

**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 - 2021
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 kỷ 20, kỹ thuật *học sâu* (Deep Learning) đã tìm được cho mình một nguồn năng lượng mạnh mẽ 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* (Artificial Neural Network - ANN). 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 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* (Artificial Intelligence - AI) 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* (Computer Science). 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 gop 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 Network - 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 Model).

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 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. Tác giả xin chân thành gửi lời cảm ơn đến các đồng nghiệp trong Khoa và Ban Chủ nhiệm Khoa. Đặc biệt, tác giả xin cảm ơn các bạn Nguyễn Trần Công Duy, Nguyễn Minh Đăng, Nguyễn Quang Đức, Lê Minh Khôi, Bùi Ngô Hoàng Long, Nguyễn Xuân Mão, Băng Ngọc Bảo Tâm, Nguyễn Gia Thịnh, Nguyễn Thành Thông, Nguyễn Võ Thùy Trang, Mai Đức Trung và Bùi Công Tuấn đã đóng góp nhiều công sức và ý kiến quý báu trong quá trình hoàn thiện nội dung sách. Lời cảm ơn cũng xin được gửi đến các học viên cao học của các lớp Hệ Thống Thông Minh CK_HK192, Cách Tiếp Cận Hiện Đại Trong Xử Lý Ngôn Ngữ Tự Nhiên CK_HK201 và CK_HK202 của Trường Đại học Bách Khoa - ĐHQG-HCM đã sử dụng và góp ý cho các phiên bản thử nghiệm đầu tiên của 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, các ví dụ và cuối cùng là kết chương. Sau mỗi chương sẽ có phần bài tập luyện tập thêm, làm rõ các nội dung trình bày trong chương.

Chương 1: Giới thiệu 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. Nội dung chương này sẽ bao gồm lịch sử mạng nơ-ron, giới thiệu các 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: Hồi quy Logistic. 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 lỗi matsu, kỹ thuật gradient descent, đạo hàm và lan truyền ngược (back propagation).

Chương 3: Mạng đa tầng và học sâu. 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. Điều 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. 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. 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: Mô hình ngôn ngữ. 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

Chương 1: Giới thiệu mạng nơ-ron nhân tạo	1
1.1 Lịch sử mạng nơ-ron	1
1.2 Các thành phần của mạng nơ-ron	3
1.2.1 Cấu tạo và hoạt động của nơ-ron sinh học	3
1.2.2 Cấu tạo và hoạt động của perceptron	4
1.3 Mạng nơ-ron đa tầng	5
1.3.1 Thế nào là mạng nơ-ron đa tầng?	6
1.3.2 Các thành phần của một mạng nơ-ron đa tầng	6
1.4 Sử dụng mạng nơ-ron cho bài toán phân loại	11
1.5 Kết chương	15
1.6 Bài tập	15
Chương 2: Hồi quy Logistic (Logistic Regression)	17
2.1 Giới thiệu về hồi quy logistic	17
2.2 Bài toán học có giám sát và hàm kích hoạt sigmoid	18
2.3 Hàm chi phí của hồi quy logistic	22
2.4 Sử dụng gradient descent để tìm giá trị tối ưu	26
2.5 Cách tính đạo hàm riêng dưới góc nhìn đồ thị tính toán	32
2.6 Tổng kết phương pháp hồi quy logistic	39
2.7 Kết chương	40
2.8 Bài tập	41
Chương 3: Mạng đa tầng và học sâu (Multilayer perceptron - MLP và Deep learning - DL)	43
3.1 Hồi quy logistic dưới góc nhìn mạng nơ-ron	43
3.2 Mạng nơ-ron đa tầng và mạng học sâu	44

3.2.1	Nhắc lại mạng nơ-ron đa tầng	44
3.2.2	Sự tiến hóa của mạng nơ-ron đa tầng	45
3.2.3	Mạng học sâu	45
3.3	Các hàm kích hoạt (activation function)	46
3.3.1	Hàm sigmoid	46
3.3.2	Hàm tanh	48
3.3.3	Hàm ReLU	49
3.3.4	Hàm leaky ReLU	50
3.4	Biểu diễn mạng nơ-ron như là vector và ma trận	51
3.4.1	Biểu diễn mạng nơ-ron	51
3.4.2	Cách biểu diễn lan truyền xuôi và lan truyền ngược	54
3.4.3	Vì sao lại biểu diễn bằng ma trận?	57
3.4.4	Kỹ nguyên mới của mạng nơ-ron	58
3.5	Kết chương	59
3.6	Bài tập	60
Chương 4:	Mô hình xử lý ngôn ngữ tự nhiên bằng mạng học sâu	63
4.1	Giới thiệu về xử lý ngôn ngữ tự nhiên	63
4.1.1	Mô hình xử lý ngôn ngữ tự nhiên cổ điển	65
4.1.2	Tiền xử lý	66
4.1.3	Mô hình hóa	67
4.2	Trích xuất đặc trưng (Feature extraction)	68
4.2.1	Phương pháp túi từ	68
4.2.2	Phương pháp TF-IDF	70
4.3	Mô hình xử lý ngôn ngữ tự nhiên bằng mạng học sâu	74
4.4	Các kỹ thuật biểu diễn từ	76
4.4.1	Biểu diễn từ bằng one-hot vector	77
4.4.2	Kỹ thuật Auto-Encoder để thu giảm số chiều	80
4.4.3	Kỹ thuật word2vec	85
4.5	Bài tập	100
Chương 5:	Mạng nơ-ron tích chập	103
5.1	Giới thiệu mạng nơ-ron tích chập	103
5.1.1	Lịch sử mạng nơ-ron tích chập	103
5.1.2	Đặc tính cơ bản của mạng nơ-ron tích chập	104

5.2	Cách thức hoạt động của mạng nơ-ron tích chập	107
5.2.1	Phép tích chập (Convolution)	107
5.2.2	Phép gộp (Pooling)	111
5.2.3	Ý nghĩa của ma trận kết quả	112
5.2.4	Bản đồ thuộc tính (Feature map)	114
5.2.5	Cách áp dụng bộ lọc	115
5.3	Mạng nơ-ron tích chập dưới góc nhìn của một mạng nơ-ron nhân tạo	116
5.4	Triển khai một mạng CNN	118
5.5	Sử dụng mạng CNN cho các bài toán NLP	121
5.5.1	Các cách tiếp cận cổ điển trước khi sử dụng CNN	121
5.5.2	Sử dụng CNN vào bài toán NLP với phương pháp Word Embedding	121
5.5.3	Case study: sử dụng CNN cho bài toán phân tích cảm xúc (sentiment analysis)	123
5.6	Bài tập	126
Chương 6:	Mạng nơ-ron truy hồi (Recurrent Neural Network - RNN)	131
6.1	Giới thiệu về dữ liệu chuỗi	131
6.1.1	Dữ liệu chuỗi là gì	131
6.1.2	Một số vấn đề của mạng nơ-ron thông thường với dữ liệu chuỗi	135
6.2	Giới thiệu về RNN	137
6.3	Cơ chế lan truyền xuôi của mạng nơ-ron truy hồi	140
6.4	Cơ chế lan truyền ngược của RNN	144
6.5	Ưu và nhược điểm của RNN	149
6.6	Ví dụ minh họa mạng RNN	150
6.7	Bài tập	154
Chương 7:	Mô hình ngôn ngữ (Language model)	157
7.1	Mô hình hoá ngôn ngữ	157
7.1.1	Giới thiệu	157
7.1.2	Ước lượng xác suất và mô hình n -gram	160
7.1.3	Đánh giá mô hình ngôn ngữ	164
7.1.4	Làm trơn mô hình ngôn ngữ	167

7.2	Mô hình ngôn ngữ với RNN	170
7.2.1	Ý tưởng cơ bản	170
7.2.2	Huấn luyện mô hình ngôn ngữ với RNN	172
7.3	Bài toán tạo sinh chuỗi	174
7.3.1	Tạo mẫu từ phân bố xác suất bất kỳ	174
7.3.2	Sinh chuỗi từ mạng RNN đã được huấn luyện	177
7.4	Sử dụng mô hình ngôn ngữ để sửa lỗi kết quả của các mô hình khác .	185
7.4.1	Cơ chế sửa lỗi của mô hình ngôn ngữ	185
7.4.2	Sử dụng mô hình ngôn ngữ để sửa lỗi theo lĩnh vực đang xử lý	189
7.4.3	Đối với bài toán Image to Text	193

Danh sách hình vẽ

1.1	Cấu tạo của một nơ-ron sinh học	4
1.2	Cấu tạo và cách thức hoạt động của perceptron	5
1.3	Kiến trúc mạng nơ-ron đa tầng với 2 tầng ẩn	6
1.4	Mạng nơ-ron biểu diễn Ví dụ 1.3.1	7
1.5	Miền giá trị cần tìm trong Ví dụ 1.3.1	9
1.6	Mạng nơ-ron biểu diễn Ví dụ 1.3.2	9
1.7	Bảng sự thật biểu diễn phép XOR trong Ví dụ 1.3.2	11
1.8	Mạng nơ-ron biểu diễn Ví dụ 1.4.1	12
1.9	Miền giá trị thỏa mãn yêu cầu Ví dụ 1.4.1	13
1.10	Miền giá trị thỏa mãn yêu cầu Ví dụ 1.4.1 sau khi ràng buộc về khoảng giá trị của điểm bài tập lớn và điểm thi cuối kỳ	13
1.11	Đồ thị biểu diễn các miền giá trị tuyến tính (a) và phi tuyến (b, c, d)	14
1.12	Đồ thị biểu diễn gần đúng miền giá trị là hình tròn	14
1.13	Mạng nơ-ron trong Bài tập 1.6.1	15
2.1	Đồ thị biểu diễn các điểm dữ liệu trong Bảng 2.1	19
2.2	Kết quả khi dùng hồi quy tuyến tính	20
2.3	Đồ thị hàm sigmoid.	21
2.4	Kết quả khi dùng hồi quy logistic.	22
2.5	Đồ thị hàm lồi (a) và đồ thị hàm không lồi (b)	23
2.6	Đồ thị minh họa kỹ thuật gradient descent	27
2.7	Sử dụng gradient descent để tìm giá trị tối ưu	28
2.8	Sử dụng gradient descent để tìm giá trị tối ưu ứng với các giá trị α .	29
2.9	Sử dụng gradient descent với hàm số không là hàm lồi	30
2.10	Kết quả khi áp dụng gradient descent thực tế	31

2.11	Sử dụng đồ thị tính toán để biểu diễn hàm số	32
2.12	Đồ thị tính toán hàm matsu mát của hồi quy logistic	36
3.1	Một mạng nơ-ron đơn giản	43
3.2	Kiến trúc mạng học sâu với 3 tầng ẩn	46
3.3	Đồ thị của hàm sigmoid	46
3.4	Đồ thị biểu diễn giá trị và đạo hàm của hàm sigmoid	47
3.5	Hoạt động của một tầng ẩn trong mạng MLP dùng hàm sigmoid	48
3.6	Đồ thị của hàm tanh và đồ thị đạo hàm của hàm tanh	49
3.7	Đồ thị biểu diễn hàm ReLU	50
3.8	Đồ thị biểu diễn hàm Leaky ReLU	51
3.9	Kiến trúc mạng nơ-ron 3 tầng	52
3.10	Kiến trúc của 2 tầng liên tiếp nhau trong một mạng nơ-ron đa tầng .	53
3.11	Quá trình lan truyền xuôi.	54
3.12	Quá trình lan truyền ngược giữa 2 tầng trong mạng nơ-ron	55
3.13	Kiến trúc mạng nơ-ron của Bài tập 3.6.1	60
3.14	Kiến trúc mạng nơ-ron của Bài tập 3.6.2	60
3.15	Kiến trúc mạng nơ-ron của Bài tập 3.6.3	61
4.1	Một pipeline phổ biến cho bài toán xử lý ngôn ngữ tự nhiên theo hướng cổ điển	65
4.2	Một luồng tiền xử lý dữ liệu phổ biến	66
4.3	Mô hình học sâu được sử dụng cho bài toán NLP	75
4.4	Cách biểu diễn HCMUT bằng one-hot vector.	77
4.5	Mô hình Auto-Encoder sử dụng mạng nơ-ron 3 lớp	81
4.6	Sơ đồ huấn luyện mô hình skip-gram khi kích thước cửa sổ là 1 . . .	88
4.7	Sơ đồ huấn luyện mô hình skip-gram khi kích thước cửa sổ là 2 . . .	88
4.8	Sơ đồ huấn luyện mô hình skip-gram khi kích thước cửa sổ là 2 và sử dụng từ mục tiêu trong tài liệu D2	88
4.9	Mô hình skip-gram dạng tổng quát	89
4.10	Sơ đồ minh họa mô hình CBOW	93
4.11	Sơ đồ minh họa mô hình CBOW áp dụng cho Ví dụ trên	93
4.12	Mô hình CBOW dạng tổng quát	94

5.1	Phương thức phân tích dữ liệu và dự đoán giá sản phẩm điện thoại thông minh	105
5.2	Quá trình học và trích xuất thuộc tính của một mạng nơ-ron dự đoán giá sản phẩm điện thoại thông minh.	106
5.3	Ma trận hình ảnh (trái) và ma trận cửa sổ tích chập (phải)	107
5.4	Quá trình nhân tích chập cho phần tử đầu tiên	108
5.5	Ma trận kết quả đầy đủ sau khi nhân tích chập với cửa sổ trượt đầu tiên	109
5.6	3 ma trận cửa sổ trượt tiếp theo	110
5.7	3 ma trận kết quả sau khi nhân tích chập từ 3 cửa sổ trượt trên	110
5.8	Quá trình thực hiện phép gộp max pooling	111
5.9	4 ma trận kết quả sau khi thực hiện max pooling.	112
5.10	4 ma trận kết quả mới sau khi lượt giảm các giá trị nhỏ hơn 5	113
5.11	Bản đồ thuộc tính được tạo thành từ việc ghép các ma trận kết quả .	114
5.12	Trích xuất đặc trưng của hình ảnh theo chiều dọc (b) và chiều ngang (c)	115
5.13	Quá trình nhân tích chập một bộ lọc tạo ra các nơ-ron	116
5.14	Kiến trúc mạng CNN	119
5.15	Kiến trúc mạng LeNet-5	120
5.16	Biểu diễn mỗi từ trong câu thành vector bằng Word Embedding	121
5.17	Sử dụng thêm tầng CNN trong cách tiếp cận mới	122
5.18	Sơ đồ biểu diễn kiến trúc CNN cho bài toán phân loại cảm xúc	123
5.19	Ma trận hình ảnh và cửa sổ tích chập trong bài tập 5.6.1	127
5.20	Ma trận hình ảnh và 2 cửa sổ tích chập ở tầng đầu tiên trong Bài tập 5.6.2	128
5.21	Cửa sổ tích chập cho tầng thứ 2 trong Bài tập 5.6.2	128
6.1	Sinh nhạc tự động	133
6.2	Phân loại từ trong một câu	133
6.3	Bài toán nhận diện giọng nói	134
6.4	Bài toán dịch máy	134
6.5	Bài toán phân tích cảm xúc	135
6.6	Áp dụng mạng đa tầng cho bài toán dữ liệu chuỗi theo dạng mô hình regressive	136

6.7	Kiến trúc của một lớp mạng nơ-ron truy hồi được biểu diễn với đường nối vòng (trái) và biểu diễn với hidden state (phải)	138
6.8	Luồng tính toán cụ thể trên sequence của RNN	139
6.9	Các mẫu thiết kế mạng nơ-ron truy hồi cho từng loại bài toán	140
6.10	Đồ thị tính toán cụ thể của một nút trong mạng nơ-ron hồi quy	142
6.11	RNN với hàm mất mát	144
6.12	RNN với hàm mất mát và trọng số cụ thể	146
6.13	Sơ đồ tính toán RNN rút gọn	150
6.14	Sơ đồ tính toán của bài toán dự đoán chữ cái dạng đầy đủ	152
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	160
7.2	Bigram Language Model	161
7.3	n -gram Language Model: trigram và 4-gram	163
7.4	Thống kê count của bigram chưa được làm tròn	168
7.5	Thống kê count của bigram đã được làm tròn bằng ước lượng add-1	169
7.6	Mô hình mạng one-to-many RNN	170
7.7	Mô hình đơn giản thể hiện ý tưởng huấn luyện Language Model với RNN	171
7.8	Minh họa Ví dụ 7.2.2, huấn luyện Language Model với RNN	173
7.9	Hàm phân phối tích lũy của các từ	176
7.10	Cấu trúc mạng RNN	177
7.11	Cấu trúc mạng RNN sinh chuỗi với các từ được gợi ý	178
7.12	Harry Potter được viết bởi <i>GPT-2</i>	181
7.13	Một bài báo được viết bởi <i>GPT-2</i>	182
7.14	Mạng RNN viết báo tiếng Việt	183
7.15	Mạng RNN sáng tác truyện Kiều	184
7.16	Cách tính khoảng cách Levenshtein.	192
7.17	Số 1 và số 4 viết tay có nét tương đồng nhau.	194
7.18	Số 9, ký tự g và ký tự q viết tay có nét tương đồng nhau.	194
7.19	Font chữ Bookman Old Style gây nhầm lẫn ký tự "l" và số "1".	195

Danh sách bảng

2.1	Dữ liệu đầu vào của Ví dụ 2.2.1	18
2.2	Dữ liệu đầu vào của Bài tập 2.8.1	41
4.1	Bảng ma trận vector thu được từ phương pháp BOW	70
4.2	Bảng từ vựng trong kho ngữ liệu của Ví dụ 4.2.2	73
4.3	Bảng tính TF cho các tài liệu	73
4.4	Bảng tính IDF cho các văn bản	73
4.5	Bảng tính TF-IDF	74
4.6	Sử dụng one-hot vector để biểu diễn một số trường đại học ở Việt Nam	78
4.7	Sử dụng one-hot vector để biểu diễn các từ trong từ điển	79
4.8	Từ mục tiêu và từ ngữ cảnh tương ứng của D1 có kích thước cửa sổ là 1	86
4.9	Từ mục tiêu và từ ngữ cảnh tương ứng của tài liệu D1 khi kích thước cửa sổ là 2	87
4.10	Ma trận trọng số của tầng ẩn sau khi huấn luyện xong	91
4.11	Ma trận trọng số của tầng kết quả sau khi huấn luyện xong	91
7.1	Perplexity tương ứng với các n-gram khác nhau	165
7.2	Khoảng cách giữa "kitten" và "sitting".	192
7.3	Khoảng cách phát âm giữa "pear" (pe-a) và "pair" (pe-a).	193
7.4	Khoảng cách phát âm giữa "strawberry" (stro-be-ri) và "pair" (pe-a).	193
5	Bảng các thuật ngữ	198

Chương 1

Giới thiệu mạng nơ-ron nhân tạo

1.1 Lịch sử mạng nơ-ron

Các *mạng nơ-ron* (neural network) lần đầu tiên được biết đến vào năm 1943 khi Warren McCulloch, một nhà sinh lý học thần kinh và Walter Pitts, một nhà toán học trẻ, 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) (McCulloch and Pitts, 1943) về cách các nơ-ron có thể hoạt động và 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 cung cấp 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, nhà tâm lý học Frank Rosenblatt 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. Trước đó, tác giả này cũng đã viết một cuốn sách về điện toán thần kinh (Rosenblatt, 1959).

Một hệ thống khác được đặt tên là ADALINE (ADAptive LInear Element) được phát triển vào năm 1960 bởi hai kỹ sư điện Bernard Widrow và Marcian Hoff (Widrow

and Hoff, 1960). Phương pháp được sử dụng cho việc học trong hệ thống này khác với phương pháp của perceptron (Rosenblatt, 1958), 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 thực sự được sử dụng trong các sản phẩm công nghiệp cho đến ngày nay, tiêu biểu là áp dụng vào tác vụ lọc 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” (Minsky and Papert, 1969), 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) có số tầng nơ-ron lớn hơn 1 (là điều mà các kiến trúc ANN ngày nay có, càng nhiều lớp thì được coi là càng sâu), thay vì chỉ có 1 tầng nơ-ron như perceptron, thì giải quyết được bài toán mô phỏng các hàm logic mà ANN 1 lớp không thực hiện 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 phải đợi lốt dưới các tên gọi khác, chẳng hạn như *xử lý tín hiệu thích nghi* (adaptive signal processing), *nhận dạng mẫu* (pattern recognition), hay *mô hình sinh học* (biological modeling).

Mãi đến thập kỉ 1980, có nhiều sự kiện diễn ra khiến sự quan tâm đến lĩnh vực này trỗi dậy. Tiên phong là John Hopfield vào năm 1982 đã trình bày một bài báo có tên “Neural Networks and Physical Systems with Emergent Collective Computational Abilities” (Hopfield, 1982a). Hopfield mô tả mạng thần kinh nhân tạo tái phát phục vụ như *hệ thống nội dung bô nhớ địa chỉ* (content-addressable memory system). Các công trình của ông đã thuyết phục rất nhiều 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.

Minh chứng rõ ràng nhất cho điều trên chính là sự thành lập liên tiếp của nhiều hội nghị và tạp chí trên khắp thế giới dành riêng cho lĩnh vực nghiên cứu về mạng

nơ-ron. Khởi đầu là *Viện Vật lý Hoa Kỳ* (the American Institute of Physics) vào năm 1985 đã quyết định tổ chức một hội thảo thường niên có tên “*Mạng Nơ-ron cho Máy tính*” (Neural Networks for Computing). Tiếp sau đó, vào năm 1987, hội thảo quốc tế đầu tiên về mạng nơ-ron trong thời hiện đại mang tên “*Hội nghị Quốc tế về Mạng Nơ-ron*” (IEEE International Conference on Neural Networks) được tổ chức tại San Diego. Vào cùng thời điểm này, “*Hiệp hội Mạng Nơ-ron Quốc tế*” (International Neural Network Society - INNS) được thành lập. Năm 1988, tạp chí “*Neural Networks*” được thành lập, sau đó là “*Neural Computation*” vào năm 1989 và “*IEEE Transactions on Neural Networks*” vào năm 1990.

Những tiến bộ đáng kể đã được minh chứng trong lĩnh vực mạng nơ-ron đủ để thu hút rất nhiều sự chú ý và tài trợ cho việc nghiên cứu sâu hơn. Ngày nay, các cuộc thảo luận về mạng nơ-ron đang diễn ra ở khắp mọi nơi và chưa hề có dấu hiệu hạ nhiệt. Các chip được xây dựng dựa trên lý thuyết nơ-ron đang là chủ đề nổi trội và được phát triển, ứng dụng nhằm giải quyết nhiều vấn đề phức tạp. Không thể phủ nhận rằng, ngày nay là thời kỳ phát triển hoàng kim 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

1.2.1 Cấu tạo và hoạt động của nơ-ron sinh học

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 cần phải hiểu được một nơ-ron sinh học hoạt động như thế nào.

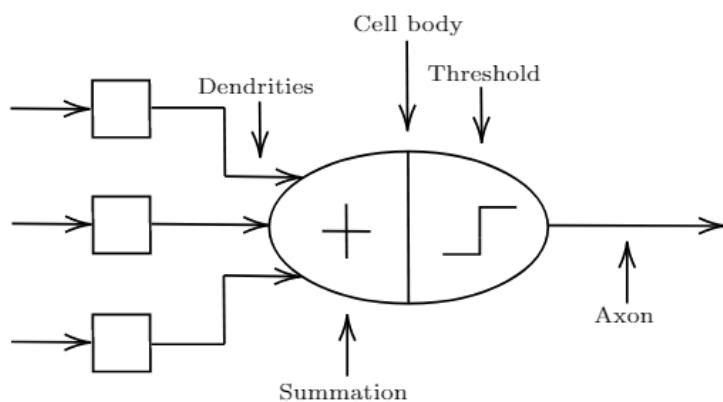
Cấu tạo của một nơ-ron sinh học được minh họa bởi Hình 1.1. Một nơ-ron sẽ nhận các tín hiệu điện (có mức độ mạnh yếu khác nhau) chứa thông tin được truyền tải từ các *khớp thần kinh* (synapse) của một hay nhiều nơ-ron khác thông qua các *đuôi gai* (dendrit). Các giá trị tín hiệu đầu vào này sẽ được tích tụ tại *thân nơ-ron* (cell body).

Nơ-ron sẽ tiến hành thực hiện một quá trình tính toán ngay tại thân nơ-ron bao

gồm 2 bước sau:

- 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* (threshold) cụ thể.

Nếu kết quả của các phép tính toán trên vượt quá giá trị ngưỡng đã đặt ra trước đó, nơ-ron sẽ phát ra một tín hiệu điện, tín hiệu này được truyền qua *sợi trục* (axon) và đi tới các *khớp thần kinh* (synapse). Lúc này, nơ-ron đó được gọi là đang *kích hoạt* (*activation*).



