đồ tính toán RNN rút gọn

Bắt đầu tính toán ta cần khởi tạo các ma trận W_{ax}, W_{aa}, W_{ya} với các giá trị ngẫu nhiên:

$$W_{ax} = \begin{pmatrix} 0.1 & 0.3 & 0.2 & 0.3 \\ 0.1 & 0.7 & 0.3 & 0.2 \\ 0.4 & 0.5 & 0.8 & 0.6 \end{pmatrix},$$

$$W_{aa} = \begin{pmatrix} 0.3 & 0.8 & 0.2 \\ 0.6 & 0.4 & 0.1 \\ 0.4 & 0.3 & 0.9 \end{pmatrix},$$

$$W_{ya} = \begin{pmatrix} 0.4 & 0.2 & 0.6 \\ 0.9 & 0.1 & 0.5 \\ 0.2 & 0.2 & 0.6 \\ 0.1 & 0.8 & 0.4 \end{pmatrix},$$

Như đã trình bày ở phần trước, tại bước thứ t đầu vào $x^{<t>}$ sẽ được kết hợp với các nút ở lớp ẩn $a^{<t-1>}$ bằng hàm g_1 (thường dùng là tanh hoặc ReLU) để tính toán ra $a^{<t>}$ và từ $a^{<t>}$ sẽ tính ra được $\widehat{y}^{<t>}$ thông qua hàm g_2 (thường là hàm softmax hoặc hàm sigmoid tùy theo số phân lớp, ở đây ta sẽ dùng softmax vì giá trị đầu ra $\widehat{y}^{<t>}$ có 4 chiều). Quá trình này lặp lại như sau:

Bước lan truyền thuận:

Giá trị $a^{<0>}$ được lấy tùy ý, ví dụ, $a^{<0>} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T$, ký tự đầu vào $x^{<1>} = H = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}^T$, ta tính được $a^{<1>}$, $\widehat{y}^{<1>}$:

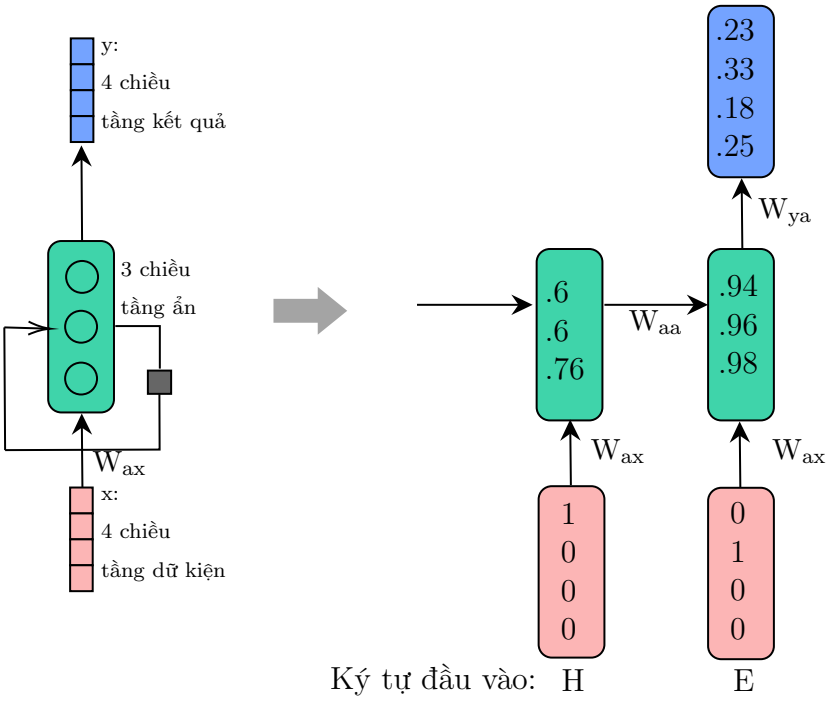
$$
\begin{aligned}
a^{<1>} &= g_1(W_{aa}a^{<0>} + W_{ax}x^{<1>} + b_a) \\
&= tanh(\begin{pmatrix} .3 & .8 & .2 \\ .6 & .4 & .1 \\ .4 & .3 & .9 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} .1 & .3 & .2 & .3 \\ .1 & .7 & .3 & .2 \\ .4 & .5 & .8 & .6 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + 0.6) \\
&= tanh(\begin{pmatrix} .7 \\ .7 \\ 1. \end{pmatrix}) \\
&= \begin{pmatrix} .6 \\ .6 \\ .76 \end{pmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\widehat{y}^{<1>} &= g_2(W_{ya}a^{<1>} + b_y) \\
&= softmax(\begin{pmatrix} .4 & .2 & .6 \\ .9 & .1 & .5 \\ .2 & .2 & .6 \\ .1 & .8 & .4 \end{pmatrix} \begin{pmatrix} .6 \\ .6 \\ .76 \end{pmatrix} + .6) \\
&= \begin{pmatrix} .24 \\ .29 \\ .21 \\ .26 \end{pmatrix}
\end{aligned}
$$

Tiếp tục sử dụng tính toán tương tự ta tính được:

$$
\begin{aligned}
a^{<2>} &= \begin{pmatrix} .94 & .96 & .98 \end{pmatrix}^T \\
\widehat{y}^{<2>} &= \begin{pmatrix} .23 & .33 & .18 & .25 \end{pmatrix}^T
\end{aligned}
$$

Tóm lại ta có sơ đồ tính toán như Hình 6.14 cho bước lan truyền thuận:



Hình 6.14: Sơ đồ tính toán của bài toán dự đoán chữ cái dạng đầy đủ

Nếu ta chuyển đổi các xác suất này để hiểu dự đoán, chúng ta sẽ thấy rằng, tại đầu vào thứ 2, $x^{<2>}$, mô hình nói rằng xác suất cao nhất sẽ xuất hiện tiếp theo là chữ "E"([0 1 0 0]) vì $\hat{y}^{<2>}$ =[.23 .33 .18 .25]. Điều đó cho thấy, với hệ thống hiện tại, máy sẽ dự đoán chữ tiếp theo sau khi nhập "HE" sẽ là chữ "E" chứ không phải chữ "L" như mong muốn.

Đó là vì ta chưa thực hiện cho máy học. Ở bước lan truyền ngược, ta cập nhật các thông số W_{ax}, W_{aa}, W_{ya}, b_a, b_y sao cho tối thiểu hàm mất mát.

Bước lan truyền ngược:

Ta viết lại các ma trận trọng số để cập nhật các bias $b_a$và $b_y$cùng lúc với W_{ax}và W_{ya}:

$$W_{ax} \longrightarrow (W_{ax} \mid b_a) \longrightarrow \begin{pmatrix} 0.1 & 0.3 & 0.2 & 0.3 & 0.6 \\ 0.1 & 0.7 & 0.3 & 0.2 & 0.6 \\ 0.4 & 0.5 & 0.8 & 0.6 & 0.6 \end{pmatrix}$$

Chọn hệ số học $\eta = 1.0$, cập nhật W_{ax} theo công thức:

$$
\begin{aligned}
W_{ya} &= W_{ya} - \eta(\widehat{y}^{<2>} - y^{<2>}) \otimes a^{<2>} \\
&= \begin{pmatrix} 0.4 & 0.2 & 0.6 & 0.6 \\ 0.9 & 0.1 & 0.5 & 0.6 \\ 0.2 & 0.2 & 0.6 & 0.6 \\ 0.1 & 0.8 & 0.4 & 0.6 \end{pmatrix} - 1.0\left(\begin{pmatrix} .23 \\ .33 \\ .18 \\ .25 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right) \otimes \begin{pmatrix} .94 \\ .96 \\ .98 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 0.4 & 0.2 & 0.6 & 0.6 \\ 0.9 & 0.1 & 0.5 & 0.6 \\ 0.2 & 0.2 & 0.6 & 0.6 \\ 0.1 & 0.8 & 0.4 & 0.6 \end{pmatrix} - \begin{pmatrix} 0.2162 & 0.2208 & 0.2254 & 0.23 \\ 0.3102 & 0.3168 & 0.3234 & 0.33 \\ -0.7708 & -0.7872 & -0.8036 & -0.82 \\ 0.235 & 0.24 & 0.245 & 0.25 \end{pmatrix} \\
&= \begin{pmatrix} 0.1838 & -0.0208 & 0.3746 & 0.37 \\ 0.5898 & -0.2168 & 0.1766 & 0.27 \\ 0.9708 & 0.9872 & 1.4036 & 1.42 \\ -0.135 & 0.56 & 0.155 & 0.35 \end{pmatrix}
\end{aligned}
$$

Tương tự ta cập nhật W_{ax} và W_{aa}. Sau một số lần lặp đủ lớn ta thu được bộ tham số (W, b) có thể sử dụng làm mô hình dự đoán. Đầu ra $\widehat{y}^{<t>}$ chính là giá trị dự đoán sau khi có t kí tự được nhập.

6.7 Bài tập

Bài tập 6.7.1. Nhận diện dữ liệu dạng chuỗi:

Trong các ngữ cảnh sau đây, dữ liệu nào thuộc về dạng chuỗi? Tại sao?

a) Dữ liệu chứng khoán: dữ liệu về giá các mã cổ phiếu bao gồm loại mã, giá mở cửa, giá đóng cửa của một sàn chứng khoán được thu thập liên tục theo thời gian.

b) Dữ liệu về thời tiết: thông tin khí tượng thu thập được theo vùng miền về nhiệt độ, độ ẩm, lượng mưa, ...

c) Dữ liệu về chuỗi văn bản: một lượng lớn văn bản bao gồm nhiều câu ở một ngôn ngữ nào đó

Bài tập 6.7.2. Nhận dạng và phân tích dạng bài toán:

Với các bài toán cụ thể sau đây, có thể mô hình hóa chúng để áp dụng RNN được không? Nếu có, hãy xếp dạng bài toán vào một dạng của RNN (one-to-many, many-to-one, many-to-many, sequence-to-sequence) và giải thích.

a) Bài toán về mô hình ngôn ngữ: Dự đoán kí tự hoặc từ tiếp theo xuất hiện trong một chuỗi văn bản đang nhập.

b) Bài toán dự báo thời tiết: Dự đoán thời tiết cho chiều hôm nay dựa vào các thông số khí tượng hiện có và thông tin thời tiết từ các ngày trước đó.

c) Dự đoán doanh số bán hàng: dựa vào thông tin doanh số của thời gian hiện tại, liền trước, cùng kì tháng trước, cùng kì quý trước và cùng kì năm trước, hãy dự báo doanh số bán hàng cho từng ngày trong một tháng tới.

d) Bài toán dịch máy: Từ một câu văn bản ở tiếng Việt, hãy dịch câu đó sang tiếng Anh. Dạng bài toán có giữ nguyên không nếu như hướng tiếp cận là dịch theo cặp từ (word-by-word)?

Bài tập 6.7.3. Lan truyền xuôi trong RNN:

a) Trong đồ thị tính toán, ý nghĩa của các nút dữ liệu $\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{y}$ là gì? Giải thích ý nghĩa của các ma trận trọng số?

b) Tại sao trên đồ thị tính toán có nhiều nút R, nhưng chỉ có một bộ trọng số? Việc dùng chung một bộ trọng số cho các nút có tác dụng gì?

Bài tập 6.7.4. Lan truyền ngược trong RNN:

a) Trong công thức tính đạo hàm, hãy viết dạng (shape) của từng thành phần (số chiều của vector, ma trận, tensor, ...). Các phép nhân trong chain-rule là phép nhân dạng gì? Phép nhân đó với tensor nhiều hơn hai chiều được thực hiện như thế nào?

b) Dựa vào đồ thị tính toán và công thức truy hồi, chứng minh lại công thức tính đạo hàm cho từng bộ tham số trong mạng.

c) Giải thích hiện tượng vanishing/exploding gradient, tại sao mạng RNN cơ bản không hiệu quả trong việc ghi nhớ chuỗi dữ liệu dài? Vanishing/exploding gradient sẽ xảy ra với bộ trọng số cụ thể nào trong mạng?

Bài tập 6.7.5.

a) So sánh RNN và mạng feed-forward truyền thống (MLP là một ví dụ).

b) Có thể dùng một mạng MLP để biểu diễn chính xác một mạng RNN cơ bản hay không? Nếu có, tại sao không nên dùng?

Bài tập 6.7.6. Có phải RNN luôn tốt hơn mạng MLP không?

Bài tập 6.7.7. Cho trước một kho ngữ liệu là "Goodbye", với mục tiêu huấn luyện là hệ thống sẽ đoán chữ "d" sau khi người dùng nhập chuỗi "Goo".

Hãy:

a) Tính toán giá trị $\widehat{y}$ và so sánh với kết quả mong muốn khi áp dụng lan truyền xuôi trong RNN. Giải thích kết quả đạt được.

b) Sử dụng kết quả tính toán được ở câu trên để cập nhật trọng số cho các ma trận trong mạng,biết rằng hệ thống có hệ số học là 0.1 và hàm mất mát là Cross Entropy.

6.8 Phụ lục

Dưới đây là một số thuật ngữ có đề cập trong chương 6:

Bảng 6.1: Bảng các thuật ngữ

Thuật ngữ	Ý nghĩa
Sequence data	dữ liệu dạng chuỗi
regularization	chính quy hóa
overfitting	sự quá khớp
recurrent neural network	mạng nơ-ron truy hồi
hidden state	trạng thái ẩn
design pattern	thiết kế mẫu
unsupervised learning	học không giám sát
supervised learning	học có giám sát
learning rate	hệ số học
vanishing gradient	đạo hàm suy biến
corpus	tập ngữ liệu
computational graph	đồ thị tính toán

Chương 7

Mô hình ngôn ngữ

7.1 Mô hình hoá ngôn ngữ

7.1.1 Giới thiệu

Từ những buổi bình minh của thời đại máy móc và tự động hoá thì ước mơ về một cỗ máy có trí tuệ và khả năng hoàn thành được những công việc phức tạp và thay thế con người luôn hiện hữu. Và điều đó càng được chứng minh rõ hơn thông qua công trình của cha đẻ của ngành Khoa học Máy tính Alan Turing với Turing Test. Thông qua paper *"Computing Machinery and Intelligence"* (1950). Khi còn ở đại học Manchester, ông đã đề xuất một cách kiểm tra để xem máy tính có khả năng hành động như con người (*acting humanly*) hay không. Cách kiểm chứng đó cũng cho chúng ta thấy khả năng hiểu và xử lý ngôn ngữ là một trong những nhánh, lĩnh vực cực kỳ quan trọng để đánh giá về trí thông minh (*Intelligence*).

Bên cạnh đó, những vấn đề được đề xuất nghiên cứu của Trí tuệ Nhân tạo sau hội nghị Dartmouth (1956) bao gồm: khả năng *lập luận* (reasoning), *biểu diễn tri thức* (knowledge representation), *hoạch địn*h (planning), *khả năng học* (learning), *xử lý ngôn ngữ tự nhiên* (natural language processing), *khả năng nhận thức* (perception) và *khả năng di chuyển và điều khiển các vật thể* (ability to move and manipulate objects). Điều đó cho chúng ta thấy thách thức để "dạy" máy tính "học" cách giao tiếp bằng ngôn ngữ tự nhiên như con người là vô cùng lớn và được các nhà khoa

học chỉ ra rất lâu và cũng như là một rào cản quan trọng để xây dựng một *Trí tuệ nhân tạo tổng quát* (Artificial General Intelligence (AGI)).

Những cách xử lý ngôn ngữ trước đây thường dựa vào các công trình của Chomsky và văn phạm của ông. Việc mô hình hoá ngôn ngữ là một công việc đầy thách thức và phức tạp vì đối với các *ngôn ngữ hình thức* (formal language) thì việc mô hình hoá là điều có thể làm được. Tuy nhiên, đối với ngôn ngữ tự nhiên, thì có rất nhiều quy tắc (*rules*), một số lượng từ rất lớn, thậm chí có nhiều cách dùng từ và biểu diễn dễ gây mơ hồ, khó hiểu mà thậm chí đối với người dùng như ngôn ngữ mẹ đẻ vẫn chưa thành tạo (Ví dụ: teencode, từ địa phương, từ đồng âm, đồng nghĩa, chơi chữ...).

Mô hình ngôn ngữ (*Language Model*): nói đơn giản là việc gán xác xuất (*probability*) cho một chuỗi các từ hay câu.

Ví dụ: xác suất xuất hiện của câu *chúng ta không thuộc về nhau.*

Bên cạnh việc gán một xác suất cho mỗi chuỗi từ, các mô hình ngôn ngữ cũng chỉ định một xác suất cho khả năng (*likelihood*) một từ nhất định (hoặc một chuỗi các từ) theo một chuỗi từ biết trước.

Ví dụ: xác suất thấy từ *nhau* sau khi thấy chuỗi *chúng ta không thuộc về.*

Biểu diễn dưới góc nhìn toán học, việc gán xác suất cho chuỗi từ có thể dùng chain-rule như sau:

$$\begin{aligned} P(w_1, w_2, ..., w_n) &= P(w_1)P(w_2|w_1)P(w_3|w1, w_2)P(w_4|w_{1:3})...P(w_n|w_{1:n-1}) \\ &= \prod_{i=1}^{n} P(w_i|w_{1:i-1}). \end{aligned} \tag{7.1}$$

Ví dụ: P("chúng ta không thuộc về nhau")
=P(chúng)×P(ta|chúng)×P(không|chúng ta)×P(thuộc|chúng ta không)
×P(về|chúng ta không thuộc)×P(nhau|chúng ta không thuộc về)

Công thức 7.1 có thể biến đổi thành:

$$P(w_i|w_1, w_2, ..., w_{i-1}) = \frac{P(w_1, w_2, ..., w_{i-1}, w_i)}{P(w_1, w_2, ..., w_{i-1})}. \\ \propto P(w_1, w_2, ..., w_{i-1}, w_i). \tag{7.2}$$

Từ (7.1), (7.2) ta thấy rằng việc gán xác suất cho một chuỗi từ có thể xem là tương đương với gán xác suất cho khả năng (*likelihood*)) một từ nhất định (hoặc một chuỗi các từ) theo một chuỗi biết trước.

Ví dụ: ta cần dự đoán sự xuất hiện của từ tiếp theo của câu:

P("chúng ta không thuộc về ___").

Giả sử trong hai từ có khả năng nhất là {"đâu","nhau"}. Khi đó, so sánh giữa:

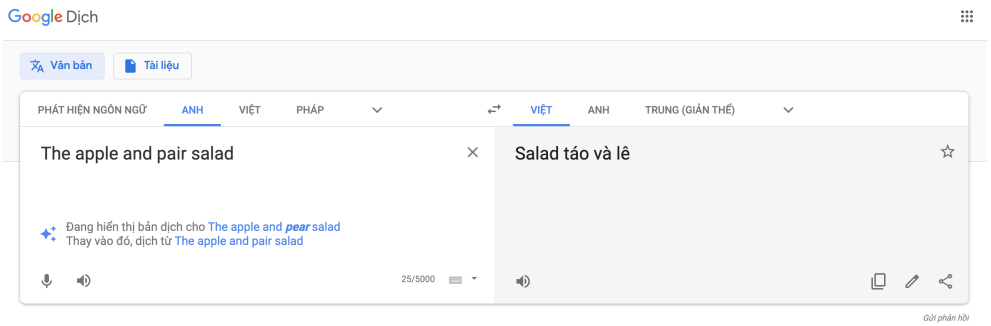
P(đâu|chúng ta không thuộc về) < P(nhau|chúng ta không thuộc về).

Việc này tương đương với so sánh:

P(chúng ta không thuộc về đâu) < P(chúng ta không thuộc về nhau).

Mô hình ngôn ngữ có thể ứng dụng vào rất nhiều việc, lĩnh vực:

- Dịch máy: P(high winds tonite) > P(large winds tonite).
- Sửa lỗi chính tả: ví dụ câu "The office is about fifteen minuets from my house". Mô hình ngôn ngữ cho ta biết: P(about fifteen miinutes from) > P(about fiIeen minuets from). Do đó có thể dùng để sửa được chính tả cho câu.
- Nhận biết giọng nói: P(The apple and pear salad) >> P(The apple and pair salad). Hai từ *pear* và *pair* phát âm giống nhau nhưng máy biết chọn đúng từ theo ngữ cảnh là nhờ mô hình hoá ngôn ngữ.



Hình 7.1: Nhờ có mô hình ngôn ngữ mà Google Translate có thể nhận biết giọng nhất (các từ đồng âm) và sửa lỗi chính tả.

7.1.2 Ước lượng xác suất và mô hình n-gram

Ước lượng xác suất

Cách đơn giản nhất để tính các xác suất là ước lượng tương đối sự xuất hiện của câu đó bằng cách đếm hết xuất hiện của câu đó trong tất cả các câu. Ví dụ:

$$P(\text{"Chúng ta không thuộc về nhau"}) = \frac{count(\text{Chúng ta không thuộc về nhau})}{count(\text{Tất cả các câu được nói/viết})}. \tag{7.3}$$

Hoặc với xác suất likelihood thì:

$$P(\text{nhau} \mid \text{chúng ta không thuộc về}) = \frac{count(\text{chúng ta không thuộc về nhau})}{count(\text{chúng ta không thuộc về})}. \tag{7.4}$$

Tuy nhiên với cách ước lượng này thì không khả thi do thực tế có rất nhiều, vô hạn chuỗi từ/câu và chúng ta không thể thấy đủ dữ liệu để ước lượng.

Do đó trên thực tế, các mô hình ngôn ngữ thường dựa vào **tính chất Markov (*Markov property*)**: tương lai không phụ thuộc quá khứ nếu biết hiện tại, cụ thể:

$$P(w_i|w_1, w_2, ..., w_{i-1}) = P(w_i|w_{i-1})$$

Ví dụ: P(nhau | chúng ta không thuộc về) = P(nhau | về).

Hay tổng quát hơn, k**-order Markov property** giả sử rằng từ tiếp theo xuất hiện trong chuỗi chỉ phụ thuộc vào k từ trước:

$$P(w_i|w_1, w_2, ..., w_{i-1}) = P(w_i|w_{1:i-1-k})$$

Mặc dầu, tính chất Markov không phải là một cách tiếp cận hoàn toàn đúng và bền vững (*robust*) (ví dụ: câu có chiều dài bất kỳ và có thể rất dài. Từ cuối lại có thể liên quan để các từ đầu tiên và ngoài khoảng k (**long-distance dependencies**). Tuy nhiên, mô hình hoá ngôn ngữ dựa vào tính chất Markov vẫn là một cách hữu hiệu và đưa ra được một mô hình hoá mạnh với một số k hợp lý.

Mô hình n-gram

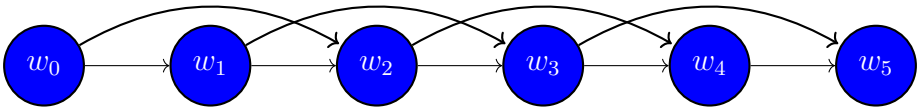
Dựa trên tính chất Markov, ta có mô hình ngôn ngữ n-gram (n-gram language model): gán xác suất dựa vào n từ (n-gram). Ý tưởng về n-gram khởi nguồn từ những công trình Lý thuyết thông tin (*Information Theory*) của *Claude Shannon* khi ông đặt câu hỏi về cho một chuỗi các ký tự thì xác suất xuất hiện (*likelihood*) của từ tiếp theo là bao nhiêu.

Trường hợp đơn giản nhất là **Unigram model** hay các từ xuất hiện sau không phụ thuộc (độc lập) với các từ phía trước.

$$P(w_1 w_2 ... w_n) \approx \prod_{i=1}^{n} P(w_i)$$

Tiếp đến là **Bigram model**: từ dự đoán phụ thuộc vào một từ trước đó.

$$P(w_i | w_1 w_2 ... w_{i-1}) \approx \prod_{i=1}^{n} P(w_i | w_{i-1})$$



Hình 7.2: Bigram Language Model.

Dựa vào Hình 7.2, ta có thấy:

- $P(w_1|w_0) = P(w_1|w_0)$
- $P(w_2|w_1 w_0) = P(w_2|w_1)$
- $P(w_3|w_2 w_1 w_0) = P(w_3|w_2)$
- ...
- $P(w_k|w_{k-1} ... w_2 w_1 w_0) = P(w_k|w_{k-1})$

Từ đó ta có thể tính

$$P(w_i|w_{i-1}) = \frac{count(w_i, w_{i-1})}{count(w_{i-1})}. \tag{7.5}$$

Ví dụ: giả sử câu bắt đầu với <s> và kết thúc với </s>. Xét 3 câu sau:

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

Ta có:

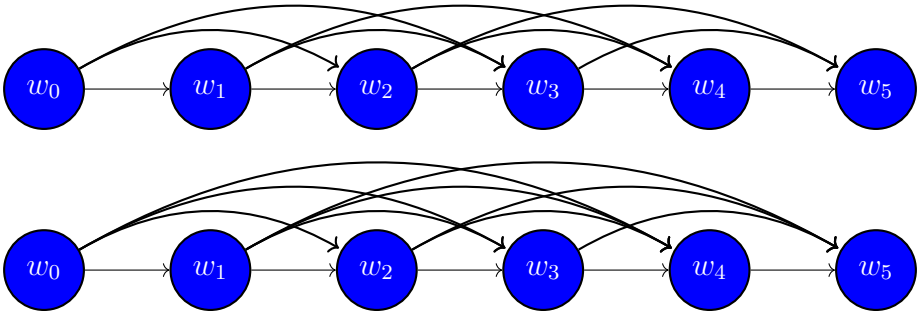
$P(I| < s >) = \frac{2}{3}$, $P(Sam| < s >) = \frac{1}{3}$,$P(am|I) = \frac{2}{3}$,

$P(< /s > |Sam) = \frac{1}{2}$,$P(Sam|am) = \frac{1}{2}$,$P(do|I) = \frac{1}{3}$.

Đối với **Trigram model** thì:

$$P(w_i|w_{i-1}w_{i-2}) = \frac{count(w_i, w_{i-1}, w_{i-2})}{count(w_{i-1}, w_{i-2})}. \tag{7.6}$$

Chúng ta có thể mở rộng ra cho trigram, 4-gram,...



Hình 7.3: n-gram Language Model: trigram và 4-gram

Thảo luận:

Như chúng ta đã biết, ngôn ngữ có sự phụ thuộc khoảng cách dài (*long distance dependencies*). Xét câu sau:

He drives a new Audi car to his penhouse. (1)

Trong câu trên để mô hình cho kết quả tốt, ta phải dùng ít nhất 7-gram. Hoặc xét câu sau:

The man who moves mountain begins by carrying away small stones. (2)

Rõ ràng từ "begins" có likelihood phụ thuộc vào từ "man" nhưng nếu n không đủ lớn thì output của mô hình ngôn ngữ sẽ không ra xác suất cao cho những câu này và ra xác suất cao cho những câu không phù hợp (có thể là dựa vào từ "mountain").

Mặt khác như n quá lớn thì cũng khó để ước lượng các xác suất bởi vì thưa dữ liệu. Ví dụ với câu (1), giả sử ta dùng 7-gram thì sẽ có $|V|^7$ biến cố (với $|V|$ là lực lượng của vocabulary), với một vocabulary rất nhỏ với $|V| = 10^4$ thì sẽ có tới 10^{28} biến cố khác nhau.

Như vậy, với việc chọn n là một **bias-variance tradeoff**. Một n nhỏ sẽ có một bias cao, và cao sẽ cho một varinace cao. Chúng ta có thể giải quyết vấn đề này bằng cách vẫn giữ nguyên n lớn và cộng với smoothing.

7.1.3 Đánh giá mô hình ngôn ngữ

Mô hình ngôn ngữ tốt là một mô hình cho xác suất các câu tốt (phù hợp và thường xuyên xuất hiện) cao hơn các câu "không" tốt (ví dụ: không phù hợp ngữ pháp và thường xuyên xuất hiện). Vì sự phức tạp của ngôn ngữ nên việc đánh giá một mô hình ngôn ngữ A tốt hơn mô hình ngôn ngữ B là rất khó khăn.

Một cách tiếp cận là **đánh giá ngoài (*extrinsic evaluation*)**. Đặt hai mô hình ngôn ngữ A và B vào một công việc cụ thể. Ví dụ: kiểm tra lỗi chính tả, dịch máy,... và kiểm tra độ chính xác cho A và B theo một metric nhất định rồi cho ra kết quả so sánh.

Ví dụ: đối với kiểm tra lỗi chính tả, ta có thể kiểm tra: Có bao nhiêu từ sai chính tả được sửa lại cho đúng.

Tuy nhiên, đối với cách đánh giá này thì vô cùng tốn thời gian (có thể mấy cả tuần, tháng) nên trong thực tế ta dùng **đánh giá trong (*intrinsic evaluation*)**: dùng độ do **Perplexity**.

Trong lý thuyết thông tin, độ do Perplexity dùng để đo một phân bố xác suất hay một mô hình xác suất có đủ tốt để dự đoán một mẫu (*sample*). Một mô hình ngôn ngữ tốt là một mô hình mà có thể dự đoán tốt trên tập dữ liệu kiểm tra (test set). Cho một corpus gồm n từ $w_1, ..., w_n$ và một mô hình ngôn ngữ gán xác suất cho một từ dựa vào lịch sử, Perplexity được tính như sau:

$$PP(p) = 2^{H(p)} = 2^{-\sum_x p_x \log_2 p_x} \tag{7.7}$$

Trong đó, $H(p)$ là hàm cross-entropy của phân bố xác suất đó.

Perplexity càng nhỏ thì mô hình ngôn ngữ càng fit tốt vì việc tối thiểu hoá perplexity sẽ tương đương với tối đa hoá xác suất. Perplexity đặc thù từng corpus nên khi so sánh hai mô hình ngôn ngữ thì ta phải chọn cùng một tập unseen corpus (test set) đánh giá.