Hình 1.1: Cấu tạo của một nơ-ron sinh học

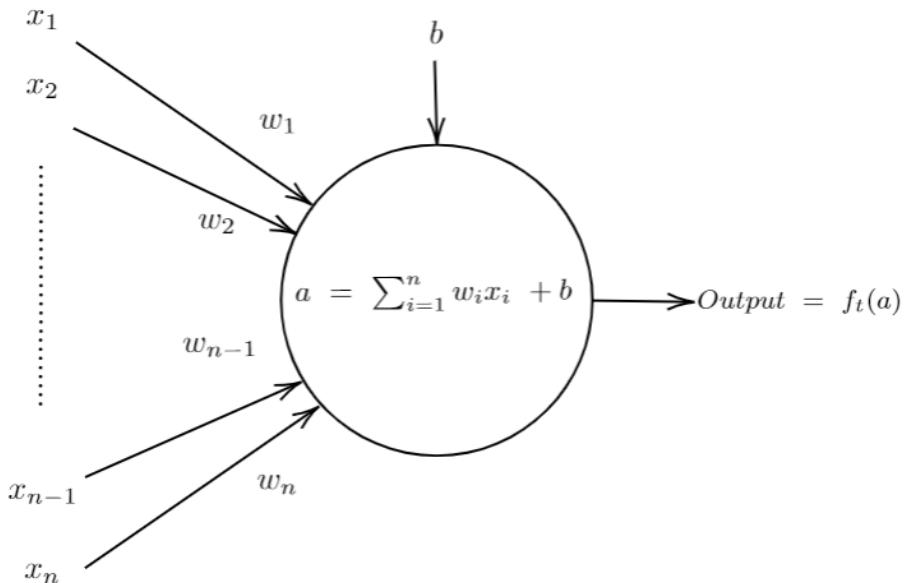
1.2.2 Cấu tạo và hoạt động của perceptron

Một nơ-ron nhân tạo (perceptron) được cấu tạo bao gồm các thành phần minh họa như Hình 1.2. Trong đó:

- Các giá trị x_1, x_2, \dots, x_n là các tín hiệu đầu vào;
- Các giá trị w_1, w_2, \dots, w_n là các trọng số của các nhánh tương ứng truyền giá trị x_1, x_2, \dots, x_n . Giá trị b (còn có thể ký hiệu là x_0) được gọi là giá trị bias, có thể nhận một giá trị bất kỳ khác 0;
- Giá trị a là kết quả của phép tính cộng b 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 $f_t(a)$;

- Kết quả của hàm $f_t(a)$ 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.



Hình 1.2: Cấu tạo và cách thức hoạt động 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

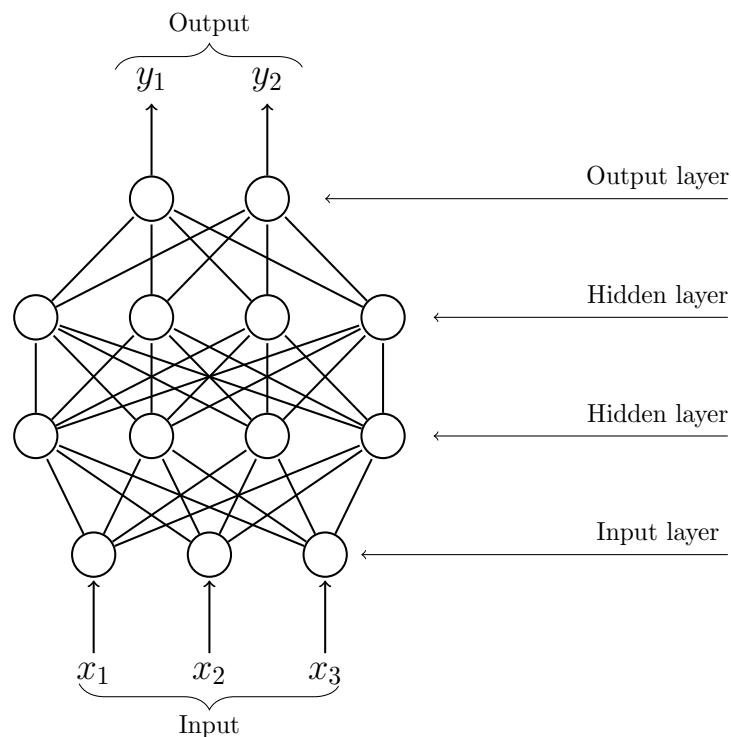
Có thể thấy rằng, một perceptron đã có thể được coi là một mạng nơ-ron, tính toán kết quả dựa trên tín hiệu đầu vào và kết quả đầu ra là perceptron đó có được kích hoạt hay không. Trong thực tế, với chỉ một perceptron thì đã có thể giải quyết được bài toán phân loại tuyến tính. Tuy nhiên, đối với các 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



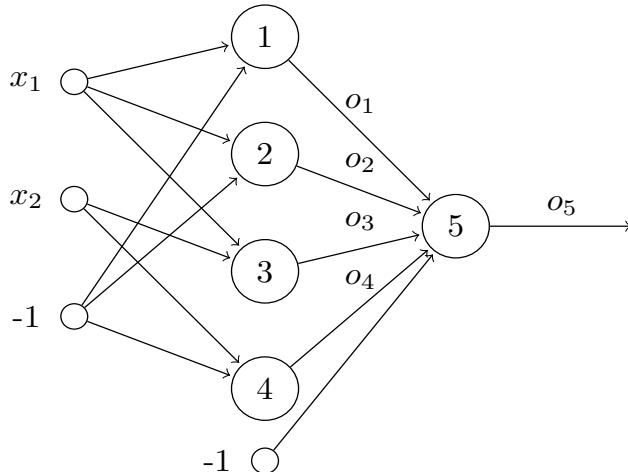
Hình 1.3: Kiến trúc mạng nơ-ron đa tầng với 2 tầng ẩn

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 3 perceptron và tầng kết quả gồm 2 perceptron. Ta có thể thấy mạng này có 2 tầng ẩn nằm ở giữa tầng dữ kiện và tầng kết quả.

Ví dụ 1.3.1. Cho một mạng nơ-ron 3 tầng với các vector trọng số như Hình 1.4.



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

Các vector trọng số có giá trị như sau:

$$\begin{aligned} w_1 &= [1 \ 0 \ -1]^T & w_2 &= [-1 \ 0 \ -2]^T \\ w_3 &= [0 \ 1 \ 0]^T & w_4 &= [0 \ -1 \ -3]^T \\ w_5 &= [1 \ 1 \ 1 \ 1 \ 3.5]^T \end{aligned}$$

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 được kích hoạt?

Trả lời. Để mạng nơ-ron trong Hình 1.4 được kích hoạt thì nơ-ron số 5 phải được kích hoạt, tức o_5 phải bằng 1, tương đương với:

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

Nhằm thỏa mãn biểu thức trên, nơ-ron số 1, 2, 3 và 4 phải được kích hoạt để o_1, o_2, o_3 và o_4 bằng 1. Điều này có nghĩa là v_1, v_2, v_3, v_4 đều phải lớn hơn hoặc bằng 0. Ta có:

$$\begin{aligned} v_1 &= x_1 \times 1 + x_2 \times 0 + (-1) \times (-1) = x_1 + 1 \\ \Rightarrow v_1 \geq 0 &\Leftrightarrow x_1 \geq -1; \end{aligned} \tag{1.1}$$

$$\begin{aligned} v_2 &= x_1 \times (-1) + x_2 \times 0 + (-1) \times (-2) = -x_1 + 2 \\ \Rightarrow v_2 \geq 0 &\Leftrightarrow x_1 \leq 2; \end{aligned} \tag{1.2}$$

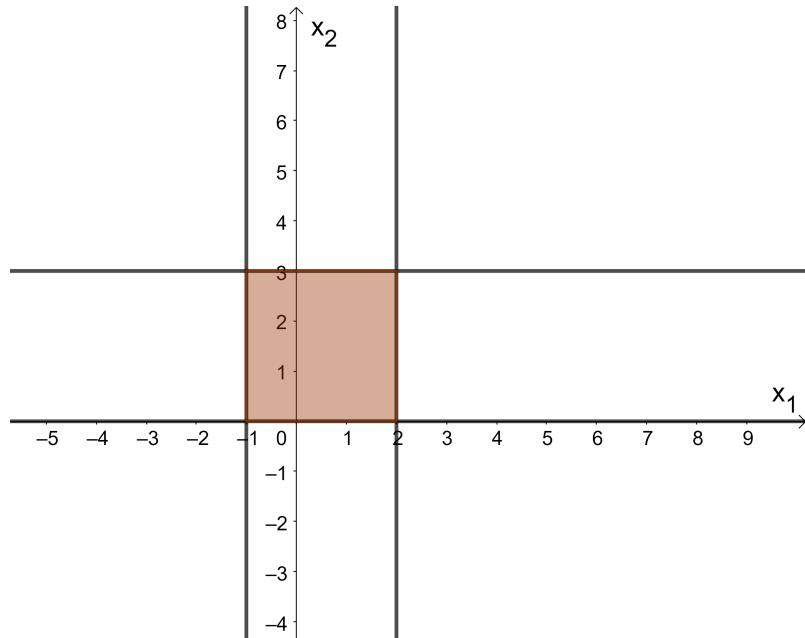
$$\begin{aligned} v_3 &= x_1 \times 0 + x_2 \times 1 + (-1) \times 0 = x_2 \\ \Rightarrow v_3 \geq 0 &\Leftrightarrow x_2 \geq 0; \end{aligned} \tag{1.3}$$

$$\begin{aligned} v_4 &= x_1 \times 0 + x_2 \times (-1) + (-1) \times (-3) = -x_2 + 3 \\ \Rightarrow v_4 \geq 0 &\Leftrightarrow x_2 \leq 3; \end{aligned} \tag{1.4}$$

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

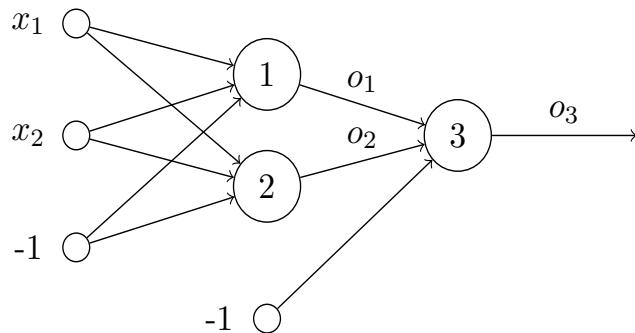
$$-1 \leq x_1 \leq 2 \quad \& \quad 0 \leq x_2 \leq 3$$

Dựa vào kết quả thu được, có thể dễ dàng vẽ miền giá trị cần tìm là vùng được tô đậm trong Hình 1.5.



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

Ví dụ 1.3.2. Cho một mạng nơ-ron 3 tầng với các vector trọng số như Hình 1.6.



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

Giả sử rằng:

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

Các vector trọng số có giá trị như sau:

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

Hàm kích hoạt:

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

Hỏi với giá trị nào của x_1 và x_2 thì mạng nơ-ron này kích hoạt?

Trả lời. Tương tự như Ví dụ 1.3.1, có thể dễ dàng thấy rằng, mạng nơ-ron được biểu diễn trong Hình 1.6 được kích hoạt khi và chỉ khi o_3 được kích hoạt, tương tự với:

$$\begin{aligned} o_3 &= 1 \\ \Leftrightarrow o_1 \times (-2) + o_2 \times 1 + (-1) \times 0.5 &\geq 0 \\ \Rightarrow -2o_1 + o_2 - 0.5 &\geq 0 \end{aligned}$$

Do o_1 và o_2 chỉ nhận giá trị là 0 hoặc 1, nên biểu thức trên được thỏa mãn khi và chỉ khi $o_1 = 0$ và $o_2 = 1$. Trong khi đó:

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

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

Kết hợp yêu cầu của bài toán với kết quả suy ra từ 2 biểu thức (1.5) và (1.6) có thể dễ dàng kết luận kết quả cần tìm là cặp giá trị x_1, x_2 thỏa mãn các điều kiện sau:

$$\begin{cases} 0.5 \leq x_1 + x_2 < 1.5 \\ 0 \leq x_1 \leq 1 \\ 0 \leq x_2 \leq 1 \end{cases}$$

Đặc biệt, nếu 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 Hình 1.7).

Đầu vào		Đầu ra
x_1	x_2	
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 giải quyết bài toán phân loại. Đối với bài toán phân loại tuyến tính thì chỉ cần xây dựng mạng nơ-ron với một perceptron. Cùng xem xét ví dụ sau đây:

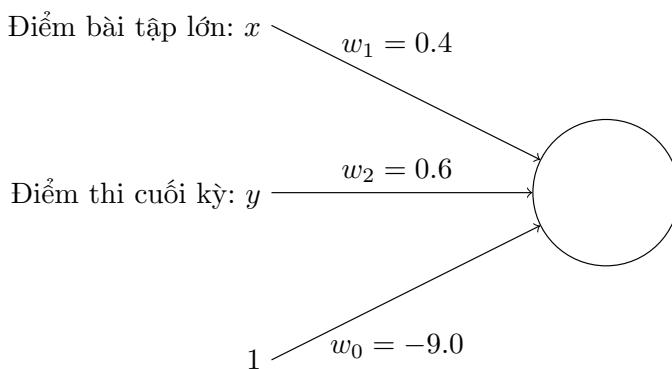
Ví dụ 1.4.1. Một sinh viên được coi là xuất sắc trong môn học Xử lí ngôn ngữ tự nhiên nếu điểm tổng kết môn học lớn hơn hoặc bằng 9.0 điểm, cấu tạo của điểm tổng kết là 40% điểm bài tập lớn và 60% điểm thi cuối kỳ.

Hãy xây dựng một mạng nơ-ron để xác định một sinh viên có được xếp loại xuất sắc hay không, trong đó điểm bài tập lớn được ký hiệu là x , điểm thi cuối kỳ được ký hiệu là y . Kết quả của mạng nơ-ron là 0 hoặc 1, tương đương với học sinh xuất sắc hoặc không.

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

$$\begin{aligned} 0.4 \times x + 0.6 \times y &\geq 9.0 \\ \Leftrightarrow 0.4 \times x + 0.6 \times y - 9.0 &\geq 0 \end{aligned}$$

Như vậy, khi chuyển sang biểu diễn dưới dạng một mạng nơ-ron, ta có tập giá trị đầu vào là một vector $[x, y, 1]$ và tập trọng số tương ứng có giá trị là $[0.4, 0.6, -9.0]$, với $w_0 = -9.0$ (trọng số bias tùy ý khác 0).



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

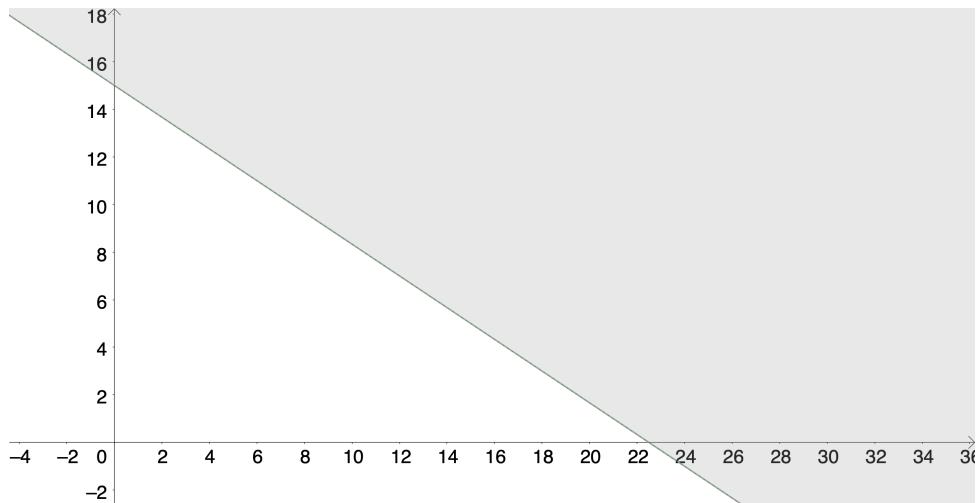
Ta có phép tổng chập (summation) là:

$$a = 0.4 \times x + 0.6 \times y - 9.0$$

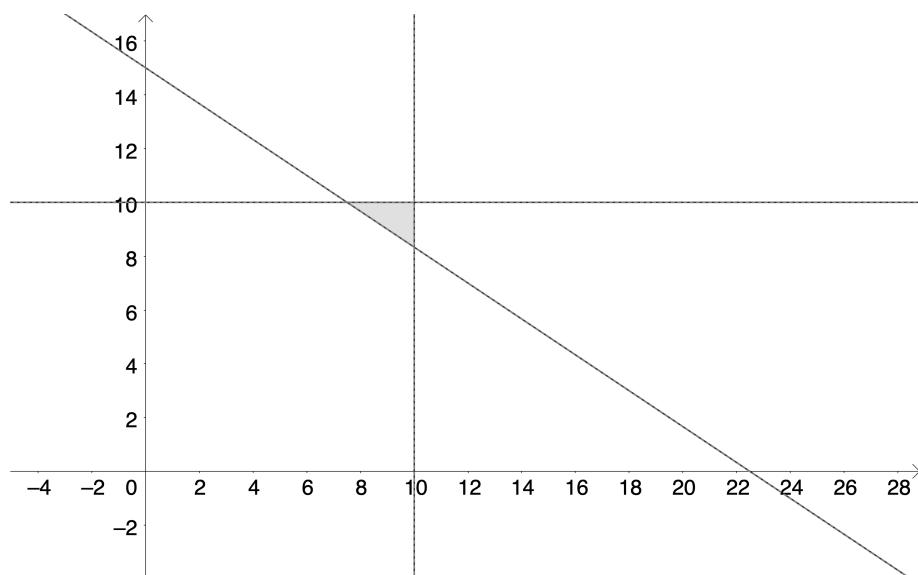
Mạng nơ-ron này được kích hoạt (ứng với việc học sinh được xếp loại xuất sắc) khi và chỉ khi $a \geq 0$, tương ứng với vùng được tô đậm trong Hình 1.9.

Tuy nhiên, thực tế điểm bài tập lớn và điểm thi cuối kỳ luôn nhận giá trị là số dương và bé hơn hoặc bằng 10. Khi gán thêm điều kiện này, ta có kết quả là miền giá trị được tô đậm trong đồ thị Hình 1.10.

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

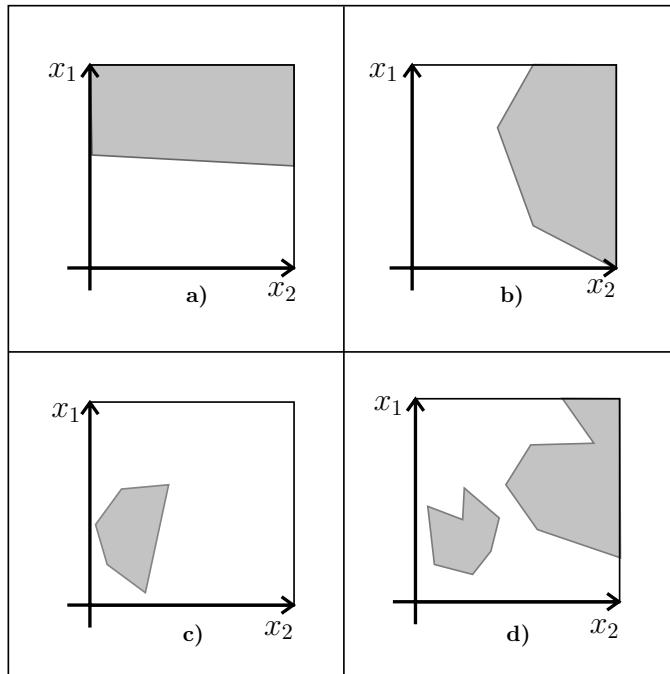


Hình 1.9: Miền giá trị thỏa mãn yêu cầu Ví dụ 1.4.1



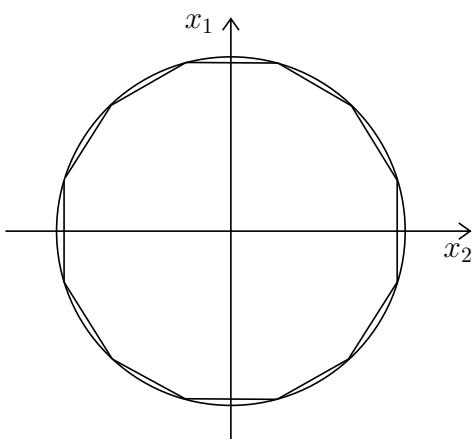
Hình 1.10: Miền giá trị thỏa mãn yêu cầu Ví dụ 1.4.1 sau khi ràng buộc về khoảng giá trị của điểm bài tập lớn và điểm thi cuối kỳ

Ví dụ 1.4.1 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, c, d) thì cần phải xây dựng mạng nơ-ron 3 tầng.



Hình 1.11: Đồ thị biểu diễn các miền giá trị tuyến tính (a) và phi tuyến (b, c, d)

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 như Hình 1.12



Hình 1.12: Đồ thị biểu diễn gần đúng miền giá trị là hình tròn

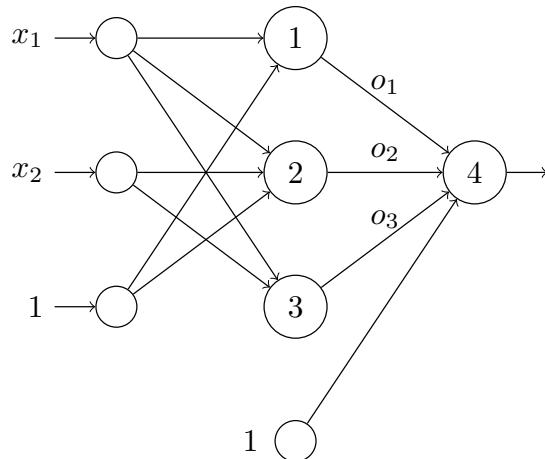
1.5 Kết chương

Có thể dễ dàng 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 thì hoàn toàn có thể áp dụng mạng nơ-ron một cách hiệu quả, nhanh chóng bằng cách cấu hình sẵn phần cứng với các bộ trọng số cần thiết.

Tuy nhiên, trong thực tế 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 cho mỗi tầng. 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 3 tầng và các vector trọng số như Hình 1.13.



Hình 1.13: Mạng nơ-ron trong Bài tập 1.6.1

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

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

Hàm kích hoạt

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

Với giá trị nào của x_1 và x_2 thì mạng nơ-ron này được kích hoạt? Vẽ biểu đồ minh họa cho mạng trên.

Bài tập 1.6.2. Trong Bài tập 1.6.1, nếu như thay đổi giá trị trọng số bias ở cả tầng dữ kiện và tầng ẩn 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.

Bài tập 1.6.3. Thiết kế một mạng nơ-ron để chọn ra các sinh viên đạt danh hiệu giỏi toàn diện. Tiêu chí đánh giá bao gồm điểm trung bình (số nguyên dương), hạnh kiểm (bao gồm các mức 0, 1, 2 và 3 tương ứng với loại Yếu, Trung Bình, Khá và Giỏi) và điểm rèn luyện (thấp nhất là 0 điểm và cao nhất là 10 điểm).

Xét biểu thức sau:

$$\text{điểm danh hiệu} = \text{điểm trung bình} \times 2 + \text{điểm hạnh kiểm} + \text{điểm rèn luyện}$$

Sinh viên sẽ được nhận danh hiệu toàn diện nếu như $\text{điểm danh hiệu} \geq 24$.

Bài tập 1.6.4. Trong Bài tập 1.6.3, nếu như thay đổi giá trị trọng số bias đã chọn ban đầu thành một giá trị khác (tùy chọn), thì tập trọng số sẽ cần được thay đổi như thế nào?

Chương 2

Hồi quy Logistic (Logistic Regression)

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 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:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \quad (2.1)$$

với x_0 là giá trị chính giữa của đường cong logistic;

k là tỷ lệ tăng của hàm logistic;

L là giá trị cực đại của hàm logistic.

Mô hình hồi quy logistic thường được sử dụng cho các bài toán phân loại, đặc biệt là phân loại nhị phân, mặc dù trong tên có chứa từ hồi quy (“*regression*”). Nội dung phần tiếp theo sẽ làm rõ lí do cho điều này.