Với một câu đủ dài (N nào đó), theo định lý ***Shannon-McMillan-Breiman*** ta có:

$$H(p) \approx -\frac{1}{N} log_2 p(W) = -\frac{1}{N} log_2 p(w_1, w_2, ..., w_N)$$

Do đó ta có thể biến đổi công thức 7.7 thành:

$$\begin{aligned} PP(p) &= 2^{-\frac{1}{N} log_2 p(w_1, w_2, ..., w_N)} \\ &= (2^{log_2 p(w_1, w_2, ..., w_N)})^{-\frac{1}{N}} \\ &= p(w_1, w_2, ..., w_N)^{-\frac{1}{N}} \\ &= \sqrt[n]{\frac{1}{p(w_1, w_2, ..., w_N)}} \end{aligned} \tag{7.8}$$

Ví dụ: cho một chuỗi số ngẫu nhiên từ 0->9, xác suất xuất hiện của từ tiếp theo biết xác xuất suất hiện của mỗi từ là 1/10 thì perplexity của mô hình này là:

$$PP(p) = p(w_1, w_2, ..., w_N)^{\frac{1}{N}} = (\frac{1}{10}^N)^{-\frac{1}{N}} = \frac{1}{10}^{-1} = 10$$

Giá trị perplexity của các mô hình ngôn ngữ n-gram khác nhau được đào tạo bằng cách sử dụng 38 triệu từ và được kiểm tra bằng cách sử dụng 1,5 triệu từ từ tập dữ

liệu của The Wall Street Journal (WSJ):

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Bảng 7.1: Perplexity tương ứng với các n-gram khác nhau

Như chúng ta đã biết, Perplexity thì mô hình ngôn ngữ càng tốtm như vậy trong ba mô hình so sánh trong Bảng 7.1 thì Trigram tốt hơn hai mô hình còn lại.

Để hiểu sâu hơn về độ đo Perplexity, ta tạm gác ngôn ngữ và gán xác suất cho từ. Giả sử mô hình dự đoán kết quả của việc tung một con xúc xắc. Một con xúc xắc thông thường có 6 mặt, do đó *hệ số phân nhánh (branching factor)* của con xúc xắc là 6. Hệ số phân nhánh cho biết có bao nhiêu kết quả có thể xảy ra bất cứ khi nào chúng ta tung.

Giả sử huấn luyện một mô hình trên con xúc xắc đều và mỗi lần tung sẽ có xác suất $\frac{1}{6}$ nhận được bất kỳ mặt nào. Sau đó, giả sử chúng ta tạo bộ thử nghiệm bằng cách tung con xúc xắc 10 lần nữa và chúng ta thu được chuỗi kết quả T = {1, 2, 3, 4, 5, 6, 1, 2, 3, 4}. Giá trị perplexity của mô hình này trên test data là bao nhiêu?

$$PP(p) = \frac{1}{(\frac{1}{6}^{10})^{\frac{1}{10}}} = 6$$

Như vậy giá trị perplexity chính là ***hệ số phân nhánh (branching factor)***.

Bây giờ, hãy tưởng tượng rằng chúng ta có một con xúc xắc không đều, mặt số 6 có xác suất xuất hiện là $\frac{7}{12}$ và tất cả các mặt còn lại có xác suất là $\frac{1}{12}$. Chúng ta huấn luyện một mô hình và sau đó tạo một test set T bằng cách tung con súc sắc 12 lần thì được mặt số 6 trên 7 tung và các số khác trên 5 tung còn lại. Giá trị perplexity:

$$PP(p) = \frac{1}{((\frac{7}{12})^7 * (\frac{1}{12})^5)^{\frac{1}{12}}} = 3.9 \approx 4$$

Giá trị perplexity lúc này thấp hơn, mô hình biết được sự xuất hiện của mặt số 6 nhiều hơn các mặt còn lại nên nó không "bất ngờ" lắm hay mô hình lúc này dự

đoán tốt hơn. Hệ số phân nhánh lúc này vẫn là 6 (có 6 mặt có thể xuất hiện) nhưng **hệ số phân nhánh có trọng số (weighted branching factor)** xấp xỉ 4. Ta có thể nói là mô hình không chắc chắn trong việc dự đoán từ 4 từ tiếp theo (thay vì 6 từ như hệ số phân nhánh ban đầu).

Rõ ràng hơn, giả sử với một con xúc xắc có không đều có xác suất xuất hiện mặt số 6 là 99% và các mặt khác chỉ $\frac{1}{500}$ trong 100 lần thử thì:

$$PP(p) = \frac{1}{((\frac{99}{100})^{99} * (\frac{1}{500})^{1})^{\frac{1}{100}}} = 1.07 \approx 1$$

Hệ số phân nhánh khi này vẫn là 6 nhưng hệ số phân nhánh có trọng số chỉ là 1. Có thể nói mô hình gần như chắc chắn mặt xuất hiện khi tung là mặt số 6 mặc dầu về lý thuyết các mặt khác vẫn có thể xuất hiện.

Trở lại với các mô hình ngôn ngữ và việc gắn xác suất cho từ, nếu chúng ta có một mô hình ngôn ngữ đang cố gắng dự đoán từ tiếp theo, thì hệ số phân nhánh đơn giản là số lượng từ có thể: bằng kích thước của từ vựng. Tuy nhiên bằng việc gán xác suất một cách "hợp lý" thì ta có thể thu giảm số này.

Ví dụ: nếu ta có giá trị perplexity là 4, thì đó *hệ số phân nhánh trung bình*. Khi cố gắng đoán từ tiếp theo, mô hình không chắc chắn như thể phải chọn giữa 4 từ khác nhau.

7.1.4 Làm trơn mô hình ngôn ngữ

Mô hình ngôn ngữ n-grams hoạt động tốt nếu tập kiểm tra giống phân bố với tập huấn luyện. Tuy nhiên trên thực tế thì các nhiều câu, chuỗi từ không biết trước và không có trong tập huấn luyện nhưng lại xuất hiện trong tập kiểm tra. Điều này làm cho việc gán xác suất bằng 0 cho các câu này (gán xác suất bằng 0 cho biến cố từ xuất hiện tiếp theo w_k nào đó và tính chất nhân tính của việc tính toán chuỗi xác suất câu).

Ví dụ: ta có tập huấn luyện gồm các câu sau:

- Chúng ta không thuộc về nhau.

- Chúng ta không thuộc về đâu.
- Chúng ta không thuộc về đây.

Nhưng tập kiểm tra thì:

- Chúng ta không thuộc về đó.
- Chúng ta không thuộc về thành phố.

Khi đó giá trị perplexity không thể tính được vì không thể chia cho xác suất 0 hay nói cách khác ta có perplexity bằng vô hạn - mô hình ngôn ngữ cực kỳ tệ.

Xét mô hình Bigram, ta có:

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}. \tag{7.9}$$

Giả sử chuỗi $< w_i, w_{i-1} >$ không xuất hiện trong tập huấn luyện thì $count(w_{i-1}, w_i) = 0$ dẫn tới $P(w_i|w_{i-1}) = 0$. Ước lượng này còn được gọi là **ước lượng hợp lý cực đại (Maximum likelihood estimation - MLE)**.

Giải quyết tình trạng xác đó ta dùng kỹ thuật *làm trơn (smoothing)* mô hình ngôn ngữ. Một trong những kỹ thuật làm trơn thường dùng nhất là **làm trơn Laplace (Laplace smoothing)**: cộng 1 cho tất cả các từ trong tập từ vựng.

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i) + 1}{count(w_{i-1}) + |V|}. \tag{7.10}$$

Trong đó $|V|$ là kích thước tập từ vượng, ước lượng này còn được gọi là **ước lượng add-1 (add-1 estimation)**. Tổng quát hơn ta có **ước lượng add-α (add-α estimation)** với $0 < \alpha \leq 1$:

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i) + \alpha}{count(w_{i-1}) + \alpha|V|}. \tag{7.11}$$

Với kỹ thuật làm trơn add-1 trong công thức 7.10, ta có thể hoàn nguyên lại số

(count) sự hiện của các từ:

$$count * (w_{i-1}, w_i) = \frac{(count(w_{i-1}, w_i) + 1) * count(w_{i-1})}{count(w_{i-1}) + |V|}. \tag{7.12}$$

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Hình 7.4: So sánh count của bigram: bảng phía trên xuất hiện nhiều count bằng 0. Bảng dưới sau khi được làm trơn bằng ước lượng add-1 và hoàn nguyên lại count.

Một trường hợp khác là mô hình n-gram nhưng ta không quan sát được đủ N khi đó một kỹ thuật làm trơn khác thường được dùng là ***back-off***: tính toán các ước lượng dựa trên $(n-1)$-gram:

$$P(w_{i+1}|w_{i-k:i}) = \beta * \frac{count(w_{i-k:i+1})}{count(w_{i-k:i})} + (1-\beta)P(w_{i+1}|w_{i-(k-1):i}). \tag{7.13}$$

Phép làm trơn trong công thức 7.13 còn gọi là ***Jelinek Mercer interpolated smoothing***.

Một cách tiếp cận khác là ***Nội suy tuyến tính (Linear Interpolation)***: trộn lẫn giữa unigram, bigram và trigram. Kí hiệu $\hat{P}$ là xác suất được làm trơn, ta có:

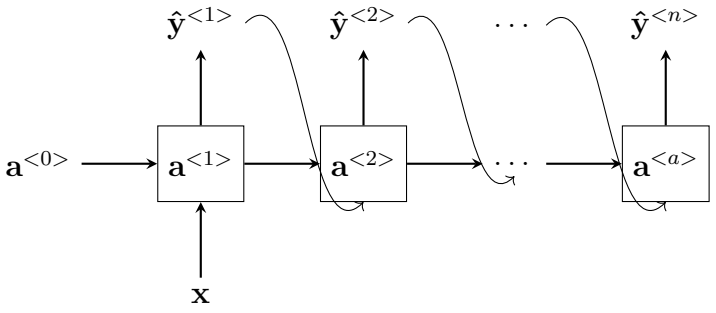
$$\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1 P(w_n|w_{n-1}w_{n-2}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)$$

Trong đó, $\lambda_1 \geq 0, \lambda_2 \geq 0, \lambda_3 \geq 0$ và $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

7.2 Mô hình ngôn ngữ với RNN

7.2.1 Ý tưởng cơ bản

Như đã biết, RNN - Recurrent Neural Network rất thích hợp để xử lí các dữ liệu dạng chuỗi, vì thế nó cũng là một mô hình rất tốt trong bài toán về mô hình hóa ngôn ngữ – Language Modelling. Ý tưởng cơ bản của việc dùng RNN để huấn luyện một mô hình ngôn ngữ cũng tương tự như dùng mô hình xác suất thống kê cổ điển. Ta sẽ tính xác suất của một câu thông qua việc tính xác suất của một từ xuất hiện khi biết các từ đã xuất hiện trước đó trong câu. Đầu vào của mạng RNN sẽ là xác suất của từng từ trong câu khi biết xác suất xuất hiện của các từ trước đó, mạng RNN sẽ tính ra xác suất của từ tiếp theo, cứ tiếp tục như thế, ta sẽ có được xác suất của một câu và cuối cùng là language model của tập huấn luyện. Dễ thấy mạng RNN được sử dụng với ý tưởng trên là một mạng one-to-many RNN như Hình 7.5

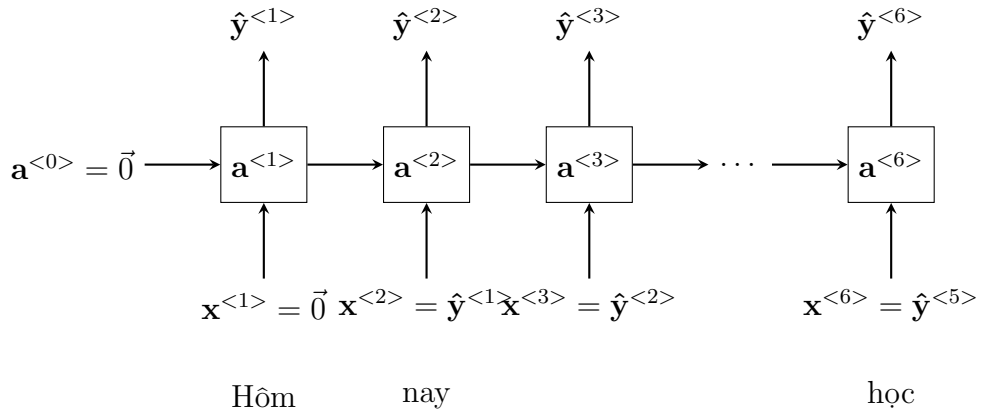


Hình 7.5: Mô hình mạng one-to-many RNN

Ví dụ 7.2.1. với câu "Hôm nay tôi đi học", ta có mạng RNN như Hình 7.6

Trong Ví dụ 7.2.1, ở đầu câu, chúng ta chưa có từ đứng trước, vì thế ta sẽ truyền vào $x^{<1>}$ và $a^{<0>}$ là vector 0 để khởi động cho mạng với ý nghĩa đầu vào là từ "mở đầu câu", như vậy:

- $\hat{y}^{<1>}$ là xác suất từ "Hôm" là xuất hiện ở đầu câu $P(Hôm)$,
- $\hat{y}^{<2>}$ là xác suất từ "nay" xuất hiện sau từ "Hôm" ở đầu câu $P(nay \mid Hôm)$,



Hình 7.6: Mô hình đơn giản thể hiện ý tưởng huấn luyện Language Model với RNN

- $\hat{y}^{<3>}$ là xác suất từ "tôi" xuất hiện sau "Hôm nay" $P(tôi \mid Hôm\ nay)$,
- $\hat{y}^{<4>}$ là xác suất từ "đi" xuất hiện sau "Hôm nay tôi" $P(đi \mid Hôm\ nay\ tôi)$,
- $\hat{y}^{<5>}$ là xác suất từ "tôi" xuất hiện sau "Hôm nay tôi đi" $P(học \mid Hôm\ nay\ tôi\ đi)$,
- $\hat{y}^{<6>}$ là xác suất câu kết thúc tại "Hôm nay tôi đi học" $P(<EOS> \mid Hôm\ nay\ tôi\ đi\ học)$, tức xác suất "Hôm nay tôi đi học" là một câu hoàn chỉnh.

Ngoài ra, chúng ta có thể tính được xác suất của một câu, một cụm từ nào đó có thể xuất hiện hay không bằng công thức tương tự như ví dụ sau:

$$\begin{aligned} P(y^{<1>}, y^{<2>}, y^{<3>}) &= P(y^{<1>})P(y^{<2>} \mid y^{<1>})P(y^{<3>} \mid y^{<1>}, y^{<2>}) \\ &= \hat{y}^{<1>}\hat{y}^{<2>}\hat{y}^{<3>} \end{aligned}$$

Với mô hình huấn luyện như trên, nếu ta sử dụng hàm kích hoạt softmax và tập dữ liệu huấn luyện là tập từ vựng trong từ điển thì kết quả đầu ra ở mỗi lần lặp là xác suất xuất hiện của mỗi từ trong từ điển nếu biết xác suất hiện của các từ đó trước đó trong câu.