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

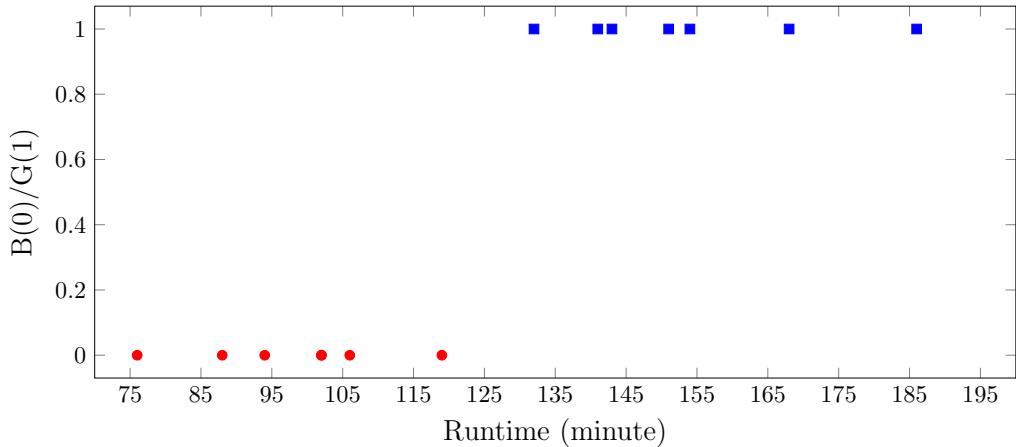
Cũng giống như *hồi quy tuyến tính* (linear regression) (Freedman, 2009), hồi quy logistic được sử dụng để giải bài toán *học có giám sát* (supervised learning) (Russel, Norvig, et al., 2013). Tuy nhiên, hồi quy logistic được dùng để giải các bài toán phân loại, còn hồi quy tuyến tính được dùng để dự 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 mà dự đoán xem bộ phim đó có được khán giả đánh giá tốt (*G*) hay không tốt (*B*). Tập dữ liệu được cung cấp 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 đã được cung cấp 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 biểu diễn các điểm dữ liệu trong Bảng 2.1

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 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{y} nằm trong khoảng $[0, 1]$.

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

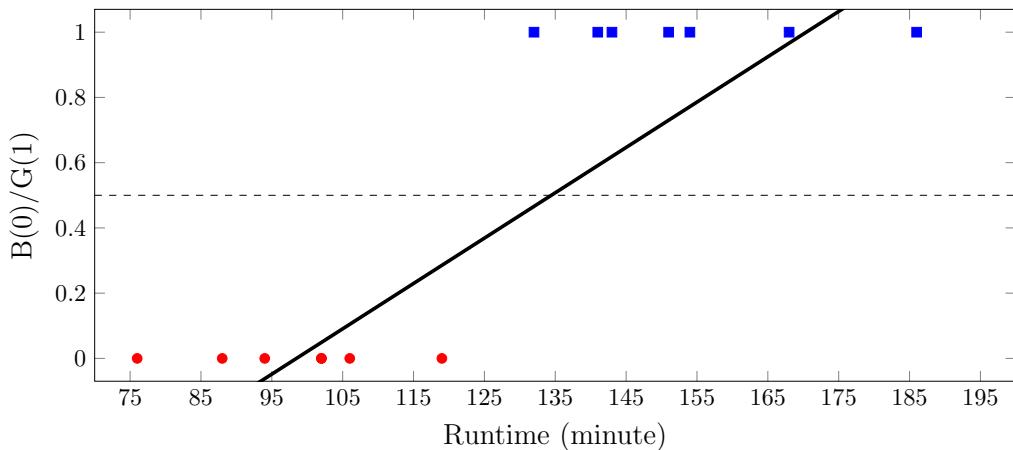
$$\hat{y} = z = \mathbf{W}^T \mathbf{X} + b \quad (2.2)$$

Khi áp dụng hồi quy tuyến tính vào bài toán trên chúng ta có thể thu được kết quả như Hình 2.2. Trong đó, đường thẳng chia cắt các điểm hình tròn và các điểm hình vuông biểu diễn đường hồi quy tuyến tính. Có 2 vấn đề có thể thấy được từ kết quả này:

- Đường thẳng hồi quy tuyến tính không bị chặn nên có thể cho ra kết quả không nằm trong vùng mong muốn. Như trong Hình 2.2, đường thẳng có thể cho ra giá trị $\hat{y} > 1$ và $\hat{y} < 0$ nên mô hình này không phù hợp cho bài toán

này. Tuy nhiên, chúng ta có thể chặn đường thẳng này 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 $\hat{y} > 0.5$, 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 đoán là không tốt, nhưng những bộ phim được đánh giá tốt sẽ được dự đoán là không tốt.



Hình 2.2: Kết quả khi dùng hồi quy tuyến tính

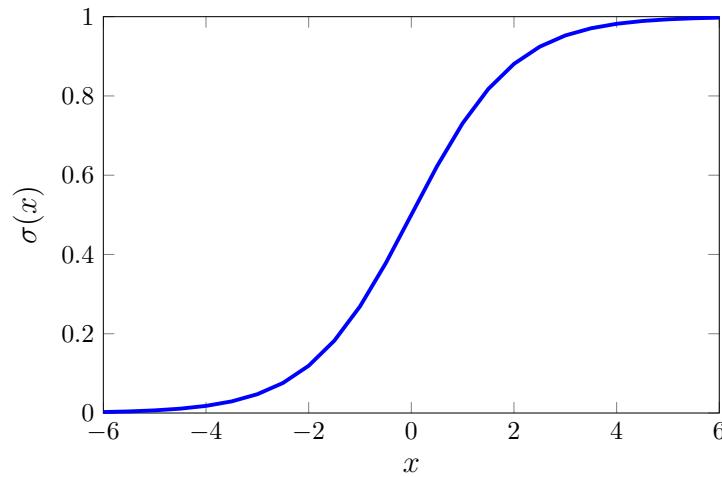
Nhờ vào những nhận xét trên mà sinh ra nhu cầu về một mô hình khác có khả năng tạo ra đường phân chia tốt hơn so với hồi quy tuyến tính và thỏa mãn các đặc tính sau:

- Là hàm số nhận giá trị thực, bị chặn trong khoảng $(0,1)$;
- Nếu coi điểm (thuộc đường phân chia) 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;

- 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

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



Hình 2.3: Đồ thị hàm sigmoid.

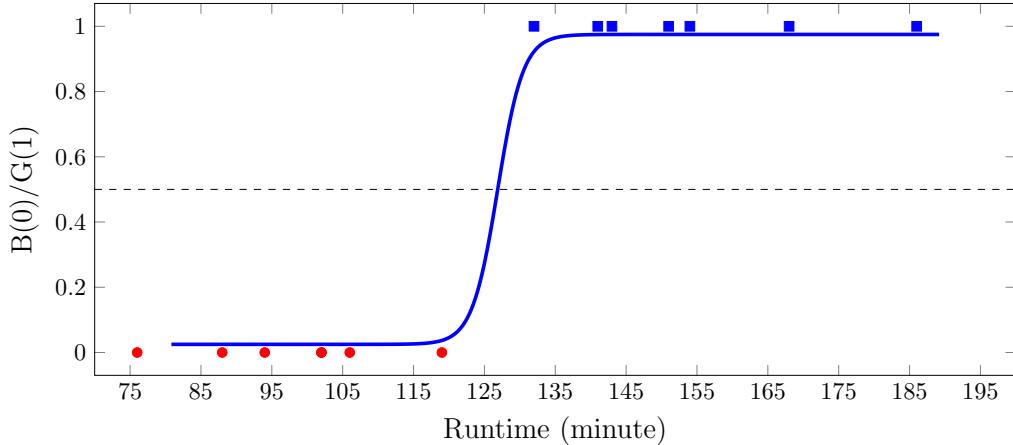
Từ phương trình (2.3) và đồ thị Hình 2.3, ta có thể dễ dàng thấy rằng:

$$\begin{cases} \lim_{s \rightarrow -\infty} \sigma(s) = 0 \\ \lim_{s \rightarrow +\infty} \sigma(s) = 1 \end{cases}$$

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 $\mathbf{W}^T \mathbf{X} + b$ làm dữ liệu đầu vào của hàm sigmoid. Phương trình mới của kết quả dự đoán \hat{y} :

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

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,

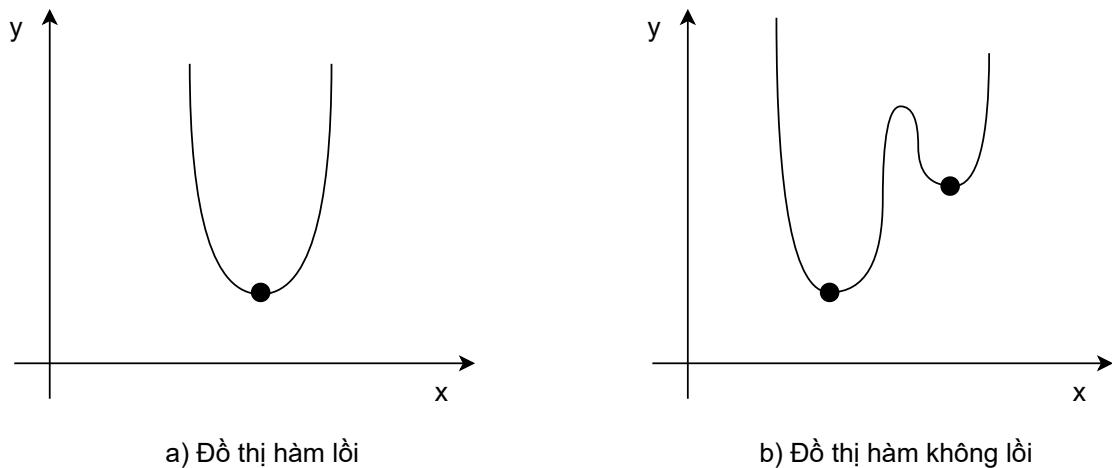


Hình 2.4: Kết quả khi dùng hồi quy logistic.

2.3 Hàm chi phí của hồi quy logistic

Trước khi đi vào nội dung chính của phần này, cần phải tìm hiểu về các 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 $\forall x, y \in [\alpha, \beta]$ và $\forall a, b \geq 0$ 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 duy nhất (Hình 2.5.a) 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. Dồ thị hàm không lồi thường có dạng sóng (Hình 2.5.b) với nhiều điểm cực trị (cực tiểu hoặc cực đại) (*local optimal*) (Stewart et al., 2020).



Hình 2.5: Đồ thị hàm lồi (a) và đồ thị hàm không lồi (b)

Nhắc lại, trong mô hình hồi quy logistic, phương trình dự đoán là:

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

Cũng như những phương pháp khác được dùng để giải các bài toán học có giám sát, quá trình học của mô hình hồi quy logistic là quá trình cập nhật trọng số \mathbf{W} . Thế nên 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:

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

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 $\hat{y} = \sigma(\mathbf{W}^T \mathbf{X} + b)$ nên khi đó LSE là một hàm không lồi với nhiều điểm cực tiểu (Hình 2.5.b).

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 măt măt khi đă sē là:

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

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

Phần tiếp theo giải thích được tại sao hàm cross-entropy có thể giúp mô hình hồi quy logistic tìm được điểm tối ưu toàn cục bằng cách chứng minh hàm măt măt dạng cross-entropy cho mô hình hồi quy logistic là hàm lồi.

Hàm logistic của mô hình hồi quy logistic:

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

Hàm măt măt của mô hình hồi quy logistic:

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

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

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

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

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

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

công thức đạo hàm bậc 1 theo w là:

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

và công thức đạo hàm bậc 2 theo w là:

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

Trong đó $H \circ (1 - H) = \begin{bmatrix} \lambda_1^2 \\ \lambda_2^2 \\ \dots \\ \lambda_m^2 \end{bmatrix}$ với $\lambda_i = \sqrt{h(x^{(i)})(1 - h(x^{(i)}))} \geq 0 \quad \forall i = 1 \dots m$

Lúc này, $\forall \mathbf{z} \in \mathbb{R}^D$ thì:

$$\frac{1}{m} \mathbf{z}^T \mathbf{X} (\mathbf{H} \circ (1 - \mathbf{H}) \circ \mathbf{X}) \mathbf{z} = \frac{1}{m} \left\| \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \dots \\ \lambda_m \end{bmatrix} \circ \mathbf{Xz} \right\|^2 \geq 0 \quad (2.14)$$

Điều này có nghĩa rằng $\frac{\partial^2 J}{\partial w}$ là một ma trận nửa xác định dương (PSD).

Như vậy có thể kết luận $J(w)$ là một hàm lồi (điều cần chứng minh).

Ở đây, phuwog trình (2.6) được định nghĩa là hàm mất mát của một điểm dữ liệu đầu vào. Vậy với tập dữ liệu đầu vào có kích thước là m thì sẽ có được phương trình tổng quát của *hàm chi phí* (cost function) trong mô hình hồi quy logistic như phương trình sau:

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

2.4 Sử dụng gradient descent để tìm giá trị tối ưu

Nhắc lại về hàm chi phí của hồi quy logistic:

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

$$\text{với } \hat{y} = \sigma(\mathbf{W}^T \mathbf{X} + b) \text{ và } \sigma(z) = \frac{1}{1 + e^{-z}}$$

Hàm chi phí này cho biết mức độ sai khác giữa giá trị đầu ra được tính toán 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 nghĩa là giá trị tính toán được sẽ càng

giống với giá trị thật sự. Bài toán bây giờ chính là tìm giá trị của w và b sao cho giá trị của hàm chi phí là nhỏ nhất.

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

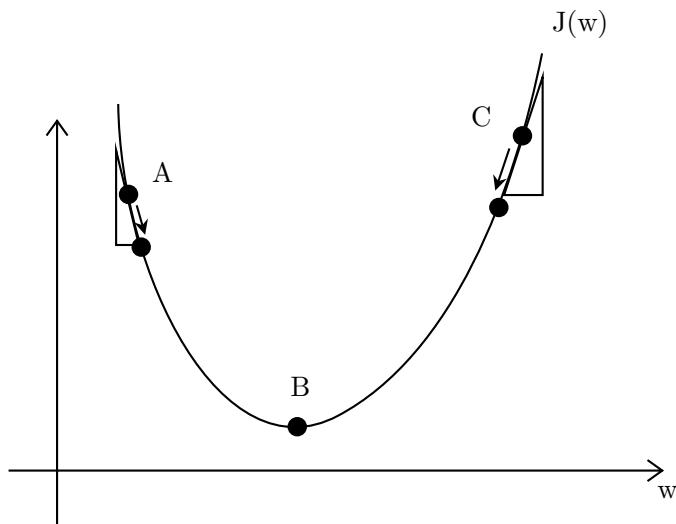
Để hiểu rõ hơn về gradient descent, dưới đây là một ví dụ đơn giản về việc sử dụng phương pháp này để tìm giá trị tối ưu của một hàm số.

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 kỳ 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 descent là:

$$w = w - \alpha \frac{\delta J(w)}{\delta w} \quad (2.17)$$

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.

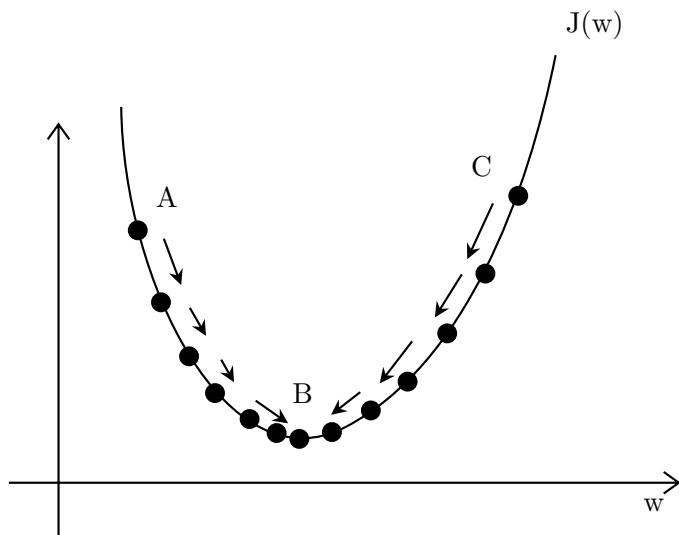


Hình 2.6: Đồ thị minh họa kỹ thuật gradient descent

Dặc điểm của phương pháp này là quá trình *di ngược chiều đạo hàm*. Nếu lấy điểm khởi đầu là A , thì khi đó, đạo hàm tại điểm A sẽ có giá trị âm (do $\Delta J(w_A) < 0$ còn $\Delta w_A > 0$). Giá trị w được cập nhật sẽ tăng lên (do $-\alpha \frac{\delta J(w_A)}{\delta w_A} > 0$). Có thể thấy rằng 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 $\Delta J(w_C) > 0$ và $\Delta w_C > 0$). Lúc này, 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 bắt kì được chọn trên hàm số đều có thể di chuyển về điểm tối ưu của hàm số.

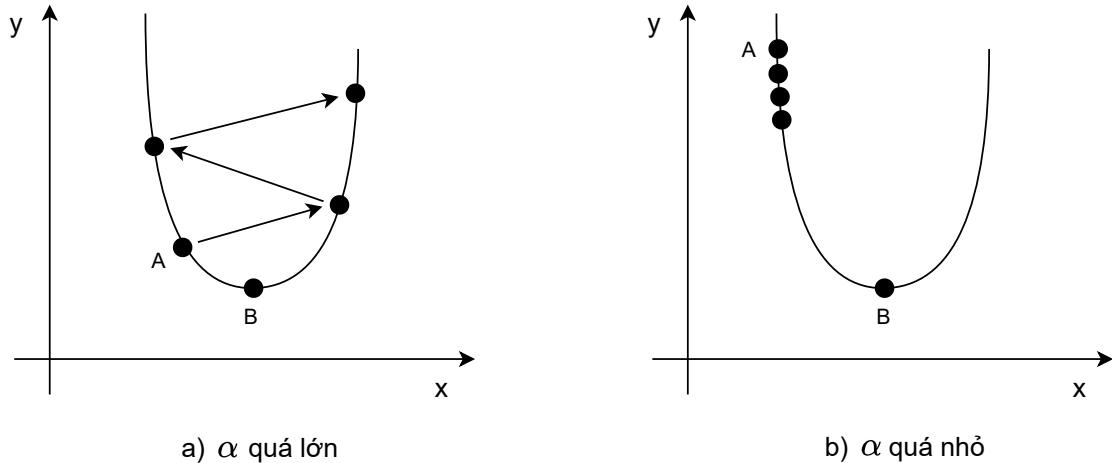


Hình 2.7: Sử dụng gradient descent để tìm giá trị tối ưu

Có thể thấy, khi sử dụng gradient descent, điểm được chọn sẽ dần “tiến về” giá trị tối ưu của hàm số dù bắt đầu ở bất kì ở vị trí nào của hàm số. Như mô tả trong 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 descent tìm giá trị tối ưu, cần phải lựa chọn tốc độ học α không quá lớn cũng không quá nhỏ. Nếu lựa chọn giá trị α quá lớn, quá trình lặp khi sử dụng gradient descent 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.



Hình 2.8: Sử dụng gradient descent để tìm giá trị tối ưu ứng với các giá trị α

Hình 2.8a biểu diễn cách tìm điểm tối ưu của hàm số bằng phương pháp gradient descent với 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 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.

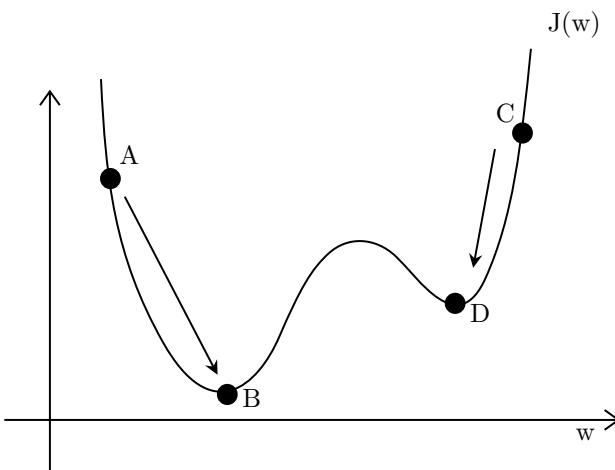
Ngược lại, Hình 2.8b mô tả trường hợp chọn giá trị α quá nhỏ. Với điểm khởi đầu ngẫu nhiên được chọn là A , sau khi tính toán đạo hàm thì điểm này vẫn di chuyển về vị trí tối ưu nhưng bởi vì giá trị α quá nhỏ nên mỗi lần lặp chỉ có thể di chuyển một khoảng rất ngắn. Vì vậy để điểm A “tiến về” điểm tối ưu cần phải lặp đi lặp lại quá trình này rất nhiều lần. Điều này có nghĩa rằng cần phải tiêu tốn nhiều tài nguyên tính toán và thời gian thực thi nhằm đạt được mục đích.

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

Gradient descent 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.

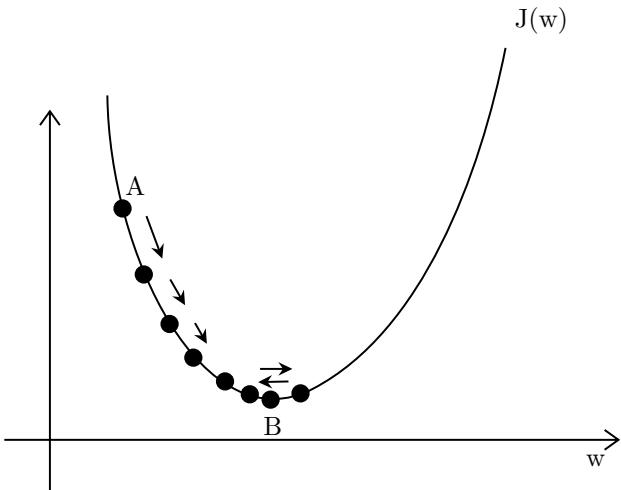
Nếu hàm số khả vi thì có thể tính được đường đi đến điểm tối ưu bằng phương pháp gradient descent dựa trên giá trị của đạo hàm ($w = w - \alpha \frac{\delta J(w)}{\delta w}$). Còn nếu hàm số không khả vi thì không thể tính được giá trị $\frac{\delta J(w)}{\delta w}$ cũng như việc tìm được điểm tối ưu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

Bên cạnh đó, 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 có thể tìm được điểm tối ưu của hàm số hay không. Vì nếu không là hàm lồi, tùy thuộc vào vị trí khởi tạo mà giá trị tối ưu thu được sau khi sử dụng gradient descent có thể là điểm tối ưu cục bộ, không phải điểm tối ưu toàn cục.



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

Đồ thị Hình 2.9 là đồ thị biểu diễn của một hàm không lồi. Đối với những hàm số không lồi, ngoài điểm tối ưu toàn cục thì còn có các điểm tối ưu cục bộ. Giả sử lấy điểm khởi đầu là điểm A , sau khi áp dụng phương pháp gradient descent, điểm tối ưu ta tìm được là điểm tối ưu toàn cục là điểm B . Tuy nhiên, nếu điểm khởi đầu là điểm C , thì sau khi áp dụng gradient descent thì điểm tối ưu tìm được sẽ là điểm D , đây không phải là điểm tối ưu toàn cục của hàm số. Vì thế, khi sử dụng gradient descent với hàm số không là hàm lồi, giá trị tối ưu tìm được có thể không là điểm tối ưu toàn cục. Trên thực tế, khi sử dụng gradient descent chỉ có thể tìm được vị trí gần với điểm tối ưu cục bộ.



Hình 2.10: Kết quả khi áp dụng gradient descent thực tế

Cho đồ thị hàm số như Hình 2.10, điểm khởi đầu là A và sử dụng gradient descent để tìm giá trị tối ưu của hàm số trên. Khi này, kết quả sẽ dần chạy lại gần điểm B . Tuy nhiên, khi đã tiến lại gần B một khoảng nhất định, nếu tiếp tục cập nhật giá trị sau khi áp dụng phương pháp gradient descent, kết quả sẽ vượt qua giá trị tối ưu một khoảng rất nhỏ. Tiếp tục lặp lại các phép tính toán theo phương pháp gradient descent một lần nữa thì sẽ đi qua điểm tối ưu 1 khoảng nhỏ. Và cứ như thế, khi thực hiện gradient descent, kết quả nhận được sẽ dao động một khoảng nhỏ xung quanh giá trị tối ưu B . Tuy nhiên, giá trị sai khác khi dùng gradient descent và giá trị tối ưu là rất nhỏ. Vì thế, kết quả ta thu được sau khi sử dụng gradient descent 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 này gồm 2 biến là w và b nên sau mỗi lần lặp của gradient descent cần phải cập nhật giá trị của cả w và b để thu được giá trị mới nhỏ hơn của hàm chi phí. Có thể thực hiện điều này bằng cách sử dụng đạo hàm riêng từng phần của hàm số theo lần lượt 2 biến.

Công thức hàm chi phí:

$$J(w, b) = \frac{-1}{m} \sum_{i=1}^m y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \quad (2.18)$$

Sau khi tính toán đạo hàm từng phần theo từng biến của hàm chi phí là w và b , kết quả cập nhật của mỗi biến qua mỗi lần lặp của gradient descent là:

$$w = w - \alpha \frac{\delta J(w, b)}{\delta w} \quad (2.19)$$

$$b = b - \alpha \frac{\delta J(w, b)}{\delta b} \quad (2.20)$$

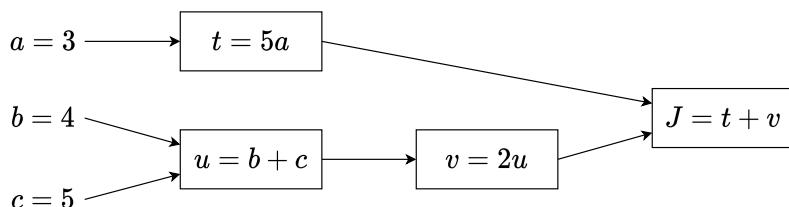
Nếu cập nhật giá trị của w và b với số lần lặp vừa đủ cùng giá trị α vừa phải sẽ tìm ra được giá trị tối ưu của hàm chi phí. Bên cạnh đó, có thể tính được giá trị của $\frac{\delta J(w, b)}{\delta w}$ và $\frac{\delta J(w, b)}{\delta b}$ bằng cách sử dụng *đồ thị tính toán* (computation graph).

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 theo 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 cần tính đạo hàm. Có thể hiểu hơn phương pháp này thông qua ví dụ sau về việc sử dụng đồ thị tính toán để tính đạo hàm riêng.

Ví dụ 2.5.1. Cho hàm số J có phương trình là:

$$J = 5a + 2(b + c) \quad (2.21)$$



Hình 2.11: Sử dụng đồ thị tính toán để biểu diễn hàm số

Hình 2.11 biểu diễn hàm số J cho trước bằng cách sử dụng đồ thị tính toán. Trong đó, 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. Có thể thực hiện việc chia nhỏ biểu thức (2.21) như sau:

$$t = 5a \quad (2.22)$$

$$u = b + c \quad (2.23)$$

$$v = 2u \quad (2.24)$$

$$J = t + v \quad (2.25)$$

Với mỗi biểu thức nhỏ ở trên là một đỉnh của đồ thị thì sẽ xây dựng được một đồ thị tính toán như Hình 2.11.

Tiếp theo là bước tính đạo hàm tại mỗi đỉnh bằng phương pháp xấp xỉ và lan truyền giá trị ngược lại.

Bắt đầu với việc tính đạo hàm tại $a = 3$ bằng phương pháp xấp xỉ. Cần chọn a_1 sao cho Δa rất nhỏ (0.001). Từ biểu thức (2.22) và (2.25), gán $a_1 = 3.001$, suy ra:

$$t_1 = 5a_1 = 15.005 \quad (2.26)$$

$$J_1 = t_1 + v = 33.005 \quad (2.27)$$

Tính giá trị đạo hàm của J theo a :

$$\frac{\Delta t}{\Delta a} = \frac{t_1 - t}{a_1 - a} = \frac{15.005 - 15}{3.001 - 3} = 5 \quad (2.28)$$

$$\frac{\Delta J}{\Delta t} = \frac{J_1 - J}{t_1 - t} = \frac{33.005 - 33}{15.005 - 15} = 1 \quad (2.29)$$

$$\frac{\Delta J}{\Delta a} = \frac{\Delta J}{\Delta t} \cdot \frac{\Delta t}{\Delta a} = 1 \cdot 5 = 5 \quad (2.30)$$

Có thể tiến hành xấp xỉ $\frac{\Delta J}{\Delta a}$ bằng $\frac{\delta J}{\delta a}$. Từ công thức (2.30), giá trị đạo hàm của J tại $a = 3$ xấp xỉ:

$$\frac{\Delta J}{\Delta a} = 5 \quad (2.31)$$

Tương tự tính giá trị đạo hàm của J tại $b = 4$ và $c = 5$. Với $b = 4$, chọn $b_1 = 4.001$. Từ các phương trình (2.23), (2.24) và (2.25) suy ra:

$$u_1 = b_1 + c = 9.001 \quad (2.32)$$

$$v_1 = 2u_1 = 18.002 \quad (2.33)$$

$$J_1 = t + v_1 = 33.002 \quad (2.34)$$

Tính giá trị đạo hàm của J theo v :

$$\frac{\Delta J}{\Delta v} = \frac{J_1 - J}{v_1 - v} = \frac{33.002 - 33}{18.002 - 18} = 1 \quad (2.35)$$

Sau đó, lan truyền kết quả trên để tính đạo hàm của J theo u dựa trên đạo hàm của J theo v (2.35) và đạo hàm của v theo u :

$$\frac{\Delta v}{\Delta u} = \frac{v_1 - v}{u_1 - u} = \frac{18.002 - 18}{9.001 - 9} = 2 \quad (2.36)$$

$$\frac{\Delta J}{\Delta u} = \frac{\Delta J}{\Delta v} \cdot \frac{\Delta v}{\Delta u} = 1 \times 2 = 2 \quad (2.37)$$

Tiếp tục lan truyền kết quả trên để tính đạo hàm của J theo b dựa trên đạo hàm của J theo u (2.37) và đạo hàm của u theo b :

$$\frac{\Delta u}{\Delta b} = \frac{u_1 - u}{b_1 - b} = \frac{9.001 - 9}{4.001 - 4} = 1 \quad (2.38)$$

$$\frac{\Delta J}{\Delta b} = \frac{\Delta J}{\Delta u} \cdot \frac{\Delta u}{\Delta b} = 2 \times 1 = 2. \quad (2.39)$$

Từ (2.39), giá trị đạo hàm của J tại $b = 4$ xấp xỉ bằng 2.

Cuối cùng, với $c = 5$, chọn $c_1 = 5.001$. Từ các phương trình (2.23), (2.24), (2.25) suy ra:

$$u_1 = b + c_1 = 9.001 \quad (2.40)$$

$$v_1 = 2u_1 = 18.002 \quad (2.41)$$

$$J_1 = t + v_1 = 33.002 \quad (2.42)$$

Tính giá trị đạo hàm của J theo v :

$$\frac{\Delta J}{\Delta v} = \frac{J_1 - J}{v_1 - v} = \frac{33.002 - 33}{18.002 - 18} = 1 \quad (2.43)$$

Lan truyền kết quả trên để tính đạo hàm của J theo u dựa trên đạo hàm của J theo v (2.43) và đạo hàm của v theo u :

$$\frac{\Delta v}{\Delta u} = \frac{v_1 - v}{u_1 - u} = \frac{18.002 - 18}{9.001 - 9} = 2 \quad (2.44)$$

$$\frac{\Delta J}{\Delta u} = \frac{\Delta J}{\Delta v} \cdot \frac{\Delta v}{\Delta u} = 1 \times 2 = 2 \quad (2.45)$$

Tiếp tục với việc lan truyền kết quả trên để tính đạo hàm của J theo c dựa trên đạo hàm của J theo u (2.45) và đạo hàm của u theo c :

$$\frac{\Delta u}{\Delta c} = \frac{u_1 - u}{c_1 - c} = \frac{9.001 - 9}{5.001 - 5} = 1 \quad (2.46)$$

$$\frac{\Delta J}{\Delta c} = \frac{\Delta J}{\Delta u} \cdot \frac{\Delta u}{\Delta c} = 2 \times 1 = 2 \quad (2.47)$$

Từ kết quả (2.47), có thể đưa ra kết luận rằng giá trị đạo hàm của J tại $c = 5$ xấp xỉ bằng 2.

Khi tính đạo hàm ở từng bước như Ví dụ 2.5.1, có thể thay việc tính xấp xỉ bằng cách dùng công thức đạo hàm để tính giá trị đạo hàm trực tiếp với các hàm số khả vi. Sau đây là cách sử dụng phương pháp này để tính đạo hàm khi thực hiện quá trình gradient descent.

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):

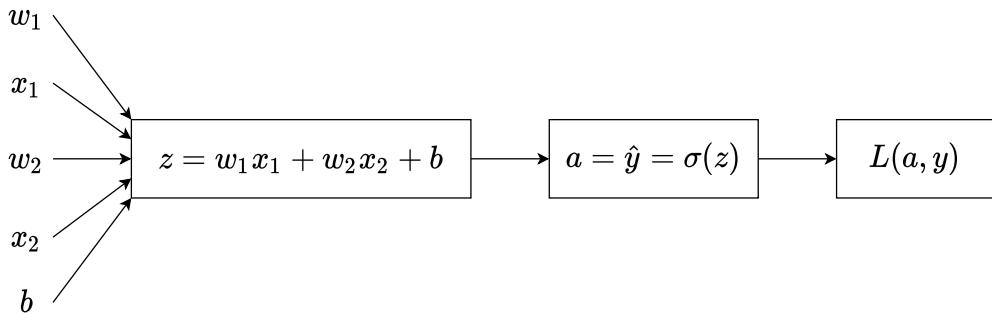
$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a)) \quad (2.48)$$

trong đó:

$$a = \hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} \quad (2.49)$$

$$z = w^T x + b \quad (2.50)$$

Tương tự như Ví dụ 2.5.1, có thể dễ dàng vẽ được đồ thị tính toán của hàm số (2.48) như Hình 2.12.



Hình 2.12: Đồ thị tính toán hàm mất mát của hồi quy logistic

Tính đạo hàm của \mathcal{L} theo a từ phương trình (2.48):

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial a} &= -\left(y \frac{1}{a} + (1 - y) \frac{-1}{1 - a}\right) = \frac{1 - y}{1 - a} - \frac{y}{a} \\ &= \frac{a(1 - y) - y(1 - a)}{a(1 - a)} = \frac{a - y}{a(1 - a)} \end{aligned} \quad (2.51)$$

Sau đó, lan truyền kết quả trên để tính đạo hàm của \mathcal{L} theo z dựa trên đạo hàm của \mathcal{L} theo a từ kết quả (2.51) và đạo hàm của a theo z :

$$\begin{aligned}
\frac{\partial a}{\partial z} &= \frac{0(1+e^{-z}) - (-e^{-z})1}{(1+e^{-z})^2} = \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}} = \frac{1}{1+e^{-z}} \cdot \frac{1+e^{-z}-1}{1+e^{-z}} \\
&= a(1-a)
\end{aligned} \tag{2.52}$$

$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial z} = \frac{a-y}{a(1-a)} a(1-a) = a-y \tag{2.53}$$

Tiếp theo là lan truyền kết quả trên để tính đạo hàm của \mathcal{L} theo w_1, w_2, b dựa trên đạo hàm của \mathcal{L} theo z từ (2.53) và đạo hàm của z theo w_1, w_2, b :

$$\begin{aligned}
\frac{\partial z}{\partial w_1} &= x_1 \\
\frac{\partial \mathcal{L}}{\partial w_1} &= \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial w_1} = (a-y)x_1
\end{aligned} \tag{2.54}$$

$$\begin{aligned}
\frac{\partial z}{\partial w_2} &= x_2 \\
\frac{\partial \mathcal{L}}{\partial w_2} &= \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial w_2} = (a-y)x_2
\end{aligned} \tag{2.55}$$

$$\begin{aligned}
\frac{\partial z}{\partial b} &= 1 \\
\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial b} = (a-y)
\end{aligned} \tag{2.56}$$

Có thể suy ra công thức đồ thị tính toán cho hàm mất mát từ (2.54), (2.55), (2.56) là:

$$w_1 = w_1 - \alpha \frac{\partial \mathcal{L}}{\partial w_1} = w_1 - \alpha(a-y)x_1 \tag{2.57}$$

$$w_2 = w_2 - \alpha \frac{\partial \mathcal{L}}{\partial w_2} = w_2 - \alpha(a-y)x_2 \tag{2.58}$$

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

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

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(a^i, y^i) \quad (2.60)$$

Từ đó, có thể tính được gradient descent của hàm chi phí bằng cách tính tổng gradient descent các hàm mất mát của dữ liệu từ các kết quả tính toán (2.57), (2.58), (2.59), (2.60):

$$w_1 = w_1 - \alpha \frac{\partial J(w, b)}{\partial w_1} = w_1 - \frac{\alpha}{m} \sum_{i=1}^m \frac{\partial \mathcal{L}(a^i, y^i)}{\partial w_1} = w_1 - \frac{\alpha}{m} \sum_{i=1}^m (a^i - y^i)x_1^i \quad (2.61)$$

$$w_2 = w_2 - \alpha \frac{\partial J(w, b)}{\partial w_2} = w_2 - \frac{\alpha}{m} \sum_{i=1}^m \frac{\partial \mathcal{L}(a^i, y^i)}{\partial w_2} = w_2 - \frac{\alpha}{m} \sum_{i=1}^m (a^i - y^i)x_2^i \quad (2.62)$$

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

Một cách tổng quát hơn, với một dữ liệu gồm n đặc trưng thì gradient descent là:

$$\begin{aligned} w_1 &= w_1 - \frac{\alpha}{m} \sum_{i=1}^m (a^i - y^i)x_1^i \\ w_2 &= w_2 - \frac{\alpha}{m} \sum_{i=1}^m (a^i - y^i)x_2^i \\ \vdots w_n &= w_n - \frac{\alpha}{m} \sum_{i=1}^m (a^i - y^i)x_n^i \\ b &= b - \frac{\alpha}{m} \sum_{i=1}^m (a^i - y^i) \end{aligned}$$

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 *làn 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. Algorithm 1 sau là mã giả của giải thuật hồi quy logistic.

Algorithm 1: Logistic regression

```
num_iter = 500, w1 = 0, w2 = 0, ..., wn = 0, b = 0, alpha = 0.1
for i ← 1 to range(num_iter) do
    J = 0, dw1 = 0, dw2 = 0, ..., dwn = 0, db = 0
    for j ← 1 to m do
        z = w1 * xj1 + w2 * xj2 + ... + wn * xjn + b
        a = 1/(1 + e(-z))
        J += -(yj * log(a)) + (1 - yj) * log(1 - a)
        dz = aj - yj
        dw1 += dz * xj1
        dw2 += dz * xj2
        ...
        dwn += dz * xjn
        db += dz
    J/m
    dw1/m
    dw2/m
    ...
    dwn/m
    db/m
    w1 = w1 - alpha * dw1
    w2 = w2 - alpha * dw2
    ...
    wn = wn - alpha * dwn
    b = b - alpha * db
```

Xét trên một tập dữ liệu gồm m dữ liệu và mỗi dữ liệu gồm n đặc trưng. Đầu tiên cần phải khai báo các biến cần thiết sau: các w_1, w_2, \dots, 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 bằng cách 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); α là tốc độ học được thiết lập cho gradient descent (giá trị này không nên quá nhỏ hoặc quá lớn).

Bước tiếp theo là tính gradient descent 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, cần phải khởi tạo giá trị của hàm chi phí J , các giá trị đạo hàm của J theo w_1, w_2, \dots, w_n là dw_1, dw_2, \dots, dw_n và đạo hàm của J theo b là w_b .

Chạy thuật toán qua từng dữ liệu trong tập dữ liệu cho trước và tính giá trị đầu ra. Tiếp đến là 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 thì tính giá trị đạo hàm của J theo từng trọng số theo công thức được suy ra từ phần trước. Cuối cùng là cập nhật giá trị các trọng số dựa trên đạo hàm của chúng theo gradient descent. Quá trình trên sẽ được lặp lại cho đến khi tìm được giá trị tối ưu của hàm chi phí và thu được bộ trọng số thỏa 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 sẽ được trình bày trong chương sau.

2.8 Bài tập

Bảng 2.2: Dữ liệu đầu vào của Bài tập 2.8.1

Cân nặng (kg)	Giới tính
2.4	Nữ
3.3	Nam
3.2	Nam
2.5	Nữ

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, 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 đó là bé nữ và $y = 1$ khi đó là bé 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 $J = 3a + 2(4b + 5cd)$

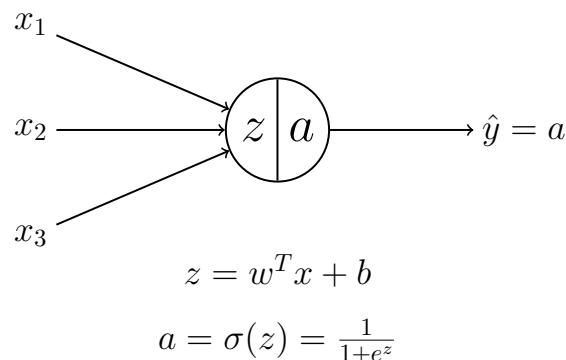
Hãy vẽ đồ thị tính toán của phương trình trên và tính đạo hàm của J tại từng biến a, b, c, d với $a = 2, b = 3, c = 4, d = 5$.

Chương 3

Mạng đa tầng và học sâu (Multilayer perceptron - MLP và Deep learning - DL)

3.1 Hồi quy logistic dưới góc nhìn mạng nơ-ron

Như đã trình bày trong chương trước, *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: Một mạng nơ-ron đơn giản

Hình 3.1 minh họa cho việc dùng hồi quy logistic như một mạng nơ-ron đơn giản. Trong đó, vector x chứa các *thuộc tính* (feature) của dữ kiện đầu vào (input), vector 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 sau:

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

Mặc dù 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 trong chương 2.

Công thức hàm Cross-entropy:

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

Công thức hàm chi phí:

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

3.2 Mạng nơ-ron đa tầng 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 các phép nhân ma trận. Bằng cách này, chúng ta có thể giải quyết rất nhiều bài toán mà 1 perceptron không thể xử lý được.

Tuy nhiên, bên cạnh hàm kích hoạt sigmoid như đã nêu ở chương trước thì các nghiên cứu về mạng nơ-ron đa tầng đưa ra một số dạng hàm kích hoạt khác.

3.2.2 Sự tiến hóa của mạng nơ-ron đa tầng

Quá trình xây dựng hàm kích hoạt sẽ được tiến hành dựa trên những luận điểm như sau:

- Hàm kích hoạt này có thể được thiết kế như một hàm phi tuyến để giải quyết những bài toán phi tuyến.
- Khi đi qua nhiều lớp thì các trọng số có thể suy biến, tức trở nên quá lớn hoặc quá nhỏ, điều này sẽ ảnh hưởng đến độ chính xác của bài toán. Để giải quyết điều này, ta có thể xây dựng một hàm kích hoạt sao cho những giá trị đầu ra sẽ nằm trong một khoảng nào đó có thể kiểm soát được hoặc có ý nghĩa trong xác suất thống kê như (-1,1) hoặc (0,1).
- Khi cần thể hiện độ tin cậy của kết quả, hàm kích hoạt sẽ được xây dựng sao cho giá trị đầu ra nằm trong khoảng (0,1) để thể hiện cho xác suất.

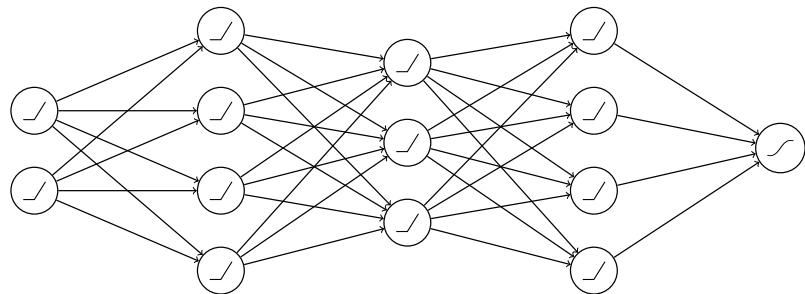
Thông thường, từ (1), (2) và (3) chúng ta sẽ thiết kế hàm kích hoạt sao cho đó là hàm phi tuyến (nhưng phải khả đạo hàm) và đầu ra nên có giá trị (-1,1) hoặc (0,1).

Sau một thời gian phát triển, các mạng học sâu đã ra đời và sử dụng những hàm kích hoạt này.

3.2.3 Mạng học sâu

Từ khái niệm mạng đa tầng thì chúng ta dẫn đến khái niệm mạng *hoc sâu* (deep learning - DL) (Schmidhuber, 2015). Nói một cách đơn giản thì mạng học sâu có thể được xem như một mạng đa tầng với số tầng là khá nhiều, trong đó số lượng perceptron ở mỗi tầng có thể khác nhau và mỗi tầng có thể sử dụng một hàm kích hoạt khác nhau. Điều này khác với mạng đa tầng truyền thống ban đầu chỉ có một hàm kích hoạt dùng chung cho toàn mạng.

Tham khảo Hình 3.2 là kiến trúc một mạng học sâu với 5 tầng có số lượng perceptron mỗi tầng khác nhau, và hàm kích hoạt được dùng cho các tầng cũng khác nhau (ReLU và sigmoid).



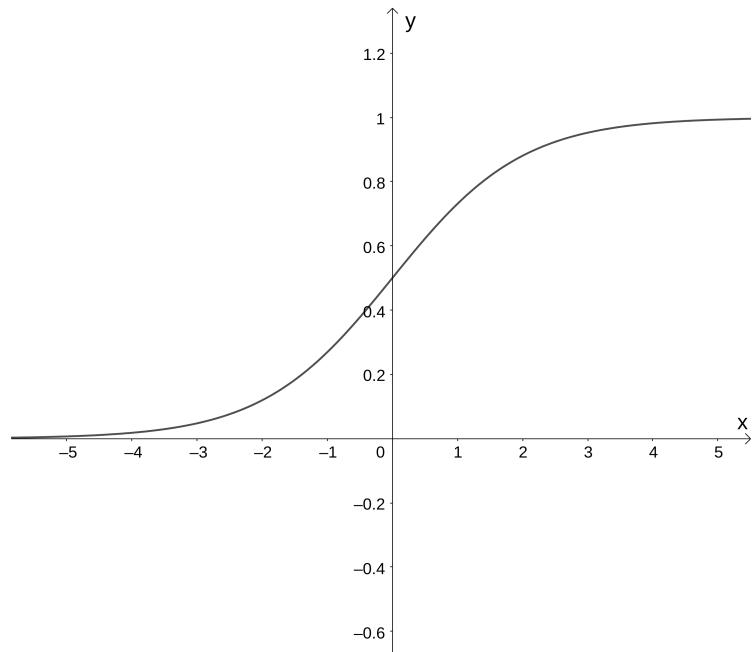
Hình 3.2: Kiến trúc mạng học sâu với 3 tầng ẩn

3.3 Các hàm kích hoạt (activation function)

3.3.1 Hàm sigmoid

Công thức:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (3.1)$$



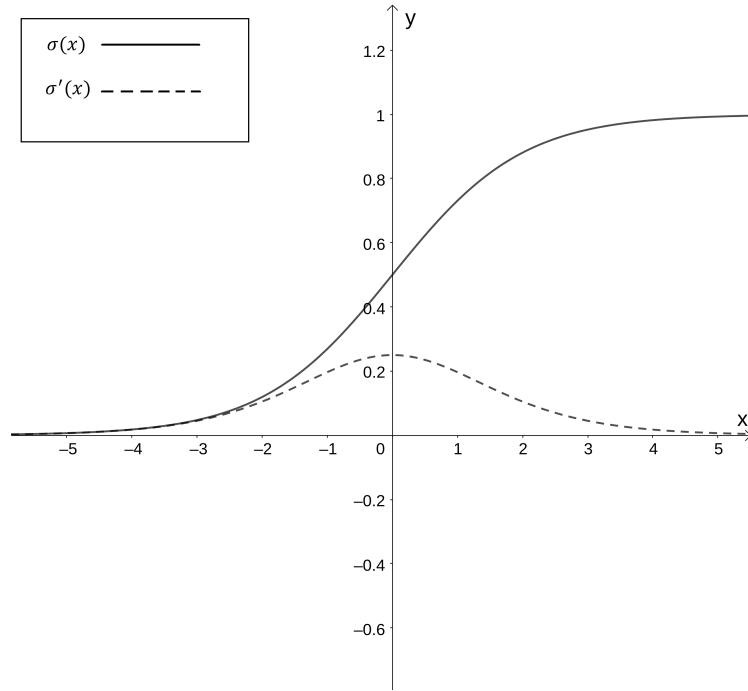
Hình 3.3: Đồ thị của hàm sigmoid

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 3.3 là đồ thị biểu diễn cho hàm sigmoid.