7.2.2 Huấn luyện Mô hình ngôn ngữ với RNN

Tương tự với các bài toán xử lí ngôn ngữ tự nhiên khác, trước khi huấn luyện một language model, chúng ta cần có bước tiền xử lí. Ở bước này chúng ta sẽ vec-tơ hóa các từ trong từ điển bằng các phương pháp đã trình bày ở các chương trước như one-hot vector, hay các kỹ thuật word embedding khác,...

Lưu ý, khi vec-tơ hóa sẽ gặp các trường hợp một từ là số, tên riêng, các từ ít gặp, hoặc các từ, vị trí đặc biệt có ý nghĩa mà ta muốn sử dụng cho ứng dụng của mình. Đối với các từ này, ta sẽ thay thế bằng các token tổng quát. Ví dụ:

- số thay bằng <NUM>.
- các từ ít gặp thay bằng <UNK>.
- thêm các token đặc biệt như <EOS> - token kết thúc câu.

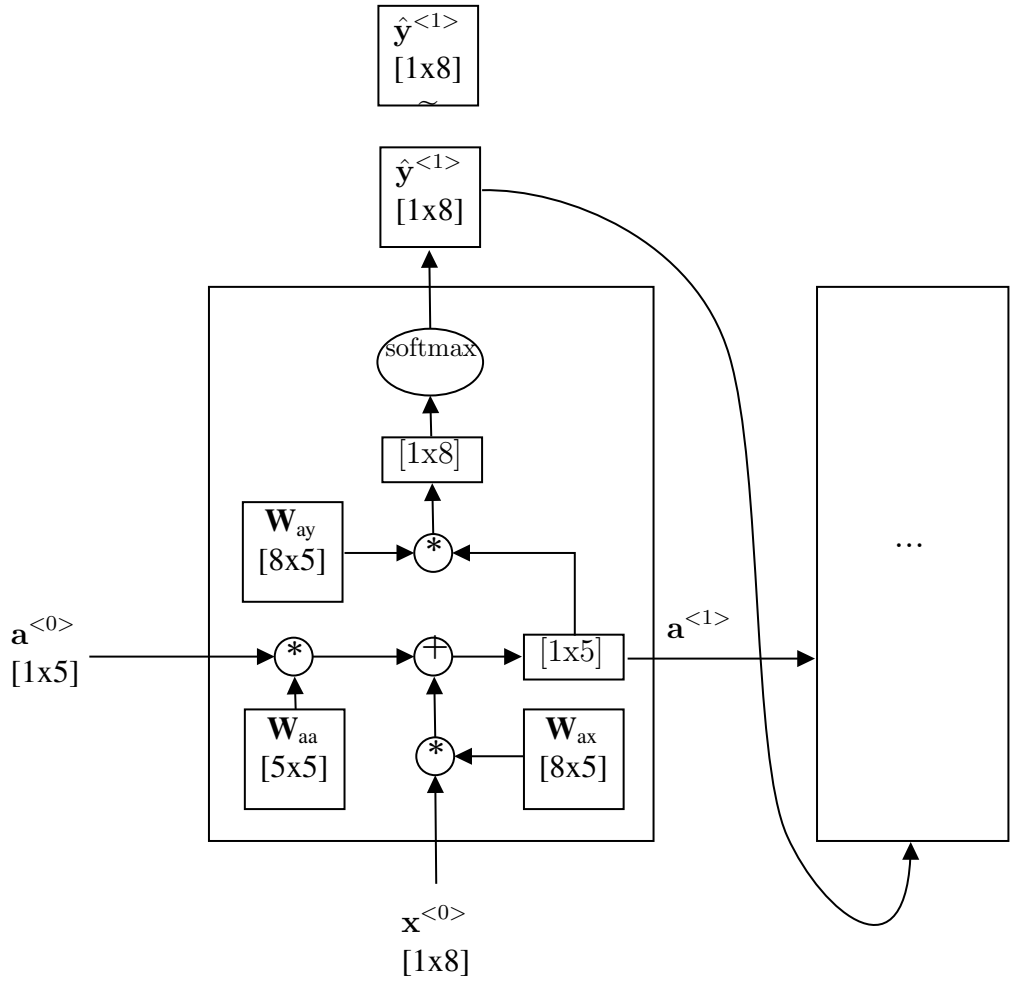
Việc dùng các token này giúp mô hình tránh được việc quá khớp với các từ khác nhau có cùng ý nghĩa hoặc vai trò trong câu như các con số, các tên riêng,... hay giúp mô hình nhận biết được một câu hoàn chỉnh bằng token <EOS>. Sau khi thực hiện tiền xử lí xong, ta sẽ dùng mạng RNN để huấn luyện language model. Cụ thể, ta cùng khảo sát Ví dụ 7.2.2.

Ví dụ 7.2.2. Huấn luyện Language Model với tài liệu:
"Hôm nay tôi đi học. Ngày mai tôi cũng đi học."

Giả sử corpus chỉ gồm các từ trong tài liệu này, sử dụng one-hot vector để biểu diễn các từ trong corpus, ta sẽ có mỗi từ là một véc-tơ 1x8. Input $x^{<0>}$ sẽ là một véc-tơ 0 với số chiều phụ thuộc vào corpus, trong trường hợp này là 1x8. Trạng thái bắt đầu $a^{<0>}$ là một vec-tơ 0 với số chiều tùy thuộc vào thiết kế của chúng ta, ở đây chọn vec-tơ 1x5.

Khi đó, W_{aa} là một ma trận 5x5, W_{ax} là một ma trận 8x5. Đầu ra của phép tính $W_{aa}a^{<i>} + W_{ax}x^{<i>}$ là một vec-tơ 1x5, trong khi đó ta cần kết quả $\hat{y}$ là một vec-tơ 1x8 để so sánh được với các véc-tơ trong corpus. Do đó, W_{ay} sẽ là ma trận 5x8.

Ta chọn hàm kích hoạt là hàm softmax để chuyển kết quả về khoảng [0, 1], tức thành xác suất rơi vào các từ trong corpus đang được biểu diễn dưới dạng one-hot. Sau đó sử dụng các phương pháp tối ưu hàm mất mát của softmax để cập nhật các ma trận trọng số.



Hình 7.7: Minh họa Ví dụ 7.2.2, huấn luyện Language Model với RNN

Hàm mất mát tại thời điểm t:

$$\mathcal{L}(\hat{y}^{<t>}, y^{<t>}) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$$

Tổng mất mát:

$$\mathcal{L} = \sum_t \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

Theo mô hình RNN trên, mất mát sẽ được tính ở cuối corpus để cập nhật trọng số. Trọng số chỉ được học hiệu quả khi corpus lớn, có thể lên đến vài tỉ từ. Việc này gây

nhiều khó khăn do giới hạn của bộ nhớ máy tính. Giải pháp cho vấn đề này là ta sẽ cắt theo một số lượng từ nhất định để tính mất mát và cập nhật trọng số, không nhất thiết phải là tại vị trí kết thúc câu.

7.3 Bài toán tạo sinh chuỗi

Có rất nhiều cách để tạo sinh chuỗi từ mạng RNN (Recurrent Neutral Network) đã được huấn luyện trước, có thể kể đến như là: Greedy Search, Bean Search, Random Sampling. Nhưng trong đó Random Sampling là một trong những cách có tính ứng dụng cao nhất. Trong phạm vi chương này chúng ta chỉ tìm hiểu về cách tạo sinh chuỗi bằng phương pháp Radom Sampling. Bạn đọc có thể tìm hiểu hai cách còn lại tại website của Daniël Heres.

7.3.1 Tạo mẫu từ phân bố xác suất bất kì

Tạo mẫu bất kì từ một phân bố xác suất cho trước là một bước quan trọng trong bài toán tạo sinh chuỗi. Như chúng ta đã biết từ các chương trước, đầu ra của mạng RNN là một phân bố xác suất rời rạc, bởi vì hàm softmax được kích hoạt ở tầng cuối của mạng. Từ các phân bố xác suất này chúng ta phải chọn ra những từ phù hợp với từng xác suất của nó. Để làm được điều đó, chúng ta cần tìm hiểu hai khái niệm: tạo mẫu từ phần bố đều và hàm phân phối tích lũy.

Phân bố đều (uniform distribution) là phân bố có biến ngẫu nhiên nhận giá trị trên đoạn $[a, b]$. Xác xuất X nhận bất kỳ giá trị nào thuộc khoảng $[a, b]$ đều bằng $\frac{1}{b-a}$. Chính tính chất này nên phân bố đều được áp dụng để phát sinh số ngẫu nhiên, đảm bảo các giá trị được sinh ra đều các khả năng như nhau.

Hàm phân phối tích lũy (cumulative distribution function - *cdf*) là hàm mô tả đầy đủ phân phối xác suất của biến ngẫu nhiên có giá trị thực x. Với mỗi số thực x, *cdf* được định nghĩa như sau:

$$F(x) = P(X <= x)$$

trong đó vế phải biểu diễn xác suất mà biến ngẫu nhiên X lấy giá trị nhỏ hơn hay bằng x. Để hiểu rõ hơn về *cdf* chúng ta cùng xem ví dụ sau. Giả sử chúng ta có

phân bố xác suất của các từ cần phát sinh là.

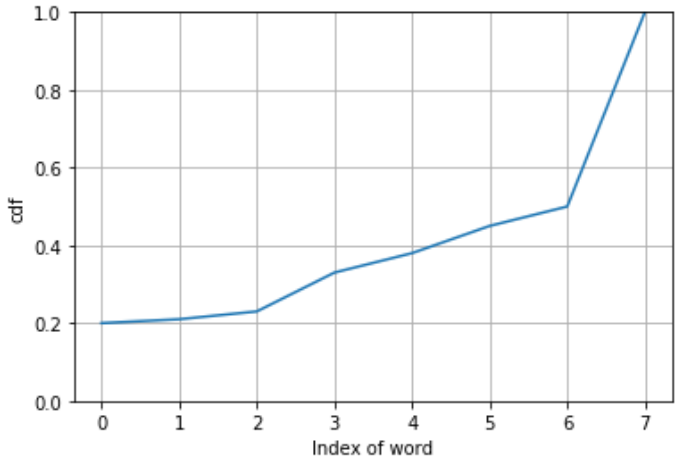
$$P(x) = [0.2,\ 0.01,\ 0.02,\ 0.1,\ 0.05,\ 0.07,\ 0.05,\ 0.5]$$

với mỗi xác suất trong P tương ứng với các từ

$$[\text{"}Hom\text{"},\ \text{"}nay\text{"},\ \text{"}toi\text{"},\ \text{"}di\text{"},\ \text{"}hoc\text{"},\ \text{"}Ngay\text{"},\ \text{"}mai\text{"},\ \text{"}cung\text{"}]$$

Hàm phân phối tích lũy của ví dụ trên được tính và có dạng như hình dưới đây.

$$F(x) = [0.2,\ 0.21,\ 0.23,\ 0.33,\ 0.38,\ 0.45,\ 0.5,\ 1.0]$$



Hình 7.8: Hàm phân phối tích lũy của các từ

Sau khi xác định được hàm phân phối xác suất của các từ, kết hợp với việc sinh số ngẫu nhiền từ phân bố đều trên đoạn $[0, 1]$. Chúng ta dễ dàng phát sinh các từ trên theo công thức.

$$O(x) = \begin{cases} "Hom" \ if \ 0 <= x < 0.2 \\ "nay" \ if \ 0.2 <= x < 0.21 \\ "toi" \ if \ 0.21 <= x < 0.23 \\ "di" \ if \ 0.23 <= x < 0.33 \\ "hoc" \ if \ 0.33 <= x < 0.38 \\ "Ngay" \ if \ 0.38 <= x < 0.45 \\ "mai" \ if \ 0.45 <= x < 0.5 \\ "cung" \ if \ 0.5 <= x <= 1.0 \end{cases}$$

trong đó x là kết quả của việc sinh số ngẫu nhiên từ phân bố đều.

Như vậy theo phương pháp trên chúng ta có thể hoàn toàn phát sinh một mẫu ngẫu nhiên từ một phân bố xác xuất cho trước. Trong thực tế chúng ta có thể sử dụng các hàm có sẵn từ numpy để tiết kiệm thời gian tính toán. Bạn đọc có thể tham khảo hàm *numpy.random.choice* tại Numpy documentation (C. R. Harris et al., 2020). Bản chất của hàm này cũng áp dụng những bước trên.

7.3.2 Sinh chuỗi từ mạng RNN đã được huấn luyện

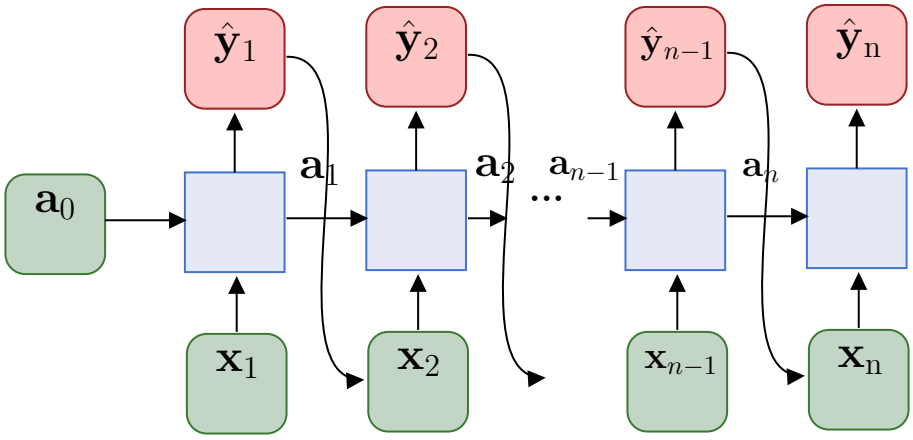
Từ những hiểu biết trên, ta có thể thiết kế một mạng RNN dùng để tự động viết tiếp chuỗi đã được cho trước dựa vào corpus có sẵn và mạng RNN đã được huấn luyện từ trước.

Mô tả bài toán

Đầu vào của bài toán là corpus của một ngôn ngữ (tiếng Anh, tiếng Việt,...), các đoạn văn bản mẫu (Truyện Kiều của Nguyễn Du, thơ Xuân Diệu, thơ Shakespeare,...) tương ứng với corpus đã cho. Từ corpus và tập văn bản mẫu này, ta sẽ huấn luyện được một mạng RNN có khả năng sinh ra các văn bản mới có cùng phong cách viết với tác giả của các đoạn văn bản mẫu.

Cấu trúc mạng RNN dùng để sinh chuỗi

Mạng RNN sử dụng trong phần này được thiết kế để sinh chuỗi. Trong phần này, ta sẽ không nhắc lại cách huấn luyện mạng RNN và giả sử mạng đã được huấn luyện (sử dụng các kỹ thuật đã được đề cập ở các phần trước)



Hình 7.9: Cấu trúc mạng RNN

Hình 7.9 miêu tả quá trình sinh chuỗi sử dụng mạng RNN được huấn luyện sẵn. Giả sử ta huẩn luyện mạng RNN với corpus Tiếng Việt và sử dụng thơ Truyện Kiều của Nguyễn Du. Ta mong đợi mạng sau khi được huấn luyện có thể tự động sinh ra các câu thơ với phong cách Nguyễn Du.