Như đã thấy ở phần 3.1, khi sử dụng hàm sigmoid như hàm kích hoạt, 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)

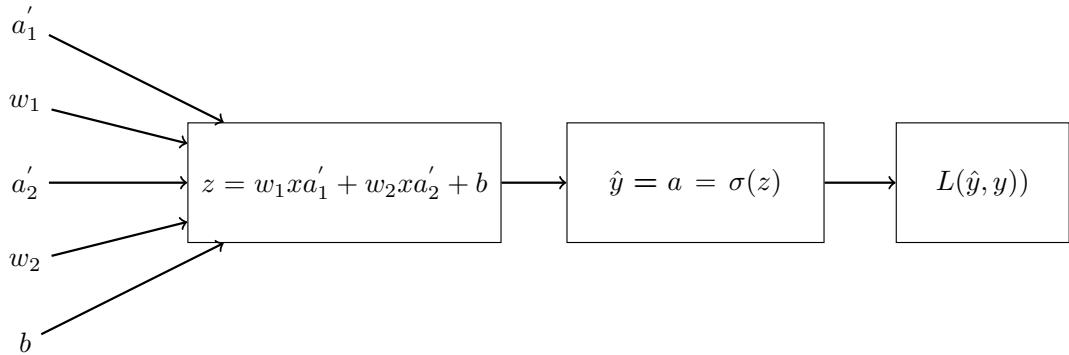


Hình 3.4: Đồ thị biểu diễn giá trị và đạo hàm của hàm sigmoid

Trên Hình 3.4, 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 3.5: Hoạt động của một tầng ẩn trong mạng MLP dùng hàm sigmoid

Ở ví dụ của mạng nơ-ron như Hình 3.5, đạo hàm riêng từng phần của hàm mất mát theo hai trọng số w_1 và w_2 sẽ được tính như sau:

$$\nabla_{w_1} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_1} = \frac{\partial L}{\partial z} a'_1 \quad (3.2)$$

$$\nabla_{w_2} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_2} = \frac{\partial L}{\partial z} a'_2 \quad (3.3)$$

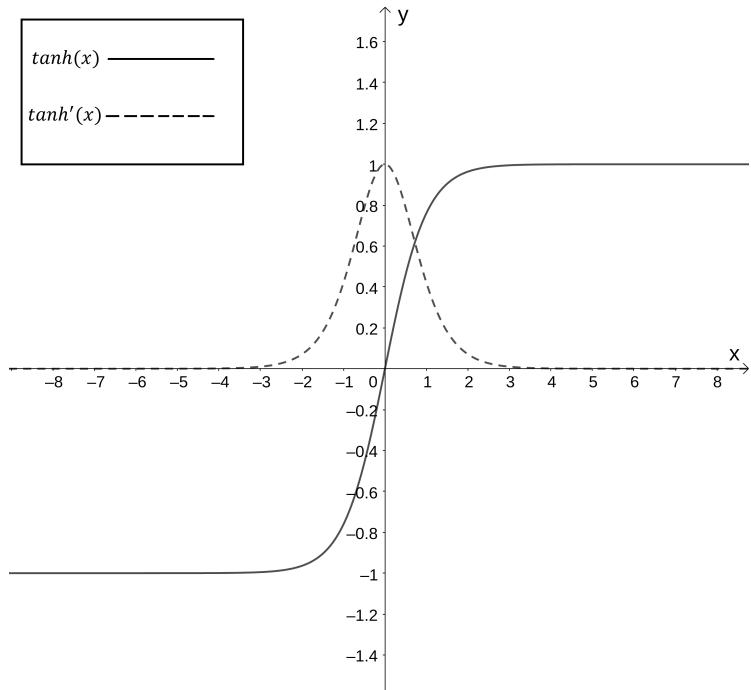
Vì a'_1 và a'_2 là kết quả của một hàm sigmoid trước đó, do đó luôn nhận giá trị dương và dấu của các gradient sẽ phụ thuộc vào $\frac{\partial L}{\partial z}$. Điều này có nghĩa là các gradient sẽ luôn cùng dương hoặc luôn cùng âm. Việc cập nhật trọng số sẽ chỉ xảy ra về một phía, hạn chế sự linh hoạt của mạng và gây khó khăn cho việc hội tụ.

3.3.2 Hàm tanh

Công thức:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3.4)$$

Hàm tanh nhận vào một số thực và trả về một giá trị trong khoảng (-1,1). Trên Hình 3.6, đường nét liền thể hiện giá trị của hàm tanh và đường nét đứt thể hiện cho giá trị đạo hàm. Dễ dàng nhận thấy rằng hàm tanh cũng gấp phải vấn đề triệt tiêu gradient như hàm sigmoid. Tuy nhiên, so với sigmoid, hàm tanh có tính chất zero-centered.



Hình 3.6: Đồ thị của hàm tanh và đồ thị đạo hàm của hàm tanh

Đặc biệt, hàm tanh cũng có thể được biểu diễn bằng hàm sigmoid như sau:

$$\tanh(x) = 2\sigma(2x) - 1 \quad (3.5)$$

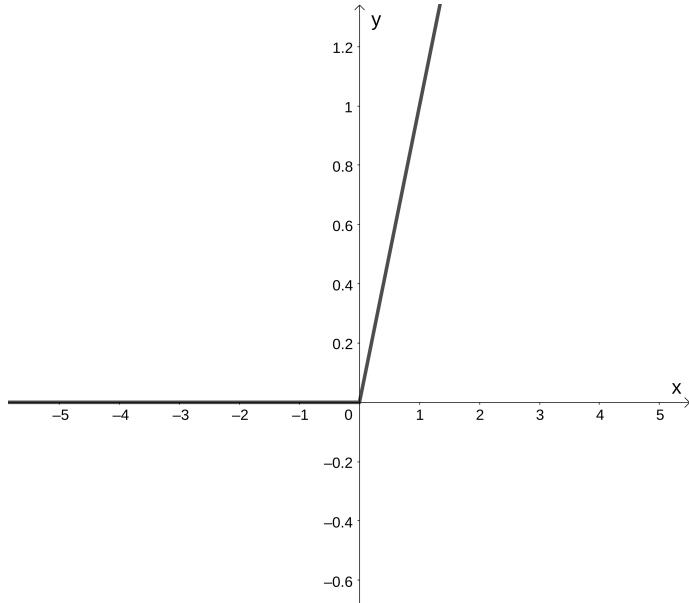
3.3.3 Hàm ReLU

Công thức:

$$f(x) = \max(0, x) \quad (3.6)$$

Quan sát công thức (3.6) và Hình 3.7, 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 sau:

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases} \quad (3.7)$$



Hình 3.7: Đồ thị biểu diễn hàm ReLU

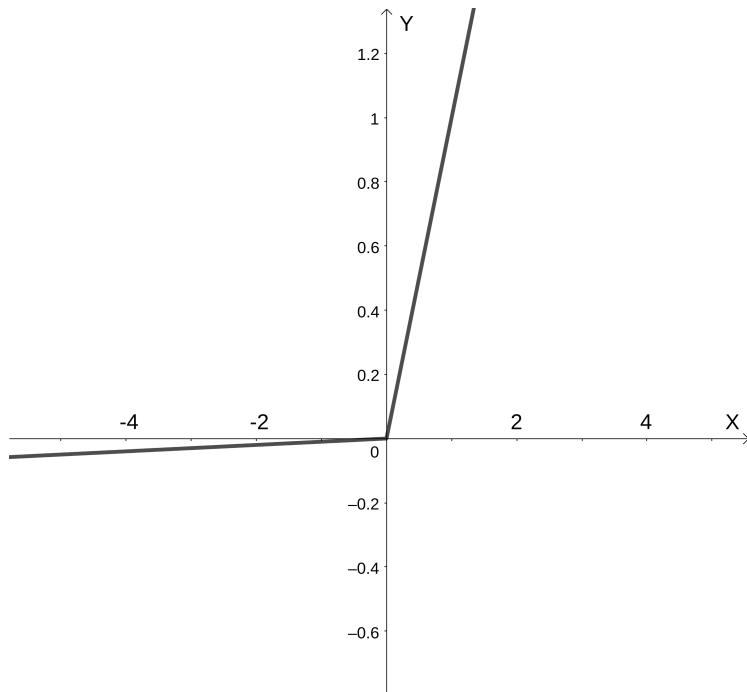
Như vậy, so với sigmoid và tanh, hàm ReLU sẽ không xuất hiện vấn đề triệt tiêu gradient. Tốc độ tính toán của hàm ReLU cũng sẽ nhanh hơn so với hai hàm trước đó. Tuy nhiên ReLU cũng tồn đọng một nhược điểm, với x có giá trị nhỏ hơn 0, qua hàm ReLU sẽ thu được kết quả bằng 0. Nếu giá trị của node bị chuyển 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* (Lu Lu and Karniadakis, 2019).

3.3.4 Hàm leaky ReLU

Công thức:

$$f(x) = \max(0.01x, x) \quad (3.8)$$

Leaky ReLU là một hàm được đề xuất để cố gắng xử lý hiện tượng 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 3.8.



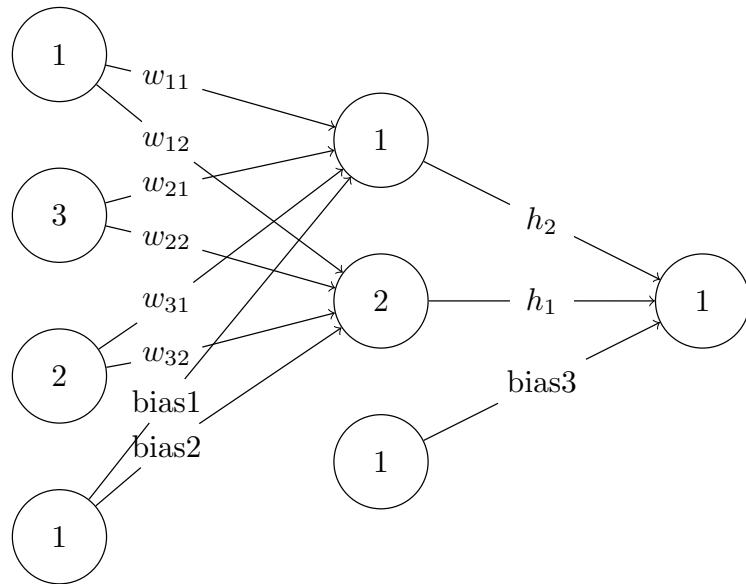
Hình 3.8: Đồ thị biểu diễn hàm Leaky ReLU

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.

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

3.4.1 Biểu diễn mạng nơ-ron

Xem xét mạng nơ-ron gồm 3 layer như sau:



Hình 3.9: Kiến trúc mạng nơ-ron 3 tầng

Theo Hình 3.9, với f là một hàm kích hoạt nào đó ta có:

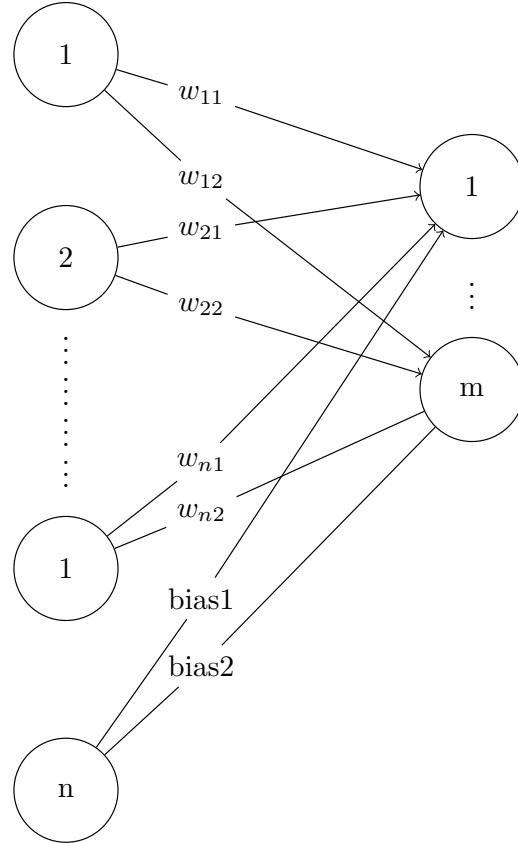
$$\begin{aligned} h_1 &= f(in_1 \times w_{11} + in_2 \times w_{21} + in_3 \times w_{31} + bias_1) \\ h_2 &= f(in_1 \times w_{12} + in_2 \times w_{22} + in_3 \times w_{32} + bias_2) \\ out &= f(h_1 \times w_{11} + h_2 \times w_{21} + bias_3) \end{aligned} \quad (3.9)$$

Với ý tưởng như vậy, ta có thể lưu các giá trị w_i vào một mảng hay một vector và xử lý tuân tự. Tuy nhiên, cần để ý rằng, quá trình tính toán ở mỗi perceptron sẽ được tính bằng cách lấy giá trị đầu ra của tầng trước đó hoặc là giá trị đầu vào nhân với trọng số tương ứng của nó, quá trình này sẽ được biểu diễn bằng kí hiệu toán học như sau :

$$h_i = f \left(\sum_{j=1}^{num_input} in_j \times w_{ji} + bias_i \right) \quad (3.10)$$

Trong đó, num_input chính là số lượng dữ liệu đầu vào được dùng để tính toán trong tầng hiện tại. Nếu 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 mỗi 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 3.10).



Hình 3.10: Kiến trúc của 2 tầng liên tiếp nhau trong một mạng nơ-ron đa tầng

Trong Hình 3.10, nếu biểu diễn dữ liệu đầu vào thành một vector hay nói cách khác là một ma trận X_{nx1} thì sẽ xây dựng được 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 (3.10) 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 hàm kích hoạt thì sẽ có kết quả của perceptron thứ i . Nếu lúc này xét h_{mx1} là ma trận của các perceptron đầu ra thì sẽ có công thức như sau:

$$h = f(W^T X + b) \quad (3.11)$$

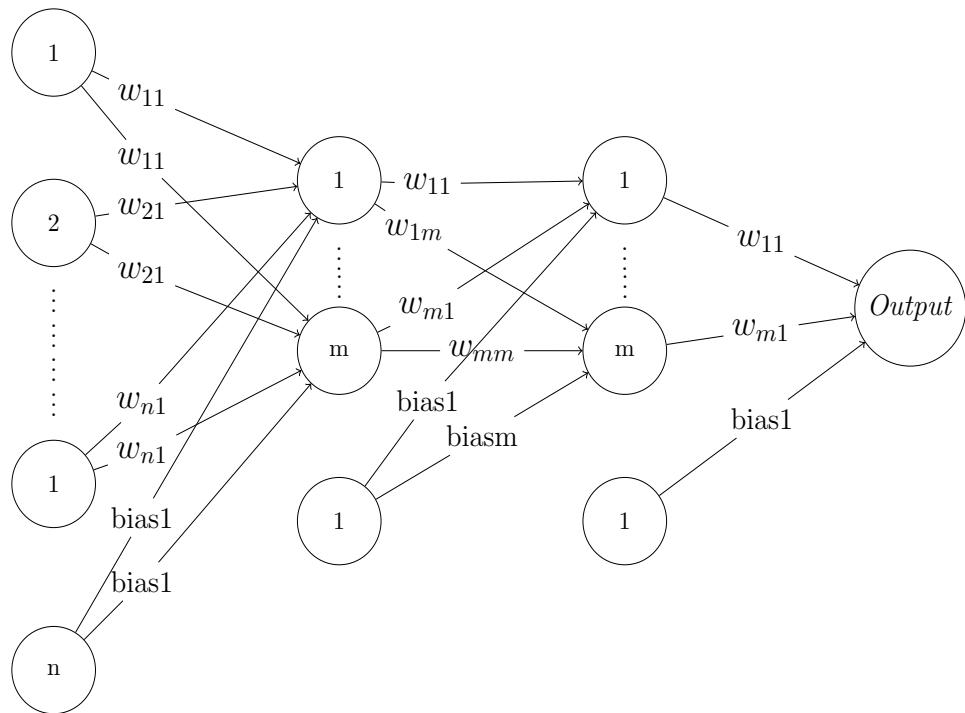
Cách biểu diễn mạng nơ-ron dưới dạng ma trận như thế này rất hữu ích trong việc

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 kích hoạt $f^{(i)}$, trong đó, output của tầng này sẽ là input của tầng tiếp theo.



Hình 3.11: Quá trình lan truyền xuôi.

Như Hình 3.11 là kiến trúc mạng gồm 4 tầng (1 input layer, 2 hidden layer và 1 output layer).

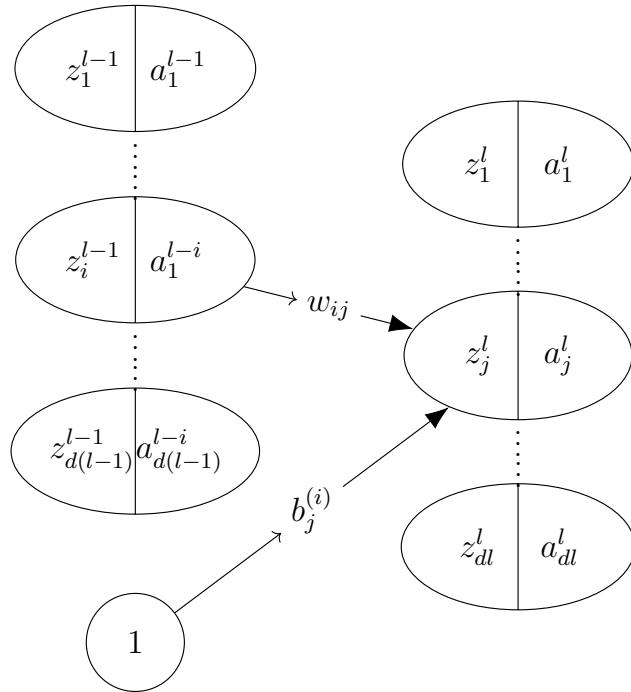
$$\begin{aligned} X^{(1)} &= f^{(1)}(W^{(1)T} \times Input + b^{(1)}) \\ X^{(2)} &= f^{(2)}(W^{(2)T} \times X^{(1)} + b^{(2)}) \\ Output &= f^{(3)}(W^{(3)T} \times X^{(2)} + b^{(3)}) \end{aligned} \quad (3.12)$$

Với cách tính (3.12) thì hoàn toàn có thể mở rộng một cách tổng quát với 1 input layer, n hidden layer và 1 output layer như sau:

$$\begin{aligned} X^{(1)} &= f^{(1)}(W^{(1)T} \times Input + b^{(1)}) \\ X^{(i)} &= f^{(i)}(W^{(i)T} \times X^{(i-1)} + b^{(i)}), i = \overline{2, n} \\ Output &= f^{(n)}(W^{(n)T} \times X^{(n)} + b^{(n+1)}) \end{aligned} \quad (3.13)$$

Lan truyền ngược

Lan truyền ngược (backward propagation hay back propagation) là quá trình cập nhật lại các trọng số để tìm ra bộ trọng số phù hợp sẽ được gọi là mô hình cho bài toán cần giải quyết. Khác với lan truyền xuôi khi sử dụng hàm kích hoạt, lan truyền ngược dựa trên việc tính đạo hàm để cho ra kết quả. Để cập nhật được trọng số thì bắt buộc cần phải có hàm mất mát J .



Hình 3.12: Quá trình lan truyền ngược giữa 2 tầng trong mạng nơ-ron

Nhằm dễ dàng hơn trong các bước tính toán tiếp theo nên nhắc lại những biểu thức sau:

$$z^{(i)} = W^{(i)T} \times X^{(i)} + b^{(i)} \quad (3.14)$$

$$a^{(i)} = f(z^{(i)}) \quad (3.15)$$

Các biểu thức (3.14) và (3.15) được biểu diễn như Hình 3.12. Nhằm đỡ rối khi nhìn nên chỉ vẽ tương trưng các đường nối cần thiết, thực tế sẽ mỗi perceptron ở tầng trước phải nối hết với các perceptron tầng sau.

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

$$\frac{\partial J}{\partial w_{ij}^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}} = e_j^{(l)} \times a_i^{(l-1)} \quad (3.16)$$

với

$$\begin{aligned} e_j^{(l)} &= \frac{\partial J}{\partial z_j^{(l)}} = \frac{\partial J}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \\ &= \left(\sum_{k=1}^{d^{(l+1)}} \frac{\partial J}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} \right) f'(z_j^{(l)}) \\ &= (w_j^{(l+1)} \times e^{(l+1)}) f'(z_j^{(l)}) \end{aligned} \quad (3.17)$$

Trong đó $e^{(l+1)} = [e_1^{(l+1)}, e_2^{(l+1)}, \dots, e_{d^{(l+1)}}^{(l+1)}]^T$ và $w_j^{(l+1)}$ là toàn bộ phần tử hàng thứ j của ma trận $W^{(l+1)}$. Nhờ xuất hiện trong công thức (3.17) thể hiện việc $a_j^{(l)}$ đóng góp vào việc tính các $z_k^{(l+1)}$.

Với cách làm tương tự, ta có:

$$\frac{\partial J}{\partial b_j^{(l)}} = e_j^{(l)} \quad (3.18)$$

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

$$\frac{\partial J}{\partial W^{(l)}} = a^{(l-1)} e^{(l)T}$$

$$e^{(l-1)} = W^{(l)T} e^{(l)} f'(z^{(l)}) \quad (3.19)$$

$$\frac{\partial J}{\partial b^{(l)}} = e^{(l)}$$

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 nơ-ron theo cách lập trình câu lệnh truyền thống sử dụng vòng lặp (imperative) mà không dùng đến ma trận với ngôn ngữ minh họa là Python.

```

1      J = 0
2      for i = 1 to m:
3          z[i] = b
4          for j in range(len(w)):
5              z[i] += w[j]*x[i][j]
6          a[i] = Sigmoid(z[i])
7          J += -[y[i]*log(a[i]) + (1-y[i])*log(1-y[i])]
8          dz[i] = a[i] - y[i]
9          for j in range(len(w)):
10             dw[j] += x[i][j]*dz[i]
11      J = J / m
12      for j in range(len(w)):
13          w[j] = w[j] - alpha*dw[j]/m

```

Code Listing 3.1: Lập trình câu lệnh truyền thống để tính toán trong mạng nơ-ron

Tiếp theo, hãy thử hiện thực các phép tính toán bằng cách sử dụng ma trận và các phép tính dành cho ma trận, cũng với ngôn ngữ Python như sau:

```

1  import numpy as np
2
3  Z = np.dot(w.T,X) + b
4  A = Sigmoid(Z) #1
5  dZ = A - Y
6  dW = (1/m) * X * dZ.T

```

```

7      dB = (1/m) * np.sum(dZ)
8      W = W - alpha * dW
9      b = b - alpha * dB

```

Code Listing 3.2: Lập trình sử dụng ma trận để tính toán trong mạng nơ-ron

Dễ dàng thấy rằng, về mặt biểu diễn thì cách tính toán trong Code Listing 3.2 gọn gàng hơn và dễ hiểu hơn đối với những người có nhiều kiến thức về toán học nhưng chưa có nhiều kiến thức về lập trình. Về mặt hiện thực, cách tính toán bằng ma trận trong Code Listing 3.2 cho phép thực hiện song song trên những máy tính có kiến trúc hỗ trợ đa luồng.

Điều này cũng cho thấy rằng vì sao các kiến trúc mạng nơ-ron học sâu thì thường yêu cầu các perceptron trong cùng một tầng phải cùng sử dụng một hàm kích hoạt nhằm hỗ trợ việc tính toán song song.

3.4.4 Kỷ nguyên mới của mạng nơ-ron

Các chương vừa trình bày cho thấy rằng mạng nơ-ron có một tiềm năng tính toán rất lớn để giải quyết nhiều bài toán có tính thực tế cao. Tuy nhiên, có một thời gian dài mạng nơ-ron không được sử dụng hay còn gọi là chìm đắm trong “màn đêm”.

Dể giải thích việc này có thể quay lại việc lan truyền xuôi và lan truyền ngược sử dụng ma trận. Về mặt lý thuyết thì việc được lan truyền qua quá nhiều tầng sẽ khiến bộ trọng số bị suy biến và các tầng sau sẽ không còn học được nhiều nữa. Về mặt hiện thực, mạng nơ-ron nhiều tầng sử dụng các ma trận có kích thước rất lớn. Điều này đòi hỏi nhiều tài nguyên tính toán khiến cho những kiến trúc máy tính thế hệ trước không đủ khả năng xử lý và tính toán trên những ma trận lớn như vậy.

Hai vấn đề trên đã dần được giải quyết nhờ vào các bước tiến vượt bậc. Về mặt lý thuyết thì đã xuất hiện những hàm kích hoạt giúp giảm thiểu việc suy biến dữ liệu như đã giới thiệu trong Mục 3.3. Ngoài ra, về mặt phần cứng, sự xuất hiện của những mạng hỗ trợ đồ họa đã giúp cho việc tính toán trên ma trận có kích thước lớn rất nhanh và hiệu quả. Tiêu biểu là sự ra đời của GPU (Graphics Processing Unit).