Để cho dễ hiểu, ta giả sử corpus có 10 chữ: *["Đầu", "lòng", "hai", "ả", "tố", "nga", "Thúy", "Kiều", <SOS>, <EOS>]*. Trong đó *<SOS>* là start of string, *<EOS>* là end of string.

Đầu tiên, ta khởi tạo trạng thái a_0 bằng vector 0, x_1 là token đầu tiên trong câu nên ta sẽ gán nó là token *<SOS>* để đánh dấu sự bắt đầu của câu mới.

Thực hiện forward bước đầu, ta thu được trạng thái a_1 và một phân phối xác suất ở ngõ ra $\hat{y}_1$. Phân phối xác suất này cho ta biết xác suất xuất hiện của từ tiếp theo trong câu. Ví dụ, nếu phân phối xác suất ở ngõ ra $\hat{y}$ có dạng:

$$\hat{y} = [0.2, 0.3, 0.05, 0.05, 0.01, 0.1, 0.09, 0.05, 0.05, 0.1]$$

Ta sẽ sử dụng phân phối này để lấy mẫu ngẫu nhiên (áp dụng giải thuật lấy mẫu nêu ở phần trên). Giải thuật này có thể cho ra bất kì từ nào trong corpus có xác suất tại $\hat{y}_1$ lớn hơn 0. Giả sử bộ lấy mẫu cho ra từ *"Đầu"*, ta sẽ gán $x_2 =$ *"Đầu"* và tiếp tục dự đoán $\hat{y}_2$.

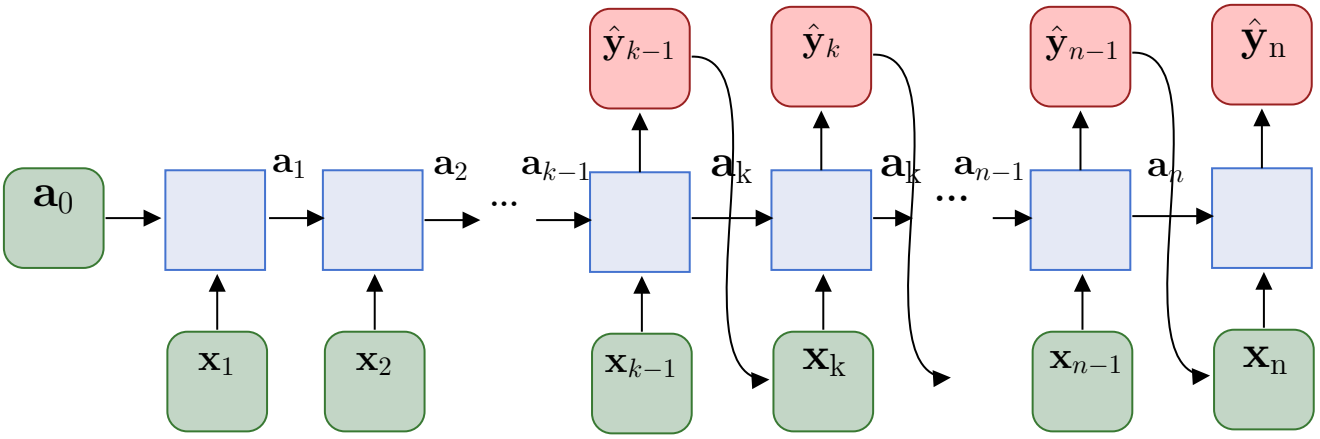
Lúc này, trạng thái a_1 cũng được đưa vào đầu vào trạng thái của mạng, ta tiếp tục tính được phân phối xác suất tại $\hat{y}_2$ và bộ lấy mẫu trả về kết quả từ *"hai"*. Cứ tiếp tục như vậy, bộ có thể sinh ra một câu văn hoàn chỉnh như sau: *"Đầu lòng hai ả tố nga <EOS>"*. Câu văn kết thúc khi *<EOS>* được sinh ra.

Sinh chuỗi với các từ được gợi ý

Để mạng RNN sinh chuỗi đúng với chủ đề mà ta muốn, ta có thể gợi ý cho mạng RNN một số từ chủ đề ban đầu và để nó tự động sinh ra các từ để hoàn thành câu.

Để làm được như vậy, ta vẫn sẽ sử dụng mạng RNN đã được huấn luyện như phần trên. Nhưng lần này, thay vì mở đầu bằng token *<SOS>*, ta sẽ đưa vào $[x_1, x_2, x_3, ..., x_k]$ k từ đầu tiên được gợi ý.

Ví dụ ta gợi ý 2 từ *"Thúy Kiều"*, ở lần forward đầu tiên, a_0 sẽ được gán bằng vector 0, x_1 sẽ là vector của từ *"Thúy"*. Mạng sẽ tính ra $\hat{y}_1$ và a_1. Tuy nhiên lần này ta sẽ không quan tâm đến $\hat{y}_1$, ta chỉ cần truyền a_1 đến ngõ vào tiếp theo của mạng. Ở lần forward thứ 2, x_2 sẽ là vector của từ *"Kiều"*, ta tính được $\hat{y}_2$ và a_2. Vì *" Kiều"* là từ cuối cùng được gợi ý, nên $\hat{y}_2$ trong trường hợp này sẽ được nhận làm x_3 và a_2



Hình 7.10: Cấu trúc mạng RNN sinh chuỗi với các từ được gợi ý

sẽ được đưa đến bước tính tiếp theo. Từ đây, các từ sẽ được sinh ra cho tới khi câu được hoàn thành.

Với cách sinh chuỗi này, lượng từ gợi ý càng nhiều thì các từ sinh ra càng tự nhiên và đúng chủ đề hơn. Nguyên nhân là do mạng RNN có khả năng ghi nhớ chủ đề và trạng thái thái sẽ được điều chỉnh lại qua mỗi từ gợi ý để sinh ra các phân phối xác suất chuẩn xác hơn ở những lần sinh mẫu sau.

Sinh chuỗi ở cấp độ từ và cấp độ ký tự

Phần trên đã mô tả việc sinh mẫu ở cấp độ từ. Như ta thấy, để sinh ra các câu văn cho một ngôn ngữ nào đó, ta cần một lượng từ vựng rất lớn, vì vậy corpus cho việc sinh mẫu cũng lớn (với tiếng Anh có thể lên đến hơn 10000 từ). Đồng thời, trước khi huấn luyện, ta phải qua bước tiền xử lý *word2vec* (*skip-gram* hoặc *CBOW* để ghi lại ngữ cảnh của từ). Đôi khi, có một khả năng từ được sinh ra là token *<UNK>* (Unknown).

Việc sinh mẫu ở cấp độ từ gây ra nhiều sự phức tạp và khó khăn, và đôi khi cho kết quả không tốt. Thay vào đó ta sẽ xem xét việc sử dụng các ký tự trong corpus thay vì các từ. Điều này cho phép chúng ta giảm bớt sự phức tạp trong việc tiền xử lý dữ liệu và giảm thiểu việc sử dụng bộ nhớ.

Để áp dụng mô hình sinh mẫu bằng ký tự, đầu tiên ta thực hiện việc thiết lập corpus cho ngôn ngữ. Ví dụ ta muốn sinh các câu văn tiếng Anh, ta sẽ phải thiết lập một

corpus bao gồm:

- Các ký tự a-z và A-Z
- Các số từ 0-9
- Các kí tự dấu câu, dấu cách, các kí tự đặc biệt nếu có

Sau khi thiết lập được corpus, ta cần phải áp dụng các bước tiền xử lý cho các văn bản mẫu. Các bước xử lý này là các bước xử lý đơn giản bao gồm việc bỏ đi các kí tự không có trong corpus và sử dụng *one-hot* vector cho từng ký tự.

Sau khi tất cả dữ liệu cho việc huấn luyện đã sẵn sàng, ta có thể thiết lập mạng RNN và bắt đầu việc huấn luyện tương tự như trên mô hình RNN ở cấp độ từ. Sau khi được huấn luyện, mô hình RNN sẽ có khả năng sinh ra các ký tự và tạo thành các đoạn văn bản có phong cách tương tự như phong cách của các văn bản huấn luyện.

Ưu điểm của việc sinh mẫu theo kí tự so với việc sinh mẫu theo từ là chất lượng văn bản được sinh ra thường sẽ tốt hơn, không sinh ra token *<UNK>*, tiết kiệm bộ nhớ hơn do sử dụng corpus nhỏ, không tốn nhiều công sức cho việc tiền xử lý dữ liệu. Tuy nhiên nó có thời gian huấn luyện lâu và chi phí tính toán cao hơn do mô hình này coi mỗi kí tự là một token thay vì mỗi từ là một token như mô hình từ vựng.

Hiện nay, mô hình sinh mẫu từ từ vựng trên thực tế vẫn chiếm đa số. Tuy nhiên với sức mạnh xử lý ngày càng cao của máy tính hiện đại, người ta đang bắt đầu áp dụng các mô hình sinh mẫu dựa trên kí tự vào thực tế ngày một nhiều hơn.

Các ứng dụng thực tế

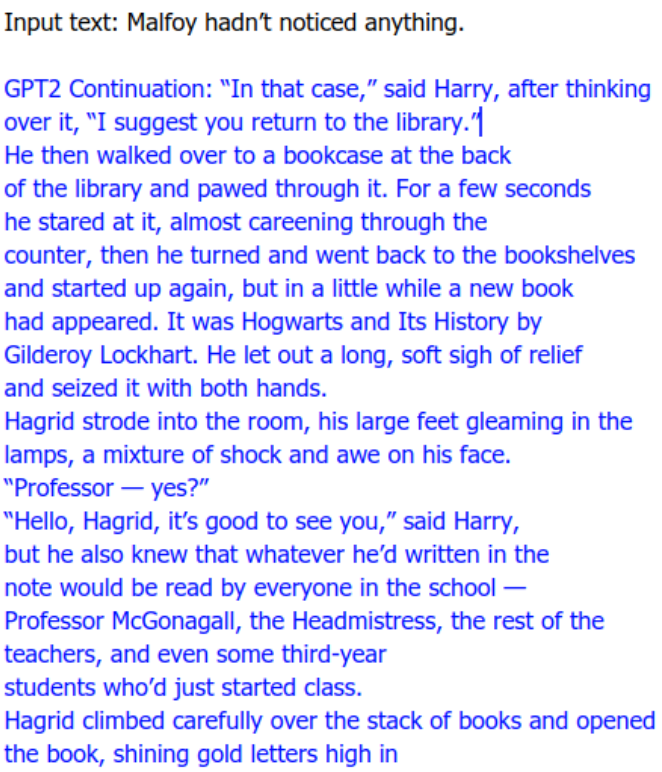
Hiện tại, việc sử dụng mạng RNN đã trở nên phổ biến trong nghiên cứu cũng như ứng dụng thực tế. Facebook, Google và các hãng công nghệ khác đã tạo ra những mô hình RNN để phục vụ cho sản phẩm của họ. Trong phần này chúng ta sẽ tìm hiểu về *GPT-2*, một mạng RNN được xây dựng bởi *OpenAI* và một mạng RNN được huấn luyện để sinh ra các bài báo tiếng Việt.

GPT-2 được hoàn thành vào năm 2019 bởi OpenAI. Mạng RNN này được huấn luyện với tập dữ liệu 40GB ngôn ngữ tiếng Anh bao gồm các bài báo và bình luận trên internet. Mạng này có thể cho ra các câu văn rất hoàn thiện, đúng ngữ pháp

và rất tự nhiên. Do lo sợ việc sẽ có người dùng mạng RNN này để tạo nên các tin giả, tác giả của mạng đã không công bố mạng hoàn chỉnh mà chỉ đưa ra cho cộng đồng một mạng được huấn luyện với tập dữ liệu nhỏ hơn. Mặc dù mạng được công bố nhỏ hơn mạng được tạo ra trong phòng thí nghiệm nhiều lần, nhưng kết quả của nó đã rất tốt và gây ấn tượng mạnh cho giới khoa học.

Hiện tại, cấu trúc của mạng này đã được công bố, nhưng trong khuôn khổ bài viết này, chúng ta sẽ không đề cập sâu đến kiến trúc của nó. Bạn đọc có thể tham khảo thêm cách để huấn luyện mạng *GPT-2* ở đây: https://minimaxir.com/2019/09/howto-gpt2/

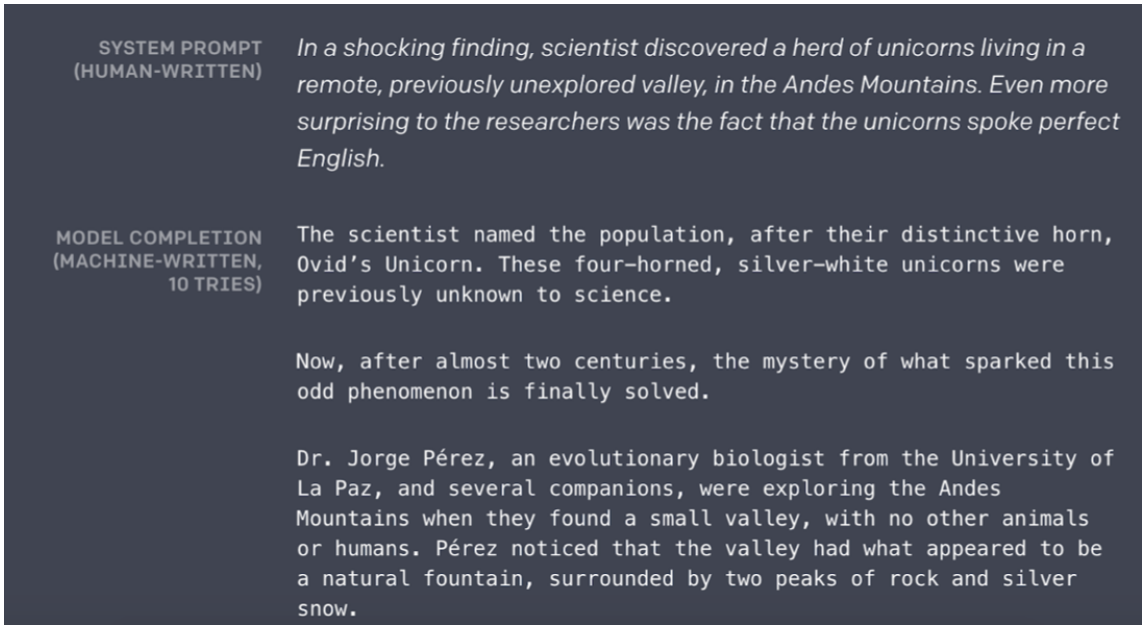
Người ta cho mạng *GPT-2* học truyện Harry Potter, sau đó cho nó một câu gợi ý (Malfoy hadn't noticed anything), và sau đây là câu chuyện được viết ra:



Hình 7.11: Harry Potter được viết bởi *GPT-2*

Ta có thể thấy văn bản được viết đúng chính tả, ngữ pháp và có văn phong tự nhiên rất giống với tác giả J.K Rowling.

Sau đây là một ví dụ khác khi *GPT-2* được huấn luyện để viết báo:



Hình 7.12: Một bài báo được viết bởi *GPT-2*

Ta có thể thấy với một đoạn văn gợi ý chủ đề được viết bởi con người ("In a shocking finding ..."), *GPT-2* đã cho ra một bài viết có văn phong nghị luận xã hội tương tự như các bài báo nó được huấn luyện.

Tiếp theo, ta sẽ xem qua một ví dụ khác về sinh văn bản tiếng Việt. Ta lần lượt cho mạng RNN học các văn bản tin tức và Truyện Kiều, sau đây là kết quả:

```
In [12]: generate_text('roger federer', 400)

Out[12]: 'roger federer là một trong những người đã có thể thấy được chiến thắng trong cuộc đua vô địch wimbledon N đã có mặ
         t tại trận chung kết giải quần vợt thế giới với tỷ số N N N N N N N trong trận đấu này đã có N lần thi đấu vào tứ k
         ết us open N trước khi giành chiến thắng N N N N N trước tay vợt người serbia vừa chiếm tỷ lệ N N N N N trong khi đ
         ó trận đấu ra sân của mình và đối thủ của federer đã có N lần gặp nha'

In [22]: generate_text('novak djo', 400)

Out[22]: 'novak djokovic và chuyển sang thi đấu trên sân cỏ những cú đánh thuận tay khi trở lại thi đấu chuyên nghiệp và đó
         là một chấn thương của nadal sau khi công bố người thắng cuộc N trước khi bước vào set N với tỷ số N N N N N N N N
         N N N N N N N N sam querrey và colombia N N N N N marin cilic N N N N N N N N N N N N N N N N N N N N N N N N N N N
         N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N '

In [27]: generate_text('andy murr', 400)

Out[27]: 'andy murray đã có mặt tại các trận đấu như vậy thì chỉ cần thi đấu không thể trở lại trong năm nay khi thi đấu chỉ
         có thể trở lại thì mới có thể nhận thấy thực sự cho thấy sự cố gắng thi đấu của mình trong sự nghiệp của mình trước
         khi chia tay sau N set đấu này cũng có thể sẽ nhận được sự chú ý của giới chuyên môn và cho thấy sự thay đổi của cá
         c tay vợt trẻ như anh là chuyện thi đấu của mình trong khi đó a'

In [28]: generate_text('bitco', 400)

Out[28]: 'bitcoin theo đó việc chuyển đổi số thuế thu nhập của tập đoàn là N triệu usd trong tháng N theo báo cáo của uber c
         ho biết thêm các tổ chức tín dụng của hệ thống tài chính tại việt nam cho thấy trong tháng N cho thấy sự thay đổi c
         ủa thị trường trong nước và quốc tế và chính phủ trung quốc có thể thấy rằng những thách thức trong các nền kinh tế
         apec cho thấy thị trường chứng khoán việt nam có thể sẽ tăng'

In [29]: generate_text('apple', 400)

Out[29]: 'apple cho biết chúng tôi cần phải có thêm một số trường hợp thu thập thông tin cho người tiêu dùng cho các nhà mạn
         g này như thế nào các chuyên gia nhận định đây chính là một trong những nguyên nhân chính trong thị trường những ng
         ày qua và đó là một chiếc điện thoại thông minh của mình và các nhà phân tích cho rằng chúng ta có thể giúp các nhà
         cung cấp dịch vụ công nghệ thông tin trên thế giới và chúng '

In [40]: generate_text('micros', 400)

Out[40]: 'microsoft theo chuyên gia giáo dục việt nam cho biết họ đã chọn thời gian tham gia các hoạt động của các nhà trườn
         g trong trường học thuộc các ngành nghề theo học bạn có thể tham gia các hoạt động thi thpt quốc gia theo thời gian
         thực tế của nhà trường và chủ nhân của trường đại học tân tạo cho biết thêm vào đó những người làm công tác tuyển c
         họn các điều kiện thuận lợi cho các em học sinh tham gia các '
```

Hình 7.13: Mạng RNN viết báo tiếng Việt

```
<|startoftext|>Hôm nay Trọng đến tìm Kiều,
Thiếp hồng tìm đến hương khuê gửi vào.<|endoftext|>
<|startoftext|>Trướng tô giáp mặt hoa đào,
Vẻ nào chẳng mặn nét nào chăng ưa?<|endoftext|>
<|startoftext|>Hải đường mơn mởn cành tơ,
Ngày xuân càng gió càng mưa càng nồng.<|endoftext|>
<|startoftext|>Nguyệt hoa hoa nguyệt não nùng,
Đêm xuân ai dễ cầm lòng được chăng?<|endoftext|>
<|startoftext|>Lạ gì thanh khí lẽ hằng,
M
====================
<|startoftext|>Hôm nay Trọng đến tìm Kiều,
Những điều vàng đá phải tìm lồng.<|endoftext|>
<|startoftext|>Ai ngày luống những âm tình,
Những là e ấp dạ tinh phận màu.<|endoftext|>
<|startoftext|>Nàng rằng: Thanh có thế giữa trời,
Bỗng không cá nước chim trời lỡ nhân.<|endoftext|>
<|startoftext|>Thương tình con trẻ cha già,
Những là đắp ngọc thử mình sau bỗng chiêm.<|endoftext|>
<|startoftext|>Về đây nước trước bẻ hoa,

====================
<|startoftext|>Hôm nay Trọng đến tìm Kiều,
Thiếp hồng tìm đến thì thôi đồng.<|endoftext|>
<|startoftext|>Những là cười phấn cợt son,
Đèn khuya chung bóng trăng tròn sánh vai.<|endoftext|>
<|startoftext|>Thú quê thuần hức bén mùi,
Giếng vàng đã rụng một vài lá ngô.<|endoftext|>
<|startoftext|>Chạnh niềm nhớ cảnh giang hồ,
Một màu quan tái mấy mùa gió trăng.<|endoftext|>
<|startoftext|>Tình riêng chưa dám rỉ răng,
Tiểu thư trước 
====================
<|startoftext|>Hôm nay Trọng đến tìm Kiều,
Thiếp hồng tìm đến hương án phồng.<|endoftext|>
<|startoftext|>Trở về minh bạch nói tường:
Mặt nàng chẳng thấy việc nàng đã tra.<|endoftext|>
<|startoftext|>Người này nặng kiếp oan gia,
Còn nhiều nợ lắm sao đà thoát cho!<|endoftext|>
<|startoftext|>Mệnh cung đang mắc nạn to,
Một năm nữa mới thăm dò được tin.<|endoftext|>
<|startoftext|>Hai bên giáp mặt chiền chiền,
Muốn
====================
<|startoftext|>Hôm nay Trọng đến tìm Kiều,
Cám lòng mấy nợ thầm mà xuân!<|endoftext|>
<|startoftext|>Những là cười phấn cợt son,
Nghe ra ngất đủ dứt tình càng mới thôi!<|endoftext|>
<|startoftext|>Nàng rằng: Trống bước lưu ly,
Phải tên xưng xuất là thằng bán tơ.<|endoftext|>
<|startoftext|>Một nhà hành một nhà hành,
Một ngày nỗi cảnh một đời xa,<|endoftext|>
<|startoftext|>Nghe càng đắm ngắm càng say,
Lạ cho m
====================
```

Hình 7.14: Mạng RNN sáng tác truyện Kiều

7.4 Sử dụng mô hình ngôn ngữ để sửa lỗi kết quả của các mô hình khác

7.4.1 Cơ chế sửa lỗi của mô hình ngôn ngữ

Mô hình ngôn ngữ (Language Model) thường được sử dụng để sữa lỗi kết quả của những bài toán như Speech to Text hoặc Image to Text.

Xét ví dụ sau:

- Câu dự đoán của mô hình nhận diện giọng nói:

  The apple and pair salad.

- Câu đúng:

  The apple and pear salad.

Trường hợp từ *"pear"* bị dự đoán nhầm thành từ *"pair"* thì mô hình sẽ tính được xác suất của câu *"The apple and pair salad"* sẽ thấp và xác suất của câu *"The apple and pear salad"* sẽ khá cao. Câu hỏi đặt ra là làm sao mô hình tính được xác suất này.

Đó là mô hình sử dụng xác suất có điều kiện để tính được xác suất nêu trên. Tức là đầu tiên sẽ tính xác suất từ *"The"* xuất hiện ở đầu câu, sau đó sẽ tính tiếp trường hợp đầu câu là từ *"The"* và tiếp đó là từ *"apple"* thì xác suất là bao nhiêu. Cứ như thế lần lượt cho *"The apple and"*, *"The apple and pair"* và cuối cùng là khi đã có xác suất của *"The apple and pair"* thì tính tiếp xác suất xuất hiện của từ tiếp đó là *"salad"* là bao nhiêu.

Vậy làm sao để tính được xác suất của từng phần đó. Trước tiên đối với từ *"The"* thì ta sẽ đếm có bao nhiêu câu bắt đầu bằng từ *"The"* trong bộ dữ liệu và từ đó sẽ tính được xác suất của câu bắt đầu bằng từ *"The"*. Tương tự như thế, ta cũng đếm tiếp có bao nhiêu câu bắt đầu bằng từ *"The"* mà tiếp sau đó là từ *"apple"* thì ta cũng tính được xác suất của *"The apple"*. Dựa vào đó ta có thể tính được xác suất cho từng phần đã nêu trên như sau:

$$P1 = P(The) \tag{7.14}$$

$$P2 = P(apple|The) * P1 \tag{7.15}$$

$$P3 = P(and|The\ apple) * P2 \tag{7.16}$$

$$P4 = P(pair|The\ apple\ and) * P3 \tag{7.17}$$

$$P5 = P(salad|The\ apple\ and\ pair) * P4 \tag{7.18}$$

$$P(The\ apple\ and\ pair\ salad) = P5 \tag{7.19}$$

Sau khi tính được xác suất của câu *"The apple and pair salad"* thì làm sao chúng ta nhận biết được câu này đúng hay sai. Như đã nói ở trên thì sau khi tính được xác suất của câu có từ *"pair"* ta nhận thấy xác suất này khá thấp, so với một *mức* (threshold) đã xác định, nên ta sẽ nghi ngờ rằng câu này chưa đúng. Và cụ thể hơn để xác định từ nào sai thì ta cũng dựa vào xác suất xuất hiện của từng từ được tính toán như trên, đến từ *"pair"* thì xác suất sẽ thấp nên ta có thể nghi ngờ từ *"pair"* là lỗi và sau đó tìm từ thích hợp hơn để thay thế.

Vậy nếu chỉ tính xác suất của cụm từ ngắn hơn trong câu mà có bao gồm từ sai thì có thể nghi ngờ câu đó là câu sai hay không, ví dụ như chỉ tính xác suất của cụm *"and pair"* thì có thể nghi ngờ câu *"The apple and pair salad"* là câu sai hay không.

Việc tính xác suất của chỉ một cụm từ ngắn thì không thể nghi ngờ được câu đang xét có sai hay không bởi xác suất tính được sẽ vẫn đủ cao để không bị nghi ngờ. Vì cụm từ xét tới có thể xuất hiện trong bất cứ ngữ cảnh nào của bộ dữ liệu, tức là cụm đó còn có thể xuất hiện nhiều lần hơn cả cụm đúng mà ta cần sửa lỗi và mức thông tin mà cụm đó cung cấp sẽ thấp bởi vì tính phổ thông của nó lớn, trừ khi cụm xét tới là tên riêng hoặc những cụm từ ít được sử dụng. Vì thế nếu muốn tính đủ xác suất để có thể nghi ngờ từ sai thì chúng ta phải xét cả phần ngữ cảnh trước đó, phần trước đó càng dài thì xác suất sẽ đủ thấp để nghi ngờ từ sai, tức là lượng

thông tin được cung cấp sẽ càng nhiều và khả năng phát hiện lỗi sẽ càng cao. Giả sử bộ dữ liệu gồm 100,000,000 câu, trong đó có:

- 25,000,000 câu bắt đầu bằng *"The"*.
- 10,000 câu bắt đầu bằng *"The apple"*.
- 3,000 câu bắt đầu bằng *"The apple and"*.
- 5 câu bắt đầu bằng *"The apple and pair"*.
- 250 câu bắt đầu bằng *"The apple and pear"*.
- 400 câu bắt đầu bằng *"The apple and strawberry"*.
- 1 câu bắt đầu bằng *"The apple and pair salad"*.
- 160 câu bắt đầu bằng *"The apple and pear salad"*.
- 220 câu bắt đầu bằng *"The apple and strawberry salad"*.
- 85,000,000 câu có từ *"and"*.
- 13,000 câu có cụm *"and pair"*.
- 9,000 câu có cụm *"and pear"*.
- 11,000 câu có cụm *"and strawberry"*.

Ta có thể tính được các xác suất xuất hiện như sau:

$$P(The) = \frac{25,000,000}{100,000,000} = 0.25 \tag{7.20}$$

$$P(The\ apple) = \frac{10,000}{25,000,000} * 0.25 = 10^{-4} \tag{7.21}$$

$$P(The\ apple\ and) = \frac{3,000}{10,000} * 10^{-4} = 3 * 10^{-5} \tag{7.22}$$

$$P(The\ apple\ and\ pair) = \frac{5}{3,000} * 3 * 10^{-5} = 5 * 10^{-8} \tag{7.23}$$

$$P(The\ apple\ and\ pear) = \frac{250}{3,000} * 3 * 10^{-5} = 2.5 * 10^{-6} \tag{7.24}$$

$$P(The\ apple\ and\ strawberry) = \frac{400}{3,000} * 3 * 10^{-5} = 4 * 10^{-6} \tag{7.25}$$

$$P(The\ apple\ and\ pair\ salad) = \frac{1}{5} * 5 * 10^{-8} = 10^{-8} \tag{7.26}$$

$$P(The\ apple\ and\ pear\ salad) = \frac{160}{250} * 2.5 * 10^{-6} = 1.6 * 10^{-6} \tag{7.27}$$

$$P(The\ apple\ and\ strawberry\ salad) = \frac{220}{400} * 4 * 10^{-6} = 2.2 * 10^{-6} \tag{7.28}$$

Ta có thể thấy xác suất khi xuất hiện từ *"pair"* (công thức 7.23) khá thấp so với *"pear"* (công thức 7.24) và *"strawberry"* (công thức 7.25). Về *mức* (threshold) để xác định từ nghi ngờ là từ sai thì chúng ta có thể giả sử nó sẽ bằng $\frac{1}{100}$ xác suất liền trước (coi như tính cho từ ở vị trí thứ tư). Khi đó:

$$Threshold(The\ apple\ and\ *) = \frac{1}{100} * P(The\ apple\ and) = 3 * 10^{-7} \tag{7.29}$$

Vì xác suất khi suất hiện của từ *"pair"* thấp hơn threshold đã tính (công thức 7.29) nên ta sẽ nghi ngờ *"pair"* là từ sai. Và ta cũng thấy xác suất của câu *"The apple and pair salad"* (công thức 7.26) thấp hơn hẳn *"The apple and pear salad"* (công thức 7.27) và *"The apple and strawberry salad"* (công thức 7.28).