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

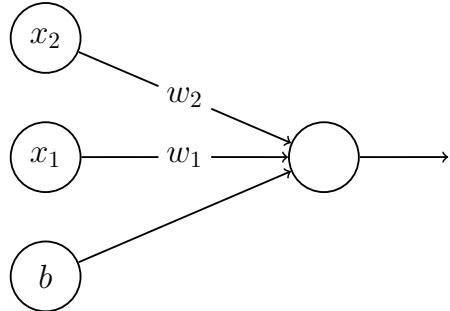
các *điểm ảnh* (pixel), vì vậy kiến trúc nội tại của GPU được xây dựng để hỗ trợ tối đa các phép tính toán trên ma trận để có thể xử lý hình ảnh tốt nhất. Thêm vào đó, GPU ngày nay lại được xử lý đa luồng và tiếp nhận thông tin gấp nhiều lần CPU để hỗ trợ đồ họa tạo hiệu ứng tối đa. Đây cũng là lý do 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 một hàm kích hoạt 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

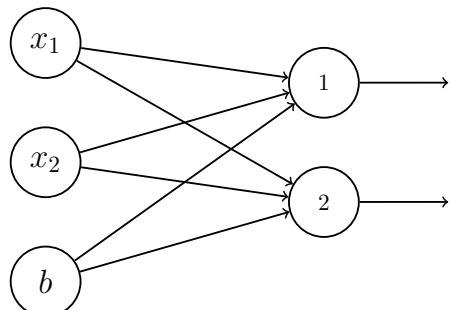
Bài tập 3.6.1. Cho mạng nơ-ron như hình vẽ sau:



Hình 3.13: Kiến trúc mạng nơ-ron của Bài tập 3.6.1

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 $\alpha = 0.1$ và hàm mất mát là *Cross Entropy*) của mạng khi hàm kích hoạt là hàm *sigmoid*.

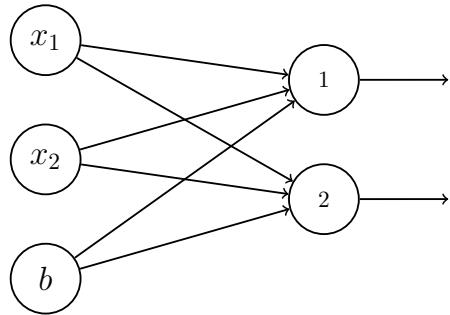
Bài tập 3.6.2. Cho mạng nơ-ron như hình vẽ sau:



Hình 3.14: Kiến trúc mạng nơ-ron của Bài tập 3.6.2

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 nơ-ron như hình vẽ sau:



Hình 3.15: Kiến trúc mạng nơ-ron của Bài tập 3.6.3

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 $\alpha = 0.1$ và hàm mất mát là *Cross Entropy*) của mạng với các kích hoạt sau:

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

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 ngành vì nó liên quan đến việc phải hiểu ý nghĩa ngôn ngữ - một công cụ hoàn hảo của con người dùng để 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). 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, từ trên có nghĩa là thay đổi ngôn ngữ của con người thành các biểu diễn mà máy tính có thể hiểu và xử lý. Tuy nhiên, xử lý ngôn ngữ tự nhiên không phải chỉ là chuyển các ký tự trong bảng chữ cái thành chuỗi các bit mà thành các dạng biểu diễn phức tạp hơn tùy theo phương pháp và mục đích ứng dụng. Các ứng dụng nổi bật của xử lý ngôn ngữ tự nhiên gồm có:

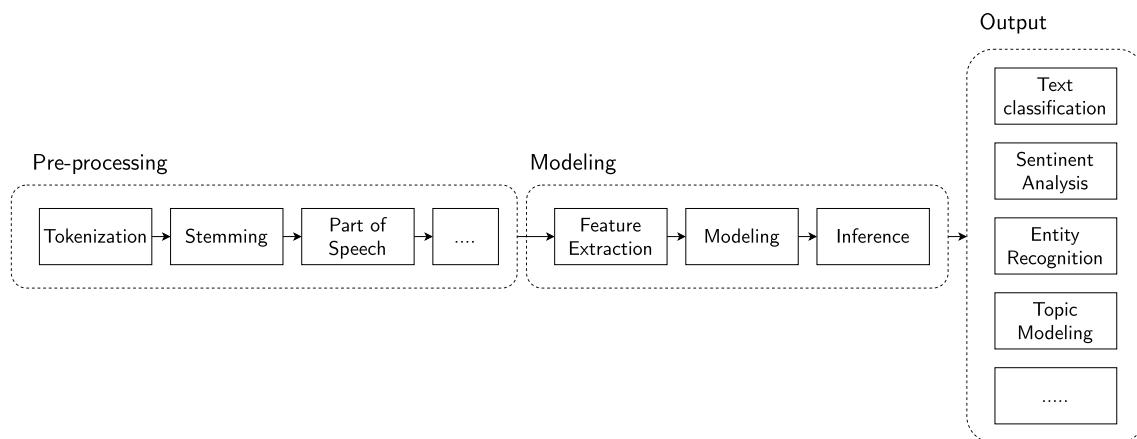
- *Phân loại văn bản* (Text Classification - TC) (Soergel, 1985): Đây có thể được coi là một trong những ứng dụng phổ biến nhất của xử lý ngôn ngữ tự nhiên. Trong phân loại văn bản, các từ (nghĩa, mối quan hệ các từ, ngữ cảnh) được

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.

- *Phân tích cảm xúc* (Sentiment Analysis - SA) (Stone et al., 1966): Ứng dụng này hướng đến phân tích cảm xúc tác giả của một đoạn văn bản. Đây có thể coi là một bài toán riêng của bài toán phân loại văn bản, trong đó 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. Trong thực tế, 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.
- *Nhận diện chủ thể* (Entity Recognition - ER) (Marsh and Perzanowski, 1998): Đây là bài toán 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 đối tượng cụ thể. Mỗi chủ thể sẽ được xác định và đưa vào các mục (thể loại) định trước. Ví dụ như "Đại học Bách Khoa TPHCM" có thể được tự động xác định trong một văn bản và được phân loại vào mục "Trường Đại học".
- *Mô hình chủ đề* (Topic Modeling - TM) (Blei, 2012): Thuật ngữ "chủ đề" có nghĩa là tập các từ "đi cùng với nhau". Những từ này sẽ giúp người đọc hình dung ra một chủ đề cụ thể. Ví dụ như tập các từ *bóng đá, sân vận động, chạy bộ, bơi lội,...* sẽ giúp người đọc liên tưởng đến chủ đề "Thể thao". Nhiệm vụ của ứng dụng này là tự động phát hiện ra các chủ đề có thể có trong một văn bản bằng cách xây dựng các tập từ có liên quan như vậy. Để làm được điều này, hệ thống Mô hình chủ đề cần phải được huấn luyện từ một tập corpus rất lớn để tính toán được xác suất xuất hiện cùng nhau của các chủ đề.
- *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. Hiện nay vẫn tồn tại thách thức đối với dịch máy bao gồm như: sự đa dạng về ngôn ngữ, bảng chữ cái và ngữ pháp. Khi đã có một bản dịch máy tự động, chúng ta vẫn có thể gặp khó khăn khi xác định bản dịch có chính xác hay không.

- *Trả lời tin nhắn tự động* (Chatbot) (Bradeško and Mladenić, 2012): Một hệ thống trả lời tin nhắn tự động có thể hiểu được câu hỏi của khách hàng với những từ ngữ tự nhiên mà không cần sự hỗ trợ từ con người. Hiện nay ứng dụng này đang thu hút nhiều sự chú ý và đang được ứng dụng trong thực tế ở nhiều lĩnh vực khác nhau.
- *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.

4.1.1 Mô hình xử lý ngôn ngữ tự nhiên cổ điển



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

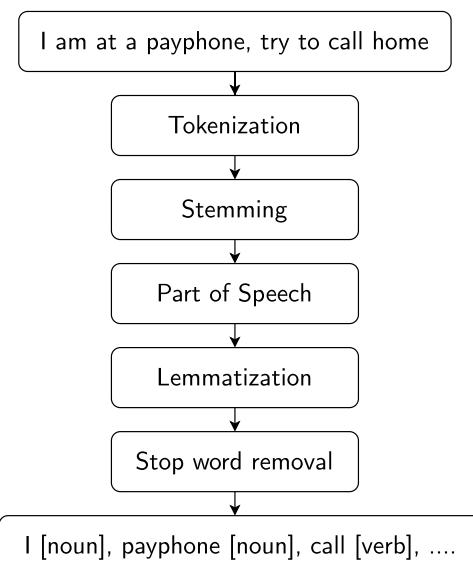
Theo phương pháp cổ điển, các tác vụ xử lý ngôn ngữ tự nhiên gồm có 2 bước chính như mô tả trong Hình 4.1:

- *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)*: Dựa ra một mô hình học máy để thực hiện tác vụ mong muốn.

4.1.2 Tiền xử lý

Tiền xử lý (Preprocessing) 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 một 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 dữ liệu phổ biến

Các bước tiền xử lý thường tùy thuộc vào từng dự án và mục đích. Hình 4.2 liệt kê các bước thông dụng với mô tả chi tiết như sau:

- *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). Dõi 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 hai token.

- *Biến đổi từ về dạng gốc (Stemming)*: đây là kỹ thuật dùng để biến đổi một từ về dạng gốc (được gọi là stem hoặc root form) bằng cách cực kỳ đơn giản là loại bỏ một 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.
- *Tù vựng hóa (Lemmatization)*: Làm trở lại nguyên dạng ban đầu các từ vựng bị biến đổi thể (inflection) hoặc được kết hợp (conjugation). 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 các bước tiền xử lý cần thiết sẽ tiến hành thực hiện bước mô hình hóa. Mô hình hóa gồm các bước như sau để thu được kết quả.

- *Rút trích đặc 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. 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 đặc trưng 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 tất cả các từ trong văn bản thành các token và ghi nhận lại sự xuất hiện của các token này trong từng văn bản.

Ví dụ 4.2.1. Hãy trích xuất đặc trưng 3 câu sau bằng phương pháp túi từ.

1. "Hôm qua tôi học lập trình"

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

Trả lời. Thông qua bước tokenization có thể 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".

Cho rằng mỗi câu là một văn bản riêng biệt và xây dựng một danh sách tất cả các từ từ 3 câu trên. Kết quả sẽ nhận được các từ như sau:

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

Bước tiếp theo là 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]

Thực hiện tương tự với các câu còn lại thì nhận được các 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
1	1	0	0	1	0	0	1	1
2	0	1	0	1	1	0	1	1
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 thì có thể 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) (Leskovec et al., 2014).

Ưu điểm của TF-IDF so với phương pháp túi từ là có khả năng biểu diễn từ với các trọng số khác nhau dựa vào độ quan trọng của từ đó trong câu.

Nhận xét câu 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 *beautiful* 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 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 suất 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).

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

IDF - Inverse Document Frequency: dùng để ước lượng mức độ quan trọng của một từ. 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 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,...

Đó là lí do cần phải giảm đi mức độ quan trọng của những từ đó bằng cách sử dụng IDF bằng công thức sau:

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

Từ đó đúc kết được công thức TF-IDF là:

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

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

Ví dụ 4.2.2. Hãy dùng phương pháp TF-IDF để phân tích ba văn bản 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"

Trả lời. Bước đầu tiên là xây dựng bảng từ vựng trong kho ngữ liệu như Bảng 4.2.

Bảng 4.2: Bảng từ vựng trong kho ngữ liệu của Ví dụ 4.2.2

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

Tiếp theo, cần tính TF cho mỗi từ ứng với mỗi văn bản đã cho như Bảng 4.3.

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

Sau đó là bước tính giá trị IDF cho mỗi từ vựng và liệt kê kết quả như Bảng 4.4.

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, dựa vào giá trị TF và IDF 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
Doc 1	0.12	0	0	0	0	0	0	0
Doc 2	0	0.1	0	0	0.1	0	0	0
Doc 3	0	0	0.1	0	0	0.1	0	0

Sau khi tính TF-IDF, kết quả nhận được cho mỗi document đã được vector hóa bằng phương pháp này là:

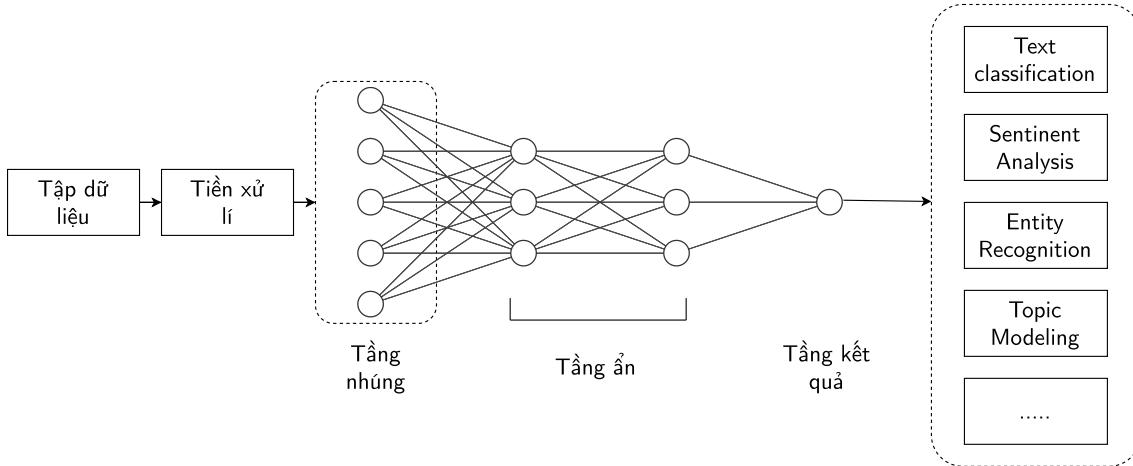
- "Hôm_qua tôi học lập_trình" $\Rightarrow [0.12, 0, 0, 0, 0, 0, 0, 0]$
- "Hôm_nay tôi cũng học lập_trình" $\Rightarrow [0, 0.1, 0, 0, 0.1, 0, 0, 0]$
- "Ngày_mai tôi không học lập_trình" $[0, 0, 0.1, 0, 0, 0.1, 0, 0]$

Số chiều của vector chính là tổng số từ vựng có được trong tất cả các document, trong ví dụ này thì các vector có số chiều bằng 8.

Thông qua Ví dụ 4.2.2, có thể nhận ra một hạn chế của phương pháp TF-IDF là không biểu diễn được quan hệ giữa các từ trong văn bản. Ví dụ như khi biểu diễn câu "Lập_trình tôi học hôm_qua" 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. Hiện nay đã có rất nhiều mô hình học sâu cùng các kỹ 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 lý ngôn ngữ tự nhiên có thể được mô tả như Hình 4.3.



Hình 4.3: Mô hình học sâu được sử dụng cho bài toán NLP

Từ Hình 4.3 có thể thấy, để giải quyết một bài toán xử lí ngôn ngữ tự nhiên bằng kiến trúc mạng học sâu thì cần thực hiện 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 dừng...
- 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 áp dụng mô hình trên vào một bài toán nhất định, có thể thấy một cách rất tự nhiên rằng sau khi 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 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 4.2.2 đã nêu ra, sẽ có một tập từ điển có 8 từ được sắp xếp theo bảng chữ cái. Tức là, mỗi tài liệu ứng với một vector 1×8 (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. Cụ thể, 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 mang học sâu nên vẫn có cách biểu diễn để giúp cho tập tài liệu thể hiện nhiều thông tin hơn. Cụ thể hơn là thay vì biểu diễn cả tài liệu chỉ thông qua một vector thì sẽ biểu diễn từng từ trong tài liệu thành từng vector. Điều này có nghĩa rằng, với cách tiếp cận này, sau khi áp dụng cho tập tài liệu "Hôm_qua tôi đi học" thì sẽ tạo ra được một ma trận 4×8 (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, 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 kĩ 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 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 kĩ 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.

Kĩ 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.

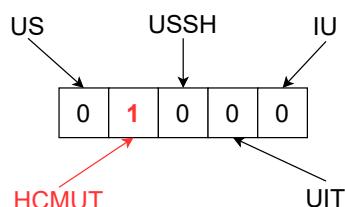
Từ khi khái niệm nhúng từ ra đời, đã có rất nhiều giải thuật được trình bày như word2vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014), ELMO (Peters et al., 2018), BERT (Devlin et al., 2019), v.v. Tuy nhiên một giải thuật nhúng từ cần phải thoả mãn 2 yêu cầu dưới đây:

- 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, kỹ thuật được sử dụng phổ biến để số hoá một từ là sử dụng *one-hot vector*. Kỹ thuật này được áp dụng trong các mô hình học máy cổ điển trong giai đoạn tiền xử lý, nhằm biến đổi những dữ liệu 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 kỹ thuật biểu diễn một tập các giá trị rời rạc sang một tập one-hot vector. Để hiểu rõ hơn, chúng ta xem xét ví dụ sau đây. Xét tập hợp một số trường Đại học ở Việt Nam bao gồm Đại học Bác Khoa (HCMUT), Đại học Khoa học Tự nhiên (US), Đại học Khoa học Xã hội và Nhân văn (USSH), Đại học Quốc tế (IU), Đại học Công nghệ Thông tin (UIT). Sử dụng kỹ thuật one-hot vector, chúng ta có thể biểu diễn giá trị HCMUT bằng một vector trong Hình 4.4.



Hình 4.4: Cách biểu diễn HCMUT bằng one-hot vector.

Vector này có 5 chiều tương đương với 5 trường đại học trong tập hợp đã cho. Trong đó tại chiều thứ 2, tương ứng với vị trí của HCMUT trong danh sách các trường đại học, mang giá trị 1, những chiều còn lại mang giá trị không. Tương tự, có thể biểu diễn các trường đại học khác bằng các vector one-hot được trình bày trong Bảng 4.6.

Bảng 4.6: Sử dụng one-hot vector để biểu diễn một số trường đại học ở Việt Nam

	US	HCMUT	USSH	UIT	IU
US	1	0	0	0	0
HCMUT	0	1	0	0	0
USSH	0	0	1	0	0
UIT	0	0	0	1	0
IU	0	0	0	0	1

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.

Ví dụ 4.4.1. Cho 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

Hãy biểu diễn các từ trong các câu trên bằng kĩ thuật one-hot encoding.

Trả lời. Tập từ điển có tất cả 8 từ: "Hôm_qua", "Hôm_nay", "Ngày_mai", "Tôi", "Học", "Lập_trình" "Cũng", "Không".

Sử dụng one-hot vector có 8 (tổng số từ có trong từ điển) chiều để biểu diễn các từ trên sang dạng số. Mở rộng ra, khi từ điển có n từ thì 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 mang giá trị 1. Bảng 4.7 thể hiện việc chuyển đổi từ sang one-hot vector.

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

	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

Kỹ thuật one-hot encoding 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 kỹ thuật này và mang lại hiệu quả khả quan, đặc biệt khi số lượng từ trong từ điển tương đối nhỏ. Tuy kỹ 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 (overfitting) xảy ra khi số lượng từ trong từ điển tăng: Với kỹ thuật này, mỗi khi 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 tổng số lượng từ có trong từ điển. Trong Ví dụ 4.4.1, 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 quát 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 quát 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, ta luôn muốn mô hình có khả năng tổng quát hoá 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 lẻ, 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

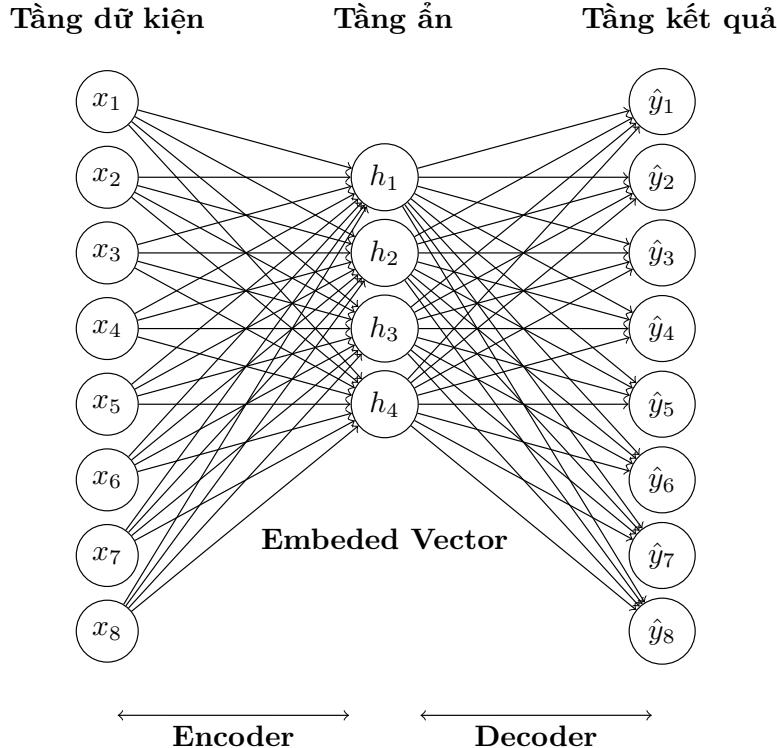
Ý tưởng về auto-encoder đã được các nhà nghiên cứu lối lạc về mạng nơ-ron đề xuất từ cuối thế kỷ 20, đại diện là LeCun (LeCun et al., 1989), Hinton và Zemel (Hinton and Zemel, 1994), với ý tưởng là mô hình mạng nơ-ron được huấn luyện dựa trên cơ chế học không giám sát, nhằm biểu diễn dữ liệu đầu vào ở dạng *vector nhiều chiều* (high-dimensional 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 hóa 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 đích 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.



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

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.

- *Decoder*: ngược 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-Encoder trong Hình 4.5, được khởi tạo lần lượt như sau:

$$\begin{array}{l}
 \text{Ma trận Encoder } 8 \times 4: \\
 \left[\begin{array}{cccc} 3 & 2 & 1 & 5 \\ 4 & 5 & 7 & 8 \\ 2 & 1 & 2 & 4 \\ 4 & 2 & 1 & 2 \\ 1 & 3 & 1 & 3 \\ 2 & 4 & 2 & 1 \\ 4 & 5 & 2 & 7 \\ 6 & 4 & 4 & 5 \end{array} \right] \\
 \\
 \text{Ma trận Decoder } 4 \times 8: \\
 \left[\begin{array}{cccccccc} 4 & 5 & 1 & 2 & 5 & 1 & 5 & 6 \\ 5 & 6 & 8 & 1 & 9 & 0 & 2 & 4 \\ 3 & 2 & 1 & 4 & 2 & 5 & 7 & 9 \\ 7 & 5 & 7 & 9 & 2 & 1 & 3 & 1 \end{array} \right]
 \end{array}$$

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 vị 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:

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

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:

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

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:

Üng 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. Vì 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 4.4.2. 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: $\{cũng, học, hôm_nay, hôm_qua, không, lập_trình, ngày_mai, tôi\}$.

Bảng 4.8: Từ mục tiêu và từ ngữ cảnh tương ứng của D1 có kích thước cửa sổ là 1

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

Nếu chọn kích thước cửa sổ là 1 thì sẽ có được thông tin ngữ cảnh của các từ trong tài liệu D1 như Bảng 4.8. Hoặc nếu chọn kích thước cửa sổ là 2 thì Bảng 4.9 đã liệt kê đầy đủ những từ mục tiêu và từ ngữ cảnh tương ứng trích xuất từ tài liệu D1.

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

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

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 thì 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)*.

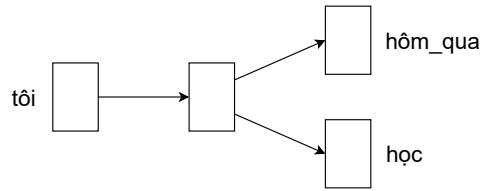
Skip-gram

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 4.4.2. Để huấn luyện mô hình skip-gram, đầu tiên dữ liệu đầu vào được đưa 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.

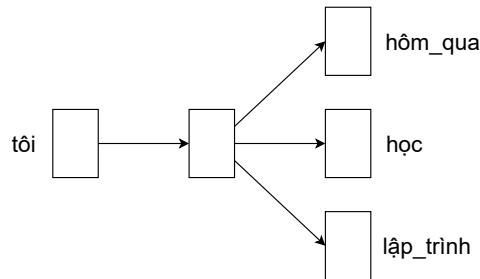
Lần lượt các Hình 4.6 và Hình 4.7 là sơ đồ huấn luyện mô hình skip-gram tương ứng với kích thước cửa sổ là 1 hoặc là 2.

Tầng dữ kiện Tầng ẩn Tầng kết quả



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

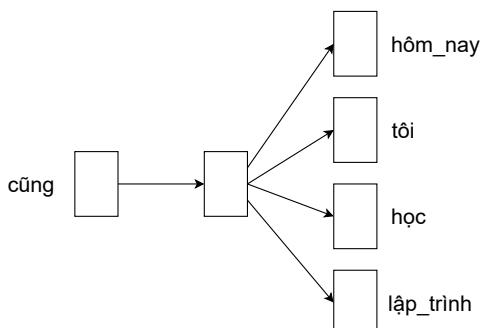
Tầng dữ kiện Tầng ẩn Tầng kết quả



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

Để ý rằng 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 chỉ có 3 cặp từ. Trong trường hợp từ mục tiêu có đầy đủ từ ngữ cảnh, sẽ có 4 cặp từ để huấn luyện (gồm 2 từ ngữ cảnh bên trái và 2 từ ngữ cảnh bên phải), minh họa bằng tài liệu D2 như Hình 4.8.

Tầng dữ kiện Tầng ẩn Tầng kết quả

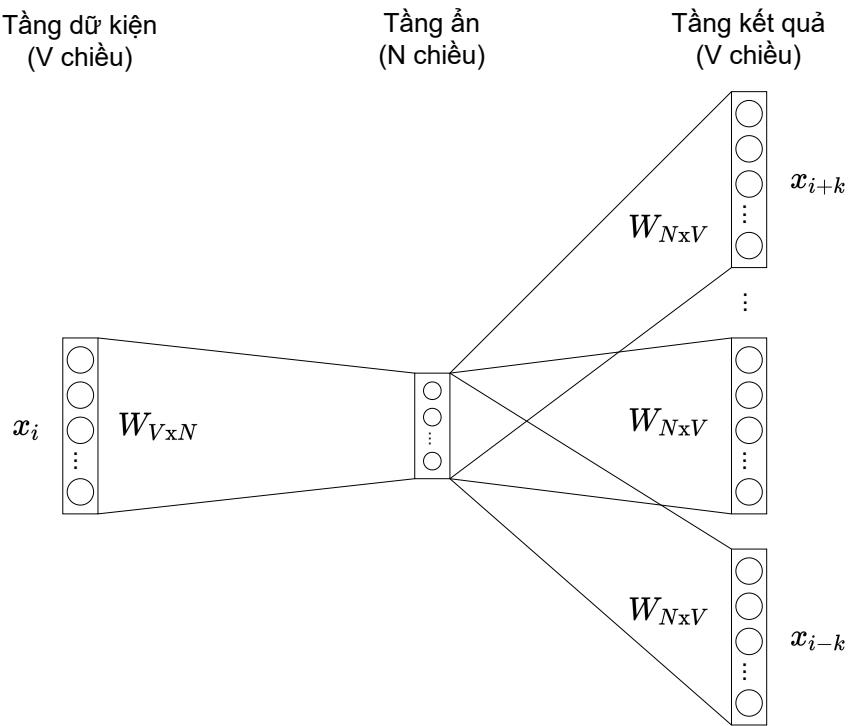


Hình 4.8: Sơ đồ huấn luyện mô hình skip-gram khi kích thước cửa sổ là 2 và sử dụng từ mục tiêu trong tài liệu D2

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 tầng kết quả. Để đơn giản ta xét kho ngữ liệu chỉ bao gồm 3 tài liệu D1, D2, D3 như ở Ví dụ 4.4.2. Khi đó, mạng nơ-ron sẽ có dạng như Hình 4.9.



Hình 4.9: Mô hình skip-gram dạng tổng quát

Dễ dàng nhận ra rằng, mô hình này 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.

Tầng dữ kiện

Tương tự như kỹ thuật Auto-Encoder, một từ cũng được biểu diễn thành một vector one-hot để làm đầu vào của mạng nơ-ron. Kho ngữ liệu gồm D1, D2, D3 trong Ví

dụ 4.4.2 có 8 *từ phân biệt* (*unique word*) nên mỗi từ sẽ được biến bằng một vector one-hot có 8 chiều, trong đó chỉ có chiều ở vị trí tương ứng với vị trí của từ trong 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. Áp dụng cách biểu diễn như vậy sẽ thu được các vector one-hot sau:

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

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

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

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

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

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

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

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

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 dữ kiện 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*), có thể điều chỉnh giá trị này để đạt được kết quả phù hợp tùy theo từng bài toán khác nhau. Sau khi trải qua quá trình huấn luyện, ma trận trọng số của mô hình có thể được biểu diễn như Bảng 4.10.

Bảng 4.10: Ma trận trọng số của tầng ẩn sau khi huấn luyện xong

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

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 thì 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.

Bảng 4.11: Ma trận trọng số của tầng kết quả sau khi huấn luyện xong

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

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:

$$c\uacute{u}ng = \begin{bmatrix} 0.3 & 0.5 & 0.2 & 0.01 \end{bmatrix}$$

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

$$l\uacute{a}p_trình = \begin{bmatrix} 2.5 & 1.1 & 0.3 & 0.55 \end{bmatrix}$$

$$h\uacute{o}m_nay = \begin{bmatrix} 1 & 0.15 & 0.6 & 2.1 \end{bmatrix}$$

$$h\uacute{o}m_qua = \begin{bmatrix} 1.3 & 1.7 & 2.1 & 0.24 \end{bmatrix}$$

$$kh\uacute{o}ng = \begin{bmatrix} 2.2 & 1.8 & 0.6 & 0.3 \end{bmatrix}$$

$$ng\uacute{a}y_mai = \begin{bmatrix} 1.4 & 0.12 & 0.04 & 2.4 \end{bmatrix}$$

$$t\uacute{o}i = \begin{bmatrix} 1.12 & 0.8 & 0.5 & 1.25 \end{bmatrix}$$

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

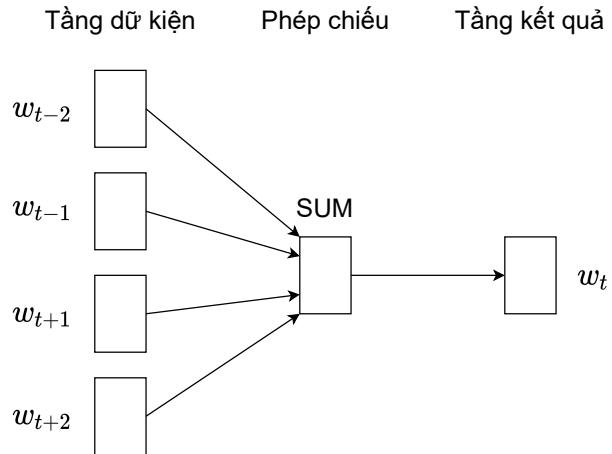
Tổng kết mô hình skip-gram

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)

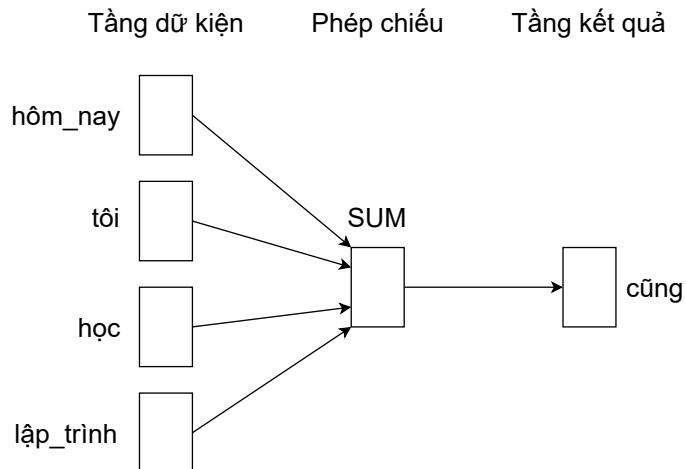
CBOW có đầu vào là từ mục tiêu, đầu ra là các từ trong ngữ cảnh. Ý tưởng chính của mô hình CBOW là dự đoán từ mục tiêu dựa vào các từ ngữ cảnh xung quanh

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 ($w_{t-m}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+m}$) xung quanh từ w_t trong phạm vi m .



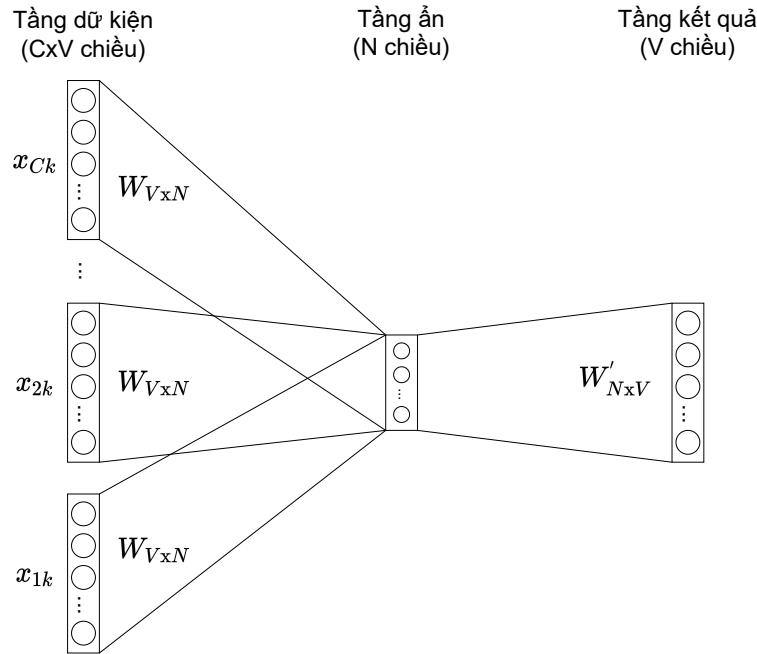
Hình 4.10: Sơ đồ minh họa mô hình CBOW

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



Hình 4.11: Sơ đồ minh họa mô hình CBOW áp dụng cho Ví dụ trên

Mô hình CBOW tổng quát được minh họa 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.



Hình 4.12: Mô hình CBOW dạng tổng quát

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

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

Với mỗi vector x^k chỉ có một phần tử x_k có giá trị 1, các phần tử còn lại có giá trị là 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 \mathbf{W} là một biểu diễn vector có số chiều là N tương ứng với một từ \mathbf{w} trong tập từ vựng.

Với mỗi vector x^k , ta có:

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

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

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

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 nối với đầu ra tương ứng chính là trọng số của từng từ có trong V, trọng số càng cao có nghĩa rằng 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 dùng softmax ở tầng này nhằm đổi trọng số của tầng kết quả thành phân phối xác suất, hay còn được gọi là bước chuẩn hóa.

Tính điểm số cho mỗi từ thứ j trong V theo công thức:

$$p_j = v'_{wj}^T h \quad (4.3)$$

với v'_{wj} là vector cột thứ j của ma trận W' .

Hàm softmax để chuẩn hóa phân phối hậu nghiệm của mỗi tập từ vựng như sau:

$$p(w_O | w_{I,1}, w_{I,2}, \dots, w_{I,C}) = y_j = \frac{\exp(u_j)}{\sum_{j'}^V \exp(u_{j'})} \quad (4.4)$$

trong đó y_j chính là đầu ra của nút thứ j trong tầng kết quả, w_O là từ mục tiêu, $w_{I,1}, \dots, w_{I,C}$ là các từ ngữ cảnh.

Mục tiêu của quá trình huấn luyện là điều chỉnh các trọng số để cực đại hóa hàm phân phối bên trên đối với từ w_O và các từ ngữ cảnh $w_{I,1}, \dots, w_{I,C}$ cho trước với công thức sau:

$$\begin{aligned} \max(p(w_O | w_{I,1}, w_{I,2}, \dots, w_{I,C})) &= \max(\log(p(w_O | w_{I,1}, w_{I,2}, \dots, w_{I,C}))) \\ &= \min(-\log(p(w_O | w_{I,1}, w_{I,2}, \dots, w_{I,C}))) \quad (4.5) \\ &= \min(E) \end{aligned}$$

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

$$\begin{aligned} E &= -\log(p(w_O | w_{I,1}, w_{I,2}, \dots, w_{I,C})) \\ &= -u_{j^*} + \log \left(\sum_{j'=1}^V \exp(u_{j'}) \right) \quad (4.6) \\ &= -v'_{wO}^T h + \log \left(\sum_{j'=1}^V \exp(v'_{wO}^T h) \right) \end{aligned}$$

với 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:

$$v_{wj'}^{new} = v_{wj'}^{old} - \eta \times \frac{\partial E}{\partial u_j} \times h \quad ; \text{với } j = 1, 2, \dots, V \quad (4.7)$$

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

$$v_{wc}^{new} = v_{wc}^{old} - \frac{1}{C} \times \eta \times \left(\frac{\partial E}{\partial h_i} \right)^T ; \quad \text{với } c = 1, 2, \dots, C \quad (4.8)$$

trong đó v_{wc} 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:

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

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.

Để hiểu hơn về cách thức hoạt động của CBOW, xem xét câu "Hôm qua tôi đi học".

Với kích thước của $s = 1$, 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:

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

Ta có:

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

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

Sau đó tính được h như sau:

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

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

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

Sau khi kết thúc quá trình huấn luyện sẽ nhận được vector mục tiêu dưới dạng xác suất của từ "đi" như sau:

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

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.

Vì có bản chất là xác suất cho nên việc hiện thực của mô hình CBOW được cho là vượt trội hơn so với phương pháp khác nói chung. Không những thế, CBOW là mô hình được đánh giá là mô hình sử dụng ít bộ nhớ trong quá trình tính toán.

Mô hình 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 ngữ cảnh và đặt ở giữa. Điều này được cho là nhược điểm của CBOW.

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

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:

$$\begin{bmatrix} 0.2 & 0.1 & 0.5 & 0.2 \\ 0.6 & 0.5 & 0.3 & 0.3 \\ 0.4 & 0.2 & 0.8 & 0.9 \\ 0.1 & 0.7 & 0.5 & 0.3 \\ 0.8 & 0.4 & 0.9 & 0.1 \\ 0.5 & 0.1 & 0.1 & 0.3 \\ 0.1 & 0.6 & 0.1 & 0.7 \\ 0.3 & 0.2 & 0.4 & 0.5 \end{bmatrix}$$

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

Hãy 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 4.4.2, 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 4.4.2, 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 và 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 4.4.2, 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.

Chương 5

Mạng nơ-ron tích chập

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 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 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 hai 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 *canh thẳng* (straight edges). Các tế bào thị giác được chia làm hai loại, đó là *tế bào tối giản* (simple cell) và *tế bào phức hợp* (complex cell). Hai nhà khoa học cũng đưa ra mô hình xếp tầng

(cascading model) để lý giải cách nhận dạng mẫu (pattern recognition) của hai loại tế bào này (Hubel and Wiesel, 1959).

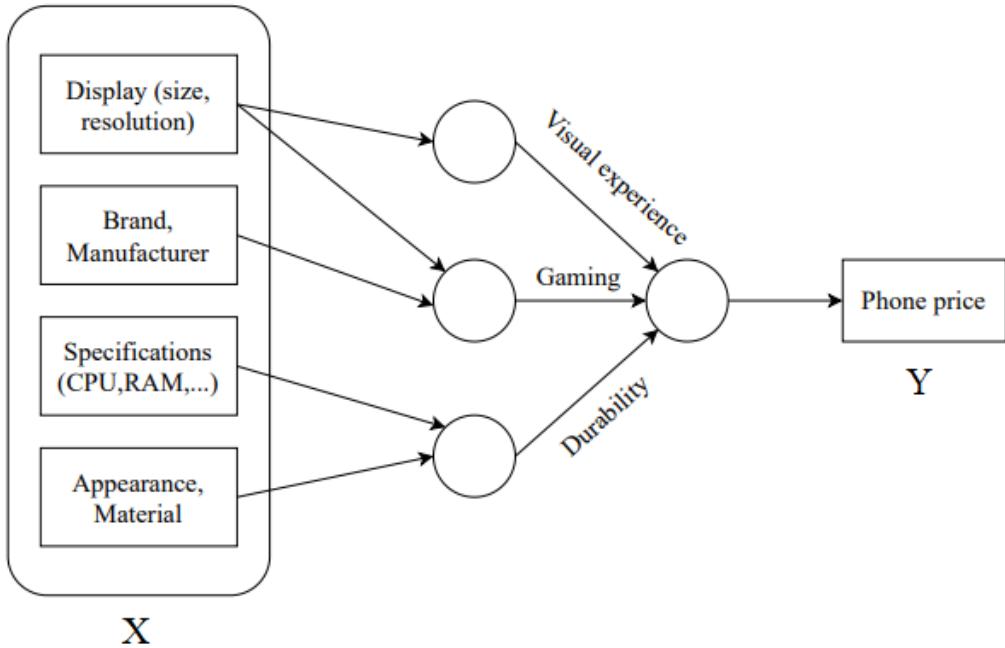
Đến năm 1979, nhà khoa học dữ liệu người Nhật Bản Kunihiko Fukushima đưa ra khái niệm "Neocognitron", đây được xem là mạng nơ-ron tích chập nguyên thủy của ngành Thị giác máy tính. Dựa trên những ý tưởng của Hubel và Wiesel, các *tầng* (layer) của Neocognitron được chia làm hai loại cơ bản: tầng S-cells 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. Thoạt nghe thì có vẻ CNN không dính dáng tới GPU lắm vì chúng thuộc hai 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.

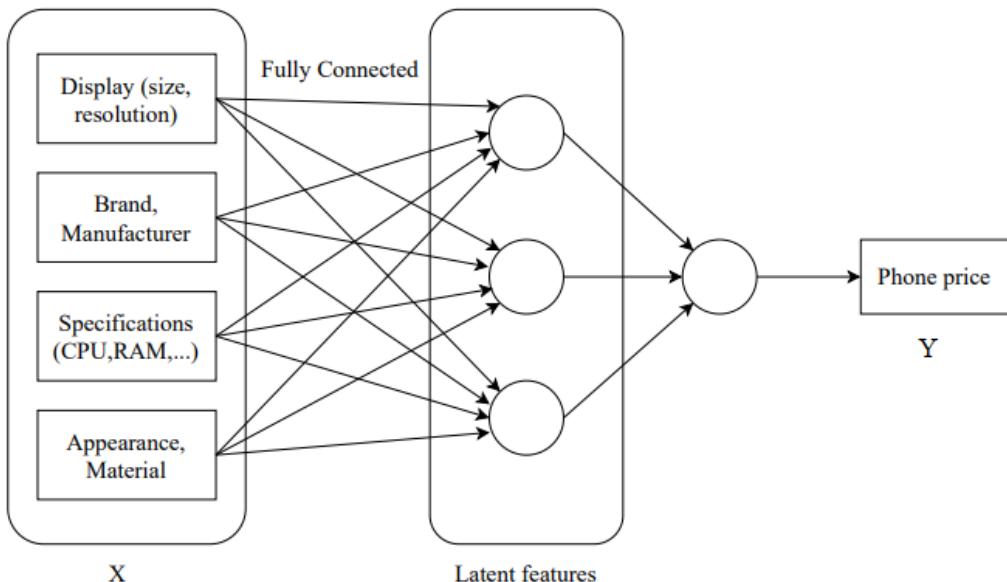


Hình 5.1: Phương thức phân tích dữ liệu và dự đoán giá sản phẩm điện thoại thông minh

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: Quá trình học và trích xuất thuộc tính của một mạng nơ-ron dự đoán giá sản phẩm điện thoại thông minh.

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 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 dày 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)

Image Matrix

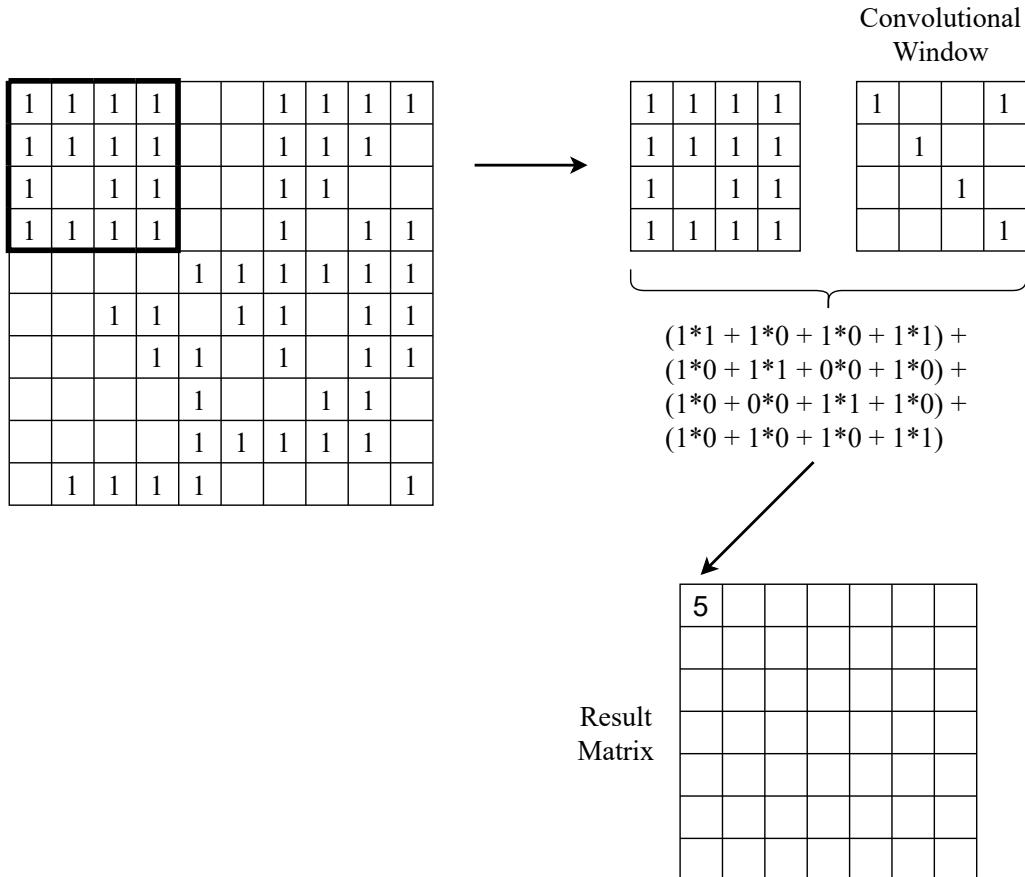
1	1	1	1			1	1	1	1
1	1	1	1			1	1	1	
1		1	1			1	1		
1	1	1	1			1		1	1
				1	1	1	1	1	1
		1	1		1	1		1	1
			1	1		1		1	1
				1	1	1	1	1	
					1	1	1	1	
	1	1	1	1					1

Convolutional Window (Filter 1)

1			1
	1		
		1	
			1

Hình 5.3: Ma trận hình ảnh (trái) và ma trận cửa sổ tích chập (phải)

Dùng 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: Quá trình nhân tích chập cho phần tử đầu tiên

Xét một hình ảnh được biểu diễn dưới dạng ma trận I kích thước 10×10 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:

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

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 4×4 tại vị trí đầu tiên của ma trận hình ảnh. Phép nhân tích chập được thực hiện bằng cách nhân từng phần tử của ma trận hình ảnh con với phần tử tại vị 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: 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 7×7 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.