Với trường hợp tính cho cụm từ ngắn:

$$P(and) = \frac{85,000,000}{100,000,000} = 0.85 \tag{7.30}$$

$$P(and\ pair) = \frac{13,000}{85,000,000} * 0.85 = 1.3 * 10^{-4} \tag{7.31}$$

$$P(and\ pear) = \frac{9,000}{85,000,000} * 0.85 = 0.9 * 10^{-4} \tag{7.32}$$

$$P(and\ strawberry) = \frac{11,000}{85,000,000} * 0.85 = 1.1 * 10^{-4} \tag{7.33}$$

Ta có thể thấy xác suất của *"and pair"* (công thức 7.31) còn cao hơn cả *"and pear"* (công thức 7.32) và *"and strawberry"* (công thức 7.33). Vì thế nên không thể phát hiện *"pair"* là từ sai được bởi thông tin cung cấp bởi *"and pair"* còn quá ít.

Trong ví dụ này, *"pear"* và *"strawberry"* đều là những ứng cử viên để thay thế *"pair"*. Tuy nhiên xác suất xuất hiện của *"strawberry"* (công thức 7.25) lớn hơn *"pear"* (công thức 7.24) và xác suất của câu *"The apple and strawberry salad"* (công thức 7.28) lớn hơn *"The apple and pear salad"* (công thức 7.27), trong khi từ đúng là *"pear"*, vậy làm sao để chọn đúng từ thay thế. Về vấn đề này, khi sửa lỗi, ngoài việc chọn từ có xác suất cao, cần phải quan tâm đến *lĩnh vực* (domain) mà chúng ta đang xử lý.

7.4.2 Sử dụng mô hình ngôn ngữ để sửa lỗi theo lĩnh vực đang xử lý

Đối với mô hình tổng quát, khi sửa lỗi thì mô hình thường sẽ gợi ý từ sửa lỗi có xác suất cao nhất, tuy nhiên đối với các lĩnh vực khác nhau thì mô hình cần phải chọn những từ có xác suất đủ cao và phải phù hợp với cách xử lý của lĩnh vực đang xét.

Đối với bài toán Speech to Text

Đối với bài toán Speech to Text, thì việc dựa trên phát âm (pronunciation) để dự đoán rất quan trọng, vậy nên việc nhận diện sai là thuộc về lỗi phát âm. Vì thế đối với gợi ý sửa lỗi phát âm thì việc đầu tiên vẫn là gợi ý những từ có xác suất xuất hiện cao và thứ hai là ta phải ưu tiên những từ có phát âm tương đối giống với từ

nghi ngờ là từ sai mà ta đang xử lý.

Một vài ví dụ khác về sự tương đồng phát âm giữa các từ:

- Accept - Except
- Base - Bass
- Sea - See
- Rice - Rise

Tuy nhiên, bởi tính phụ thuộc vào phát âm trong lĩnh vực này nên chúng ta có thể xây dựng mô hình đánh giá sự tương đồng về phát âm và chọn ra một từ có phát âm tương đồng nhất với từ sai trong tập hợp các từ được gợi ý sửa lỗi.

Một cách tiếp cận khác là chúng ta có thể chuyển các từ về cùng một dạng có thể so sánh được theo cách phát âm và đánh giá sự tương đồng.

Ví dụ:

- Pair -> pe-a
- Pear -> pe-a
- Strawberry -> stro-be-ri
- Data -> dei-ta
- Wednesday -> wenz-di

Ngoài ra chúng ta cũng có thể sử dụng Mô hình biến đổi từ (Edit Model) (phần 7.4.3) để lựa chọn từ tập gợi ý, tuy nhiên phương pháp này không mang nhiều ý nghĩa về phát âm mà chỉ là biến đổi từ.

Xét ví dụ đã nêu ở phần trước:

- Câu dự đoán của mô hình nhận diện giọng nói:

  The apple and pair salad.

- Câu đúng:

  The apple and pear salad.

Sau khi đã tính xác suất và nghi ngờ từ sai là *"pair"* và cũng xác định được các ứng cử viên là *"pear"* và *"strawberry"* thì mô hình ngôn ngữ sẽ tính độ tương đồng về phát âm giữa các ứng cử viên và từ sai rồi sẽ chọn từ có độ tương đồng cao nhất.

Ở ví dụ này chúng ta có thể thấy mô hình sẽ chọn *"pear"* để sửa lỗi vì *"pear"* (pe-a) có phát âm giống hệt *"pair"* (pe-a).

Hoặc chúng ta cũng có thể tính khoảng cách từ vựng giữa dạng phát âm của các ứng cử viên và của từ sai để tìm từ có khoảng cách phát âm ngắn nhất so với từ sai để sửa lỗi. Với cách tiếp cận này chúng ta có thể sử dụng *Khoảng cách Levenshtein* (Levenshtein distance) [**??**] để tính khoảng cách giữa các từ.

Công thức *Khoảng cách Levenshtein*:

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0, \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases} \tag{7.34}$$

Trong đó:

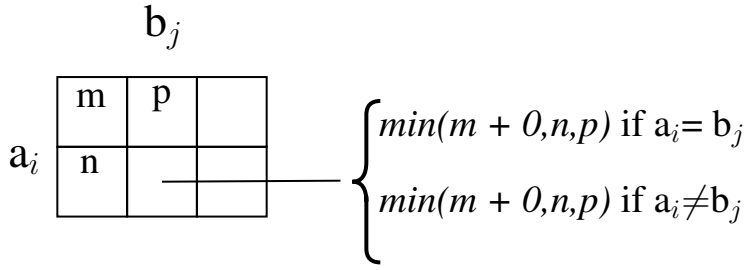
- a, b là hai từ dùng để đo khoảng cách giữa chúng.
- i, j là chỉ số ký tự đang xét đến của a và b.
- $1_{(a_i \neq b_j)}$ là một hàm đặc trưng trả về 0 nếu $a_i = b_j$ và trả về 1 trong trường hợp ngược lại.
- $lev_{a,b}(i,j)$ là khoảng cách giữa i ký tự đầu tiên của a và j ký tự đầu tiên của b.
- Khoảng cách giữa a và b là giá trị tại $lev_{a,b}(|a|,|b|)$.

Khoảng cách Levenshtein tính toán khoảng cách giữa hai từ thông qua ba phép biến đổi (từ a sang b) như sau:

- $lev_{a,b}(i-1,j) + 1$ tương ứng với phép xóa một ký tự.
- $lev_{a,b}(i,j-1) + 1$ tương ứng với phép thêm một ký tự.

- $lev_{a,b}(i-1, j-1) + 1_{(a_i \neq b_j)}$ tương ứng với phép thêm ký tự này bằng ký tự khác.

Cách tính khoảng cách Levenshtein có thể biểu diễn đơn giản như sau (Hình 7.15).



Hình 7.15: Cách tính khoảng cách Levenshtein.

Ví dụ: Khoảng cách Levenshtein giữa *"kitten"* và *"sitting"* là 3 (Bảng 7.2), vì phải dùng ít nhất ba lần biến đổi:

1. *kitten* -> *sitten* (thay *"k"* bằng *"s"*).
2. *sitten* -> *sittin* (thay *"e"* bằng *"i"*).
3. *sittin* -> *sitting* (thêm ký tự *"g"*).

Bảng 7.2: Khoảng cách giữa *"kitten"* và *"sitting"*.

		s	**i**	**t**	**t**	**i**	**n**	**g**
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
k	**1**	1	2	3	4	5	6	7
i	**2**	2	1	2	3	4	5	6
t	**3**	3	2	1	2	3	4	5
t	**4**	4	3	2	1	2	3	4
e	**5**	5	4	3	2	2	3	4
n	**6**	6	5	4	3	3	2	**3**

Áp dụng khoảng cách Levenshtein vào tính khoảng cách phát âm giữa *"pair"* (pe-a) với *"pear"* (pe-a) (Bảng 7.3) và *"strawberry"* (stro-be-ri) (Bảng 7.4).

Bảng 7.3: Khoảng cách phát âm giữa *"pear"* (pe-a) và *"pair"* (pe-a).

		p	**e**	**a**
	0	**1**	**2**	**3**
p	**1**	0	1	2
e	**2**	1	0	1
a	**3**	2	1	**0**

Bảng 7.4: Khoảng cách phát âm giữa *"strawberry"* (stro-be-ri) và *"pair"* (pe-a).

		p	**e**	**a**
	0	**1**	**2**	**3**
s	**1**	1	2	3
t	**2**	2	2	3
r	**3**	3	3	3
o	**4**	4	4	4
b	**5**	5	5	5
e	**6**	6	5	6
r	**7**	7	6	6
i	**8**	8	7	**7**

Ta có thể thấy khoảng cách phát âm giữa *"pear"* (pe-a) và *"pair"* (pe-a) là 0 và giữa *"strawberry"* (stro-be-ri) và *"pair"* (pe-a) là 7, vì thế mô hình sẽ chọn *"pear"* là từ thay thế.

7.4.3 Đối với bài toán Image to Text

Đối với bài toán Image to Text thì ta phải ưu tiên những từ có đặc điểm, nét viết gần giống với từ sai. Và tất nhiên là những từ đó vẫn phải có xác suất xuất hiện cao.

Ví dụ:

- Nhầm lẫn giữa *"1"* và *"4"* (Hình 7.16).

((a)) 1 ((b)) 4

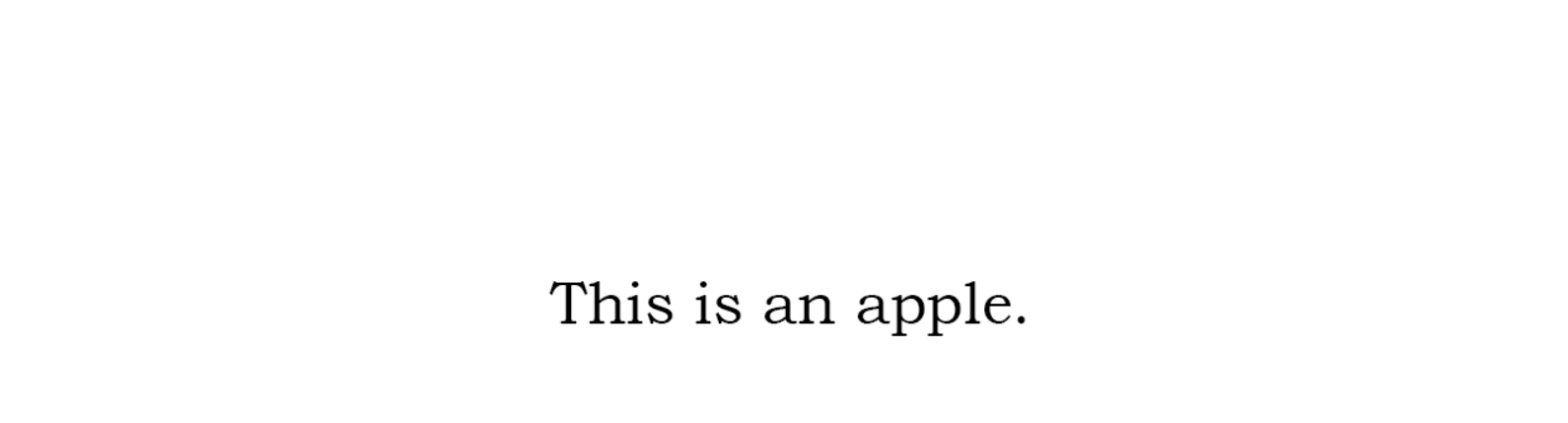
Hình 7.16: Số 1 và số 4 viết tay có nét tương đồng nhau.

- Nhầm lẫn giữa *"9"*, *"g"* và *"q"* (Hình 7.17).

Hình 7.17: Số 9, ký tự g và ký tự q viết tay có nét tương đồng nhau.

Về lĩnh vực này thì chúng ta cũng có thể xây dựng mô hình đánh giá sự tương đồng về nét, hình ảnh của các từ, ký tự. Hoặc chúng ta có thể tạo ra danh sách các ký tự có nét giống nhau và dùng những tập hợp đó để đánh giá và chọn từ sửa lỗi. Ngoài ra còn có thể sử dụng mô hình biến đổi từ (Edit Model) (phần 7.4.3) hoặc tính *Khoảng cách Levenshtein* (công thức 7.34) để chọn từ thay thế, tuy nhiên những cách này không mang nhiều ý nghĩa về sự tương đồng hình ảnh của các ký tự.

Xét ví dụ sau: Nhận diện text trên hình khi sử dụng font chữ dễ gây nhầm lẫn (Bookman Old Style) (Hình 7.18).



Hình 7.18: Font chữ Bookman Old Style gây nhầm lẫn ký tự *"l"* và số *"1"*.

- Mô hình dự đoán:

  This is an app1e.

- Câu đúng:

  This is an apple.

Xác suất khi tính đến từ *"app1e"* sẽ rất thấp hoặc có thể bằng 0, vậy nên mô hình sẽ nghi ngờ đó là từ sai và sẽ tìm từ thay thế.

Mô hình có thể sử dụng các tập hợp các ký tự tương tự nhau để thay vào từ sai và tạo ra tập hợp các ứng cử viên từ đúng và tính xác suất.

Ví dụ:

- Tập hợp các ký tự giống số *"1"*: {1, l, I, i, t, /, |}
- Tập hợp các ứng cử viên sau khi thay thế số *"1"* vào từ sai *"app1e"*: {apple, appIe, appie, appte, app/e, app|e}

Mô hình sẽ tính được xác suất của từ *"apple"* là cao nhất nên sẽ chọn *"apple"* là từ dùng để sửa lỗi.

Đối với bài toán sửa lỗi chính tả (typo/spelling)

Đối với bài toán chỉnh sửa lỗi chính tả (typo/spelling), thì ta phải ưu tiên những từ có khoảng cách chính tả gần với từ sai.

Đối với lĩnh vực này, chúng ta có thể xây dựng một số phương pháp như sau:

- Dựa trên thống kê lỗi trên dữ liệu văn bản thu thập được để xây dựng mô hình xác xuất. Phương pháp này rất khó thực hiện, vì tần suất lỗi chính tả trong

dữ liệu thu thập thường rất thấp. Để có mô hình tốt, cần phải có dữ liệu rất lớn.

- Định nghĩa các loại lỗi thường xảy ra và xây dựng mô hình đánh giá khả năng xảy ra lỗi (Edit Model). Đây là phương pháp đơn giản và cũng thường được dùng nhất.

Với phương pháp xây dựng mô hình đánh giá khả năng xảy ra lỗi của từ, chúng ta có thể xét tới các loại lỗi chính tả khi nhập từ bàn phím như sau (ví dụ với lỗi xảy ra đối với từ *"example"*):

- Lỗi nhập sai ký tự trong từ bằng ký tự khác (replace): e**x**ample -> e**z**ample
- Lỗi nhập sai thứ tự các ký tự trong từ (transpose): examp**le** -> exmap**el**
- Lỗi nhập thiếu ký tự trong từ (delete): examp**l**e -> exampe
- Lỗi nhập thừa ký tự (insert): example -> exa**a**mple

Đối với mô hình sửa lỗi này, khi đã phát hiện từ sai từ mô hình ngôn ngữ, ta sẽ áp dụng cả bốn phép biến đổi sau vào từ sai như sau:

- Replace: Thay thế mỗi ký tự trong bảng chữ cái vào từng vị trí của từ sai.
- Transpose: Đảo từng cặp ký tự liền kề nhau trong từ.
- Delete: Xóa từng ký tự trong từ.
- Insert: Thêm từng ký tự trong bảng chữ cái vào giữa từng vị trí của từ.

Xét ví dụ:

- Người dùng nhập vào:

I need an *exampe* for this problem.

- Câu đúng:

I need an *example* for this problem.

Sau khi tính xác suất đến từ *"exampe"* của câu sai, mô hình sẽ tính được xác suất đó là rất thấp hoặc có thể bằng 0 và sẽ nghi ngờ *"exampe"* là từ sai. Sau đó áp dụng mô hình sửa lỗi từ vào từ *"exampe"* để tìm tập các ứng cử viên thay thế như sau:

- Replace -> **a**xampe, **b**xampe, ..., e**a**ampe, e**b**ampe, ..., examp**y**, examp**z**
- Transpose -> **xe**ampe, e**ax**mpe, ex**ma**pe, exa**pm**e, exam**ep**
- Delete -> xampe (**_xampe**), eampe (**e_ampe**), exmpe (**ex_mpe**), exape (**exa_pe**), exame (**exam_e**), examp (**examp_**)
- Insert -> **a**exampe, **b**exampe, ..., e**a**xampe, e**b**xampe, ..., exampl**e**, examp**m**e, ..., exampe**y**, exampe**z**

Sau khi biến đổi như trên, ta sẽ có một tập hợp tất cả các từ đã được biến đổi, từ được chọn để sửa sai sẽ là một từ trong tập hợp trên và có xác suất xuất hiện cao nhất trong bộ dữ liệu. Trong ví dụ trên, từ *"example"* sẽ là từ có xác suất xuất hiện cao nhất nên mô hình sẽ lựa chọn *"example"* là từ được dùng để sửa lỗi.

Phương pháp biến đổi nêu trên để áp dụng cho trường hợp một ký tự lỗi, ta có thể áp dụng tương tự cho các trường hợp nhiều ký tự lỗi, tuy nhiên số lượng từ để xét tới sẽ trở nên lớn, tốc độ xử lý khi sửa lỗi chậm và tập hợp ứng cử viên sẽ lớn nên xác suất nhầm lẫn giữa các từ gần giống nhau về mặt ký tự sẽ lớn.

Ngoài ra, chúng ta có thể tìm tập ứng cử viên trong bộ dữ liệu và sử dụng *Khoảng cách Levenshtein* (công thức 7.34) để tính khoảng cách từ vựng với từ sai và tìm từ thay thế.

7.5 Phụ lục

Dưới đây là một số thuật ngữ có đề cập trong chương 7:

Bảng 7.5: Bảng các thuật ngữ

Thuật ngữ	Ý nghĩa
Language Model	mô hình ngôn ngữ
likelihood	xác suất xuất hiện
Machine Translation	dịch máy
Spelling Correction	sửa lỗi chính tả
Speech Recognition	nhận biết giọng nói
long-distance dependency	sự phụ thuộc khoảng cách dài
vocabulary	từ điển
smoothing	kỹ thuật làm trơn
extrinsic evaluation	đánh giá ngoài
intrinsic evaluation	đánh giá trong
sample	mẫu
branching factor	hệ số phân nhánh
weighted branching factor	hệ số phân nhánh có trọng số
Maximum likelihood estimation (MLE)	ước lượng hợp lý cực đại
Laplace smoothing	kỹ thuật làm trơn Laplace
Linear Interpolation	nội suy tuyến tính
uniform distribution	phân phối đều
cumulative distribution function	hàm phân phối tích lũy
threshold	ngưỡng
Edit Model	mô hình biến đổi từ

Phụ lục B

Danh sách lớp CH_KH192 môn Hệ thống thông minh tham gia soạn các chương sách.

1. Chương 1:
    - Nguyễn Đức Hiệp
    - Hồ Phương Ngọc
2. Chương 2:
    - Trương Đức Tuấn
    - Trần Đức Thắng
3. Chương 3:
    - Cáp Đặng Xuân Kiệt
    - Dương Đức Tín
4. Chương 4:
    - Nguyễn Danh Khôi
    - Văn Quân
    - Lê Triều Dương
    - Nguyễn Thái Hoàng
    - Nguyễn Sư Phước
5. Chương 5:
    - Trần Quang Minh
    - Trần Công Hậu
    - Nguyễn Thiện Nhân
    - Tô Thành Nhân

6. Chương 6:
    - Phạm Phương Uyên
    - Trương Nguyễn Duy Khang
    - Vũ Hoàng Văn
    - Từ Lãng Phiêu
7. Chương 7:
    - Nguyễn Xuân Quang
    - Trần Trương Tuấn Phát
    - Nguyễn Trọng Tính
    - Trần Hoàng Tuấn
    - Nguyễn Phạm Anh Nguyên

Phụ lục C

Vấn đề vanishing gradient khi huấn luyện mạng RNN

Xét mạng RNN với cách thức cập nhật trạng thái ở mỗi state được cho bởi công thức dưới đây:

$$\begin{aligned} a_t &= \mathbf{F}(x_t, a_{t-1}, \theta) \\ &= g(W_{aa} a_{t-1} + W_{ax} x_t + b_a) \end{aligned} \tag{35}$$

g: Hàm truyền (tanh, sigmoid)
$W_{aa} \in R^{n \times n}$: Ma trận trọng số để chuyển đổi từ trạng thái a_{t-1} qua trạng thái a_t
$W_{ax} \in R^{n \times D}$: Ma trận trọng số để chuyển đổi từ không gian đầu vào $x \in R^D$ tới không gian trạng thái $a_t \in R^n$

$$\hat{y}_t = f(W_{ya} a_t + b_y) \tag{36}$$

$$L = \sum_{t=1}^{T} L_t(\hat{y}_t) \tag{37}$$

$$\frac{\partial L}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \theta} \tag{38}$$

Hàm mất mát L được định nghĩa bằng tổng tổng của tất cả các hàm mất mát con tại thời điểm t L_t. Dựa vào công thức số (35), đạo hàm của L đối với bộ tham số $\theta(W_{aa}, W_{ax}, ba)$ được tính băng chain-rule như sau:

$$\frac{\partial L_t}{\partial \theta} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial a_t} \frac{\partial a_t}{\partial a_k} \frac{\partial_k^a}{\partial \theta} \tag{39}$$

$\frac{\partial_k^a}{\partial \theta}$ biểu diễn đạo hàm riêng "tức thời" của a_k đối với θ, nghĩa là trong cách tính $\frac{\partial_k^a}{\partial \theta}$, giá trị a_{k-1} được xem là hằng số đối với θ

$$\frac{\partial a_t}{\partial a_k} = \prod_{i=k+1}^{t} \frac{\partial a_i}{\partial a_{i-1}} = \prod_{i=k+1}^{t} diag(g'(a_i)) W_{aa} \tag{40}$$

Trong công thức trên $\|diag((g'(a_i)))\| \leq \gamma$ là ma trận đường chéo, trong đó mỗi thành phần trên đường chéo chính tương ứng với giá trị đạo hàm của $g'(a_i$ đối với a_i. Vì đạo hàm của hàm truyền g bị chặn trên bởi giá trị $\gamma \in R$ ($\gamma = 1$ đối với *tanh*, $\gamma = 1/4$ đối với *sigmoid*), nên $\|diag((g'(a_i)))\| \leq \gamma$. Gọi σ_1 là singular value lớn nhất của W_{aa} (tương đương với σ_1^2 là trị riêng lớn nhất của W_{aa} hoặc $W_a a_{aa}^W$), dựa vào định nghĩa và các tính chất của norm đối với ma trận (xem phụ lục phía dưới), dễ dàng chứng minh:

$$\begin{aligned} \left\| \frac{\partial a_i}{\partial a_{i-1}} \right\| &\leq \|diag(g'(a_i))\| \, \|W_{aa}\| \\ &\leq \gamma \sigma_1 \end{aligned} \tag{41}$$

$$\Rightarrow \left\| \frac{\partial a_t}{\partial a_k} \right\| = \left\| \prod_{i=k+1}^{t} \frac{\partial a_i}{\partial a_{i-1}} \right\| \leq (\gamma \sigma_1)^{t-k} \tag{42}$$

Thay (42) vào (39) (40), sử dụng định nghĩa norm của ma trận vào từng cột của ma trận Jacobian $\frac{\partial_k^a}{\partial \theta}$(tương đương với đạo hàm "tức thời" của a_k với từng chiều θ_j trong bộ tham số θ), ta có:

$$\begin{aligned} \left\| \frac{\partial L_t}{\partial \theta} \right| &= \left\| \sum_{k=1}^{t} \frac{\partial L_t}{\partial a_t} \frac{\partial a_t}{\partial a_k} \frac{\partial_k^a}{\partial \theta} \right\| \\ &\leq \sum_{k=1}^{t} \left[\left\| \frac{\partial L_t}{\partial a_t} \right\| \left\| \frac{\partial a_t}{\partial a_k} \right\| \right] \\ &\leq \sum_{k=1}^{t} \left[\left\| \frac{\partial L_t}{\partial a_t} \right\| (\gamma \sigma_1)^{t-k} \right] \end{aligned} \tag{43}$$

Khi $\sigma_1 < \frac{1}{\gamma}$, (i.e $(\gamma \sigma_1)^{t-k} = \eta^{t-k}$ với $\eta < 1$), theo công thức (43), sự đóng góp của những input và trạng thái ở xa so với trạng thái t hiện hành trong một chuỗi quá dài ($t - k$ lớn) sẽ về 0 với tốc độ hội tụ theo hàm mũ, đây chính là hiện tượng vanishing gradient trong RNN.

Matrix Norm

Định nghĩa: Norm của ma trận $A \in R^{m \times n}$ được định nghĩa như sau:

$$\|A\| = \sup \left\{ \frac{\|Ax\|}{\|x\|} \quad : x \in R^n \quad , \quad x \neq 0 \right\} \tag{44}$$

Định lý

$$\|A\| = \sigma_{max} = \sigma_1$$

Where σ_1 là singular value lớn nhất của A

Chứng minh. Định lý 7.5 có thể chứng minh trực tiếp bằng khai triển Singular Value Decomposition.

$$\begin{aligned}
\|A\| &=_{x \neq 0} \frac{\|Ax\|}{\|x\|} =_{x \neq 0} \frac{\|U \Sigma V^x\|}{\|x\|} \\
&=_{x \neq 0} \frac{\|\Sigma V^x\|}{\|x\|} \\
&=_{y \neq 0} \frac{\|\Sigma y\|^2}{\|Vy\|^2} \\
&=_{y \neq 0} \frac{\left(\sum_{=1}^{r} \sigma_i^2 |y_i|\right)^{1/2}}{\left(\sum_{=1}^{r} |y_i|\right)^{1/2}} \\
&\leq \sigma_1
\end{aligned}$$

Dấu bằng xảy ra khi $y = \begin{bmatrix} 1 & 0 & ... & 0 \end{bmatrix}$, $\|\Sigma y\| = \sigma_1$. Lúc đó $x = v_1$, vector đầu tiên trong tập các vector tạo nên cơ sở trực chuẩn V □

Bibliography

Bengio, Y., Ducharme, R., & Vincent, P. (2001). A neural probabilistic language model. In T. Leen, T. Dietterich, & V. Tresp (Eds.), *Advances in neural information processing systems* (pp. 932–938). MIT Press. https://proceedings.neurips.cc/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf

Bertsekas, D. P., Nedic, A., & Ozdaglar, A. E. (2003). Convex analysis and optimization athena scientific. *Journal of Mathematical Analysis & Applications*, *129*(2), 420–432.

Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM*, *55*(4), 77–84.

Bradeško, L., & Mladenić, D. (2012). A survey of chatbot systems through a loebner prize competition. *Proceedings of Slovenian language technologies society eighth conference of language technologies*, 34–37.

Charnes, A., Frome, E. L., & Yu, P.-L. (1976). The equivalence of generalized least squares and maximum likelihood estimates in the exponential family. *Journal of the American Statistical Association*, *71*(353), 169–171.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

Freedman, D. A. (2009). *Statistical models: Theory and practice*. cambridge university press.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*, 193–202.

Guida, G., & Mauri, G. (1986). Evaluation of natural language processing systems: Issues and approaches. *Proceedings of the IEEE*, *74*(7), 1026–1035.

Han, J., & Moraga, C. (1995). The influence of the sigmoid function parameters on the speed of backpropagation learning. *International Workshop on Artificial Neural Networks*, 195–201.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del R'ıo, J. F., Wiebe, M., Peterson, P., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Harris, Z. S. (1954). Distributional structure. *Word*, *10*(2-3), 146–162.

Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory*. Wiley.

Hinton, G. E., & Zemel, R. (1994). Autoencoders, minimum description length and helmholtz free energy. In J. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in neural information processing systems* (pp. 3–10). Morgan-Kaufmann. https://proceedings.neurips.cc/paper/1993/file/9e3cfc48eccf81a0d57663e129aef3cb-Paper.pdf

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, *79*(8), 2554–2558. http://view.ncbi.nlm.nih.gov/pubmed/6953413]

Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, *148*, 574–591.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1746–1751. https://doi.org/10.3115/v1/D14-1181

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551.

Lemaréchal, C. (2012). Cauchy and the gradient method. *Doc Math Extra*, *251*(254), 10.

Leskovec, J., Rajaraman, A., & Ullman, J. D. (2014). *Mining of massive datasets* (Second). Cambridge University Press. http://mmds.org

Madsen, M. W. (2009). The limits of machine translation. *Center for Language Technology, Univ. of Copenhagen, Copenhagen.*

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press. http://nlp.stanford.edu/IR-book/information-retrieval-book.html

Marsh, E., & Perzanowski, D. (1998). Muc-7 evaluation of ie technology: Overview of results. *Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29-May 1, 1998*.

Maybury, M. (1999). *Advances in automatic text summarization*. MIT press.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*, 115–133.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space.

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. MIT press.

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)*, 1532–1543. https://doi.org/10.3115/v1/D14-1162

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519

Rosenblatt, F. (1959). *Principles of neurodynamics*. Spartan Books.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart & J. L. Mcclelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundations* (pp. 318–362). MIT Press.

Russel, S., Norvig, P. et al. (2013). *Artificial intelligence: A modern approach*. Pearson Education Limited London.

Russell, S., & Norvig, P. (2002). Artificial intelligence: A modern approach.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, *61*, 85–117.

Soergel, D. (1985). *Organizing information: Principles of data base and retrieval systems*. Elsevier.

Stewart, J., Clegg, D. K., & Watson, S. (2020). *Calculus: Early transcendentals*. Cengage Learning.

Stone, P. J., Dunphy, D. C., & Smith, M. S. (1966). The general inquirer: A computer approach to content analysis.

Tolles, J., & Meurer, W. J. (2016). Logistic regression: Relating patient characteristics to outcomes. *Jama*, *316*(5), 533–534.

Voleti, R. (2021). Unfolding the evolution of machine learning and its expediency.

Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. *1960 IRE WESCON Convention Record, Part 4*, 96–104.

Wołk, K., & Marasek, K. (2014). A sentence meaning based alignment method for parallel text corpora preparation. *New perspectives in information systems and technologies, volume 1* (pp. 229–237). Springer.