			1
		1	
	1		
1			1

Filter 2

1			1
1			
1			
1			

Filter 3

		1	
	1		1
		1	
		1	

Filter 4

Hình 5.6: 3 ma trận cửa sổ trượt tiếp theo

3	3	2	3	2	4	5
3	3	2	2	4	5	2
2	3	2	2	4	4	3
2	2	1	5	3	3	4
1	2	3	3	3	3	2
0	3	3	3	3	3	4
2	3	2	3	2	3	4

Kết quả tích chập sử dụng filter 2

5	3	3	5	1	1	5
4	3	0	4	2	2	4
3	2	3	3	3	2	3
2	2	2	3	3	3	4
0	1	1	2	5	3	3
0	2	2	1	4	3	3
1	2	1	3	4	2	3

Kết quả tích chập sử dụng filter 3

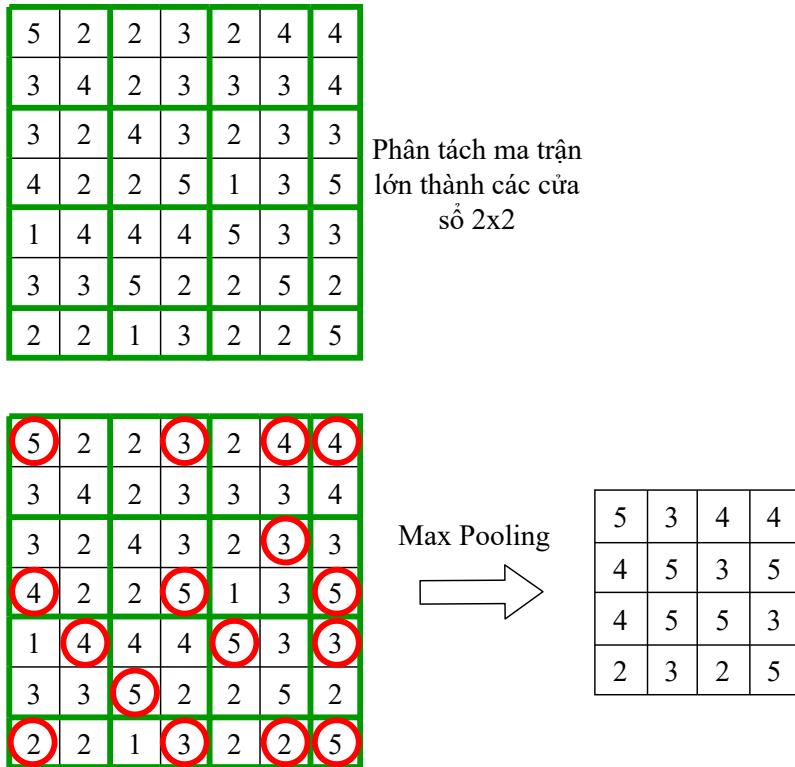
5	3	1	1	4	4	3
2	3	1	2	4	3	4
4	2	3	3	2	4	3
2	3	3	3	4	3	4
1	3	4	2	3	3	4
2	1	4	4	4	4	4
2	3	3	1	3	2	3

Kết quả tích chập sử dụng filter 4

Hình 5.7: 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 7×7 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 (average 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: 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.

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.

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

Kết quả max pooling sử dụng filter 1

3	3	4	5
3	5	4	4
3	3	3	4
3	3	3	4

Kết quả max pooling sử dụng filter 2

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

Kết quả max pooling sử dụng filter 3

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

Kết quả max pooling sử dụng filter 4

Hình 5.9: 4 ma trận kết quả sau khi thực hiện max pooling.

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 có được 4 ma trận kết quả thì dễ dàng nhận thấy rằng giá trị tối đa có được của mỗi phần tử là 5. Giá trị này chỉ có thể có khi toàn bộ các phần tử có giá trị là 1 trên cửa sổ filter cũng có giá trị là 1 tại vị trí tương ứng trên ma trận gốc, hay nói cách khác, toàn bộ hình ảnh trên cửa sổ filter đều xuất hiện toàn vẹn tại vị trí tương ứng trên cửa sổ gốc. Vì vậy, có thể nói rằng, số 5 thể hiện sự xuất hiện hoàn chỉnh của đối tượng được biểu diễn filter trên cửa sổ gốc.

Để thể hiện điều này, chúng ta sẽ tạo ra một cửa sổ hiển thị bằng cách chỉ biểu diễn các giá trị 5 (dưới dạng dấu \times trên cửa sổ kết quả) như Hình 5.10.

x			
	x		x
	x	x	
			x

Result 1

			x
	x		

Result 2

x	x		x
		x	

Result 3

x			

Result 4

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

Có thể thấy, sau khi biến đổi thì các ma trận kết quả đang thể hiện được sự phân 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 nầm 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 toán như *bộ phát hiện cạnh* (edge detector), *bộ phát hiện nhiễu* (noise detector),...

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*, 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.

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: Bản đồ thuộc tính được tạo thành từ việc ghép các ma trận kết quả

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*.

Như vậy, đối với con người thì bản đồ thuộc tính chỉ là một ma trận mới được

tạo ra từ ma trận ban đầu sau khi thực hiện các phép tích chập và phép gộp. Tuy nhiên, đối với máy tính thì ma trận đặc trưng là bộ biểu diễn các thuộc tính tiềm ẩn được rút trích từ các ma trận trước.

Vì bản chất của ma trận đặc trưng là ma trận mới được tạo ra từ ma trận gốc nên có thể tiếp tục lặp lại quá trình tích chập và gộp thêm một số lần khác nữa để rút trích ra được cá thuộc tính trừu tượng cao hơn.

5.2.5 Cách áp dụng bộ lọc

Cho một hình ảnh thực như Hình 5.12a có thể được biểu diễn dưới dạng mảng 2 chiều chứa các giá trị pixel.

Dựa vào nội dung trình bày về phép tích chập trong Mục 5.2.1, ta có thể dễ dàng xây dựng bộ lọc phát hiện các đường thẳng hoặc đường ngang từ ảnh gốc.

Như Hình 5.12b, bộ lọc có ma trận cửa sổ trượt được chọn có kích thước 3×3 dùng để lọc các đối tượng có dạng đường thẳng có trong hình gốc. Và ngược lại, Hình 5.12c là bộ lọc với ma trận cửa sổ trượt 3×3 được dùng để lọc ra và hiển thị tất cả các đối tượng có dạng đường ngang xuất hiện trong hình gốc.

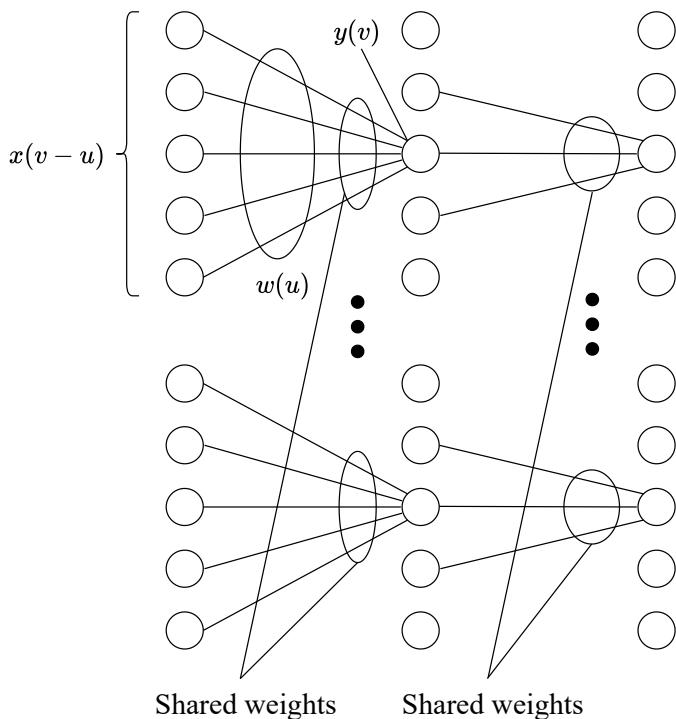


Hình 5.12: Trích xuất đặc trưng của hình ảnh theo chiều dọc (b) và chiều ngang (c)

Như vậy, tùy vào mục đích của bộ lọc mà ta cần thiết kế và chọn lựa ma trận cửa sổ trượt thích hợp nhằm trích xuất được đặc trưng ứng với mục tiêu đề ra.

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 trận 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 5.13: Quá trình nhân tích chập một bộ lọc tạo ra các nơ-ron

Hình 5.13 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 lọc 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 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 $3 \times 3 \times 3$ 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 toán 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 $3 \times 3 \times 3 = 27$
- Tổng số tham số cho mỗi bộ lọc là $27 + 1 = 28$
- Tổng số tham số cho 10 bộ lọc là $28 \times 10 = 280$

Qua ví dụ trên, có thể rút ra nhận xét rằng số lượng tham số được tạo ra bởi mạng CNN này luôn luôn là 280, không phụ thuộc vào độ lớn của dữ liệu đầu vào.

5.4 Triển khai một mạng CNN

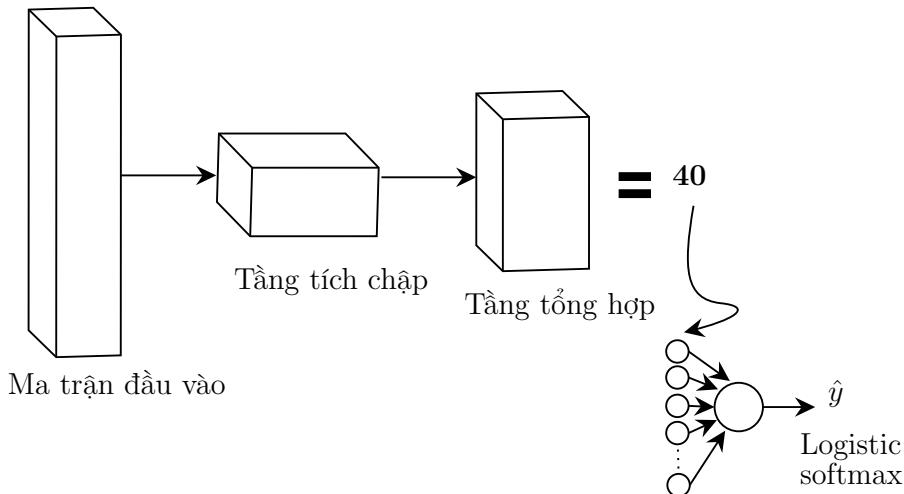
Khi triển khai trong một kiến trúc mạng end-to-end hoàn chỉnh thì một mạng CNN cơ bản bao gồm 3 bộ phận chính: *Tầng tích chập* (Convolution), *tầng tổng hợp* (Pooling), *tầng kết nối đầy đủ* (Fully Connected) có cách thức hoạt động giống như tầng kết nối trong mạng nơ-ron đã mô tả trong Chương 3. Trong đó:

- 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. Tầng này có các bộ phận chính là một ma trận đầu vào, bản đồ thuộc tính và các cửa sổ trượt là các khái niệm đã được trình bày trong Mục 5.2.
- Tầng tổng hợp có mục đích làm giảm số lượng thông số mà ta phải tính toán, điều này giúp 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*).

Nhưng tại sao tầng cuối phải là kết nối đầy đủ? Trong thực tế, mạng CNN phải được huấn luyện như một hàm mục tiêu cụ thể để giải quyết một bài toán cụ thể nên tầng cuối đóng vai trò của việc đưa ra kết quả học và tính được giá trị của hàm mất mát, từ đó mạng CNN có thể học và cập nhật lại bộ trọng số của mạng bằng phương pháp lan truyền ngược.

Như vậy, vai trò của một tầng kết nối đầy đủ là giúp cho mạng CNN có thể học được các trọng số trên các cửa sổ trượt. Nếu chỉ có tầng tích chập và tầng tổng hợp thì mạng CNN không thể biết được cần phải học những gì cũng như không có cơ sở nào để có thể thực hiện việc cập nhật lại bộ trọng số.

Hình 5.14 biểu diễn kiến trúc một mạng CNN để giải quyết bài toán phân loại nhị phân. Trong đó bao gồm một ma trận đầu vào, một lớp tích chập và một lớp tổng hợp. Giả sử 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 kết quả sau cùng.



Hình 5.14: Kiến trúc mạng CNN

Như phần trên đã đề cập, mạng CNN có thể kết nối nhiều tầng tích chập và tầng tổng hợp liên tiếp nhau để học các thuộc tính ẩn có tính trừu tượng cao. Tuy nhiên, kiến trúc mạng CNN phải luôn luôn kết thúc với một tầng kết nối đầy đủ nhằm thực hiện việc học và cập nhật trọng số thông qua cơ chế lan truyền ngược. Trong thực tế, hoàn toàn có thể dùng nhiều hơn một tầng kết nối đầy đủ nhằm tăng khả năng học, nhưng không nên dùng quá nhiều vì sẽ mất đi ý nghĩa của mạng CNN là giảm bớt số lượng của bộ trọng số cần phải học.

Mạng LeNet5 (Lecun Y. and P., 1998) 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.

Một số thông tin về kiến trúc mạng LeNet5 như sau:

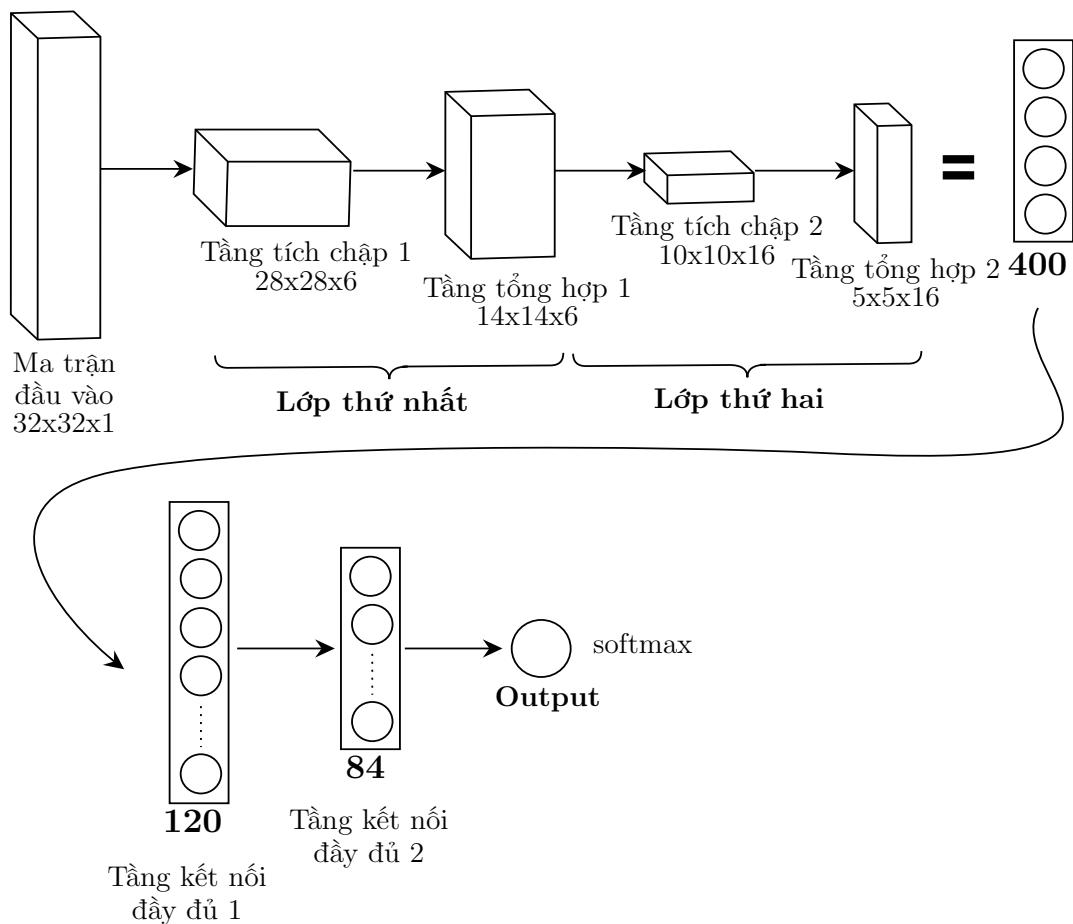
- Đầu vào là một ma trận có kích thước $32 \times 32 \times 1$, nghĩa là một tấm ảnh có chiều dài 32 pixel, chiều rộng ảnh là 32 pixel và số lượng kênh ảnh là 1 (ảnh đen trắng).
- Lớp thứ nhất bao gồm:
 - Tầng tích chập thứ 1: là một ma trận $5 \times 5 \times 3$, tốc độ stride là 1, đi qua 6 cửa sổ 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 $5 \times 5 \times 16$.



Hình 5.15: Kiến trúc mạng LeNet-5

- 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 trong Chương 4, *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, TF-IDF hay mô hình n-gram để giải quyết.

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

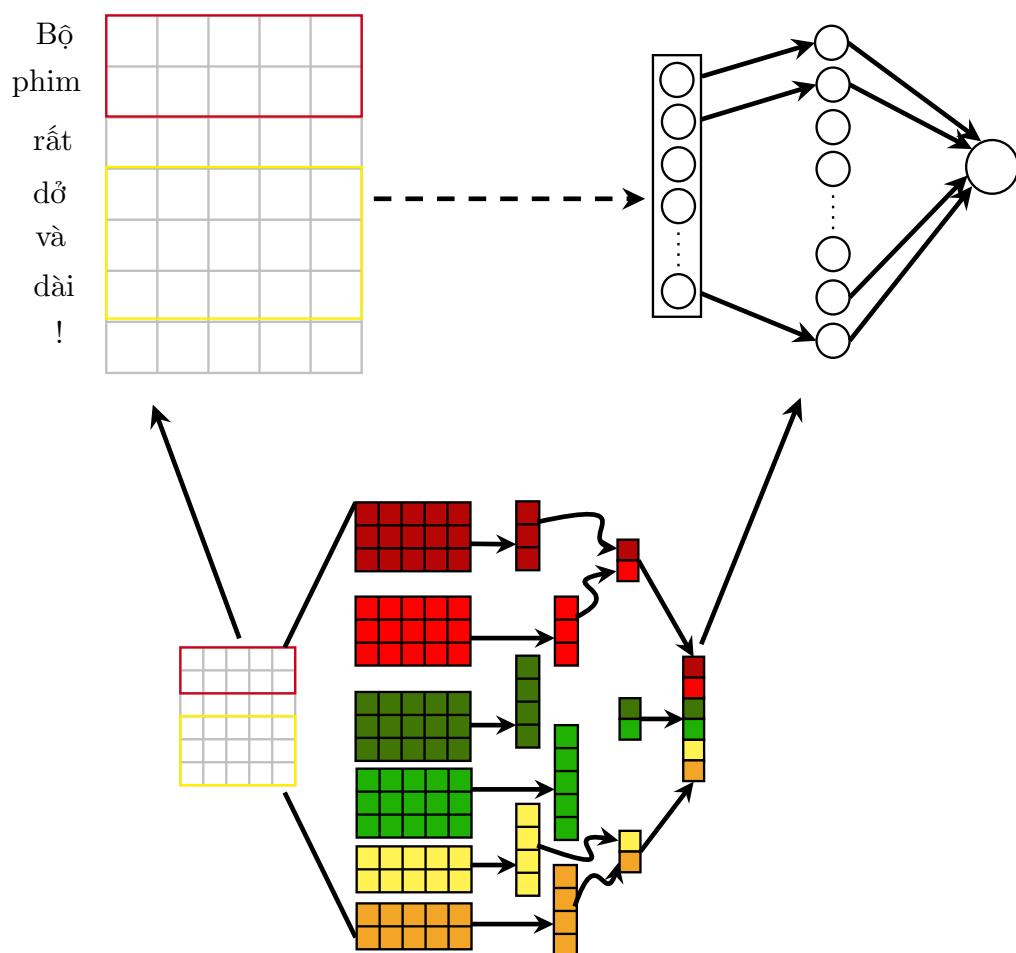
Với phương pháp Word Embedding, mỗi từ có thể biểu diễn thành một vector, từ đó một văn bản có thể giải thích thành một ma trận như hình bên dưới:



Hình 5.16: Biểu diễn mỗi từ trong câu thành vector bằng Word Embedding

Từ Hình 5.16 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 ($2 \times k$) 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.

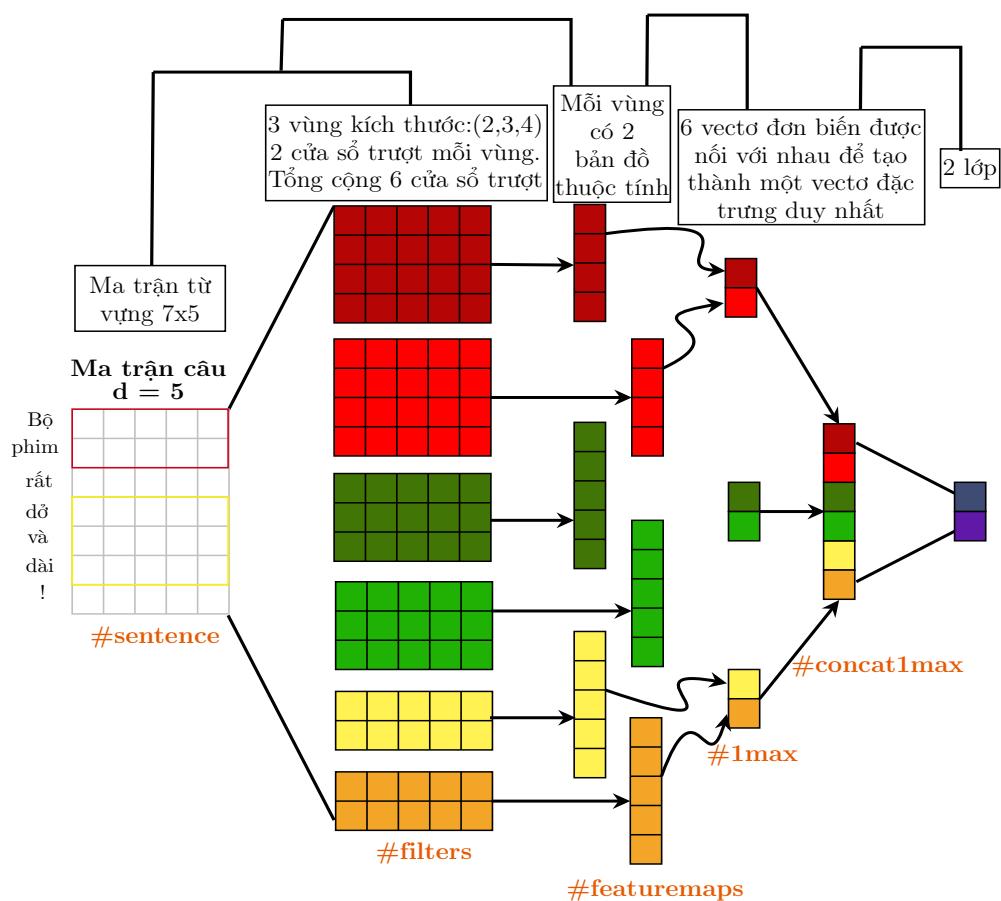
Với ma trận từ được biểu diễn như Hình 5.16 ở trên, ta hoàn toàn có thể sử dụng như là input cho một mạng nơ-ron như bình thường, tuy nhiên với cách tiếp cận mới bằng các kỹ thuật Word Embedding, ta có thể sử dụng thêm tầng CNN để rút trích các latent features, được thể hiện như Hình 5.17 ở dưới, và cách biểu diễn kinh điển của một tầng CNN sẽ được trình bày trong case study tiếp theo.



Hình 5.17: Sử dụng thêm tầng CNN trong cách tiếp cận mới

5.5.3 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.



Hình 5.18: Sơ đồ biểu diễn kiến trúc CNN cho bài toán phân loại cảm xúc

Trong phần này, chúng ta sẽ đi vào chi tiết một case study là sử dụng CNN cho bài toán *sentiment analysis*. Hình 5.18 là một sơ đồ biểu diễn kiến trúc CNN cho bài toán phân loại cảm xúc.

Dễ dẽ hiểu, chúng ta sẽ chia sơ đồ Hình 5.18 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 $s \times d$ là 7×5 .

#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 convolution đầu tiên ta sẽ có 6 bộ 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 ($= \text{số bộ lọc} = \text{số bộ lọc cho mỗi kích thước vùng (2)} \times \text{số lượng kích thước vùng được xem xét (3)}$). 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 mượt mà 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 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* 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 5.19. 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.

Image Matrix

			1			1	1	1
1	1		1	1		1		
1		1	1			1		
1	1		1			1	1	1
	1			1		1		1
		1	1		1		1	
1		1	1					
	1			1				
1	1	1	1	1				1

Convolutional Window

1			1
1			1

Hình 5.19: 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 5.20 và ở tầng thứ hai có 1 cửa sổ trượt như Hình 5.21. 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

	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

Image Matrix

		1	1		1	1			1
1	1	1					1	1	
1	1			1	1	1		1	1
1					1	1			
				1			1		1
1	1				1	1	1		1
				1				1	1
			1	1	1				
	1	1	1			1	1		
1	1		1	1		1	1	1	1

Convolutional Window

1			1
	1		
1		1	
			1

1			1
1			
			1

Hình 5.20: Ma trận hình ảnh và 2 cửa sổ tích chập ở tầng đầu tiên trong Bài tập 5.6.2

Convolutional
Window

1			1
	1	1	
		1	
1			1

Hình 5.21: Cửa sổ tích chập cho tầng thứ 2 trong Bài tập 5.6.2

Ma trận Embedding của câu này sẽ là

$$\begin{bmatrix} 8 & 2 & 1 & 9 \\ 2 & 4 & 7 & 0 \\ 1 & 3 & 5 & 8 \\ 0 & 1 & 9 & 4 \\ 7 & 8 & 1 & 2 \\ 1 & 5 & 8 & 9 \\ 5 & 1 & 5 & 2 \end{bmatrix}$$

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.

Chương 6

Mạng nơ-ron truy hồi (Recurrent Neural Network - RNN)

6.1 Giới thiệu về dữ liệu chuỗi

6.1.1 Dữ liệu chuỗi là gì

Dữ liệu chuỗi (*sequence data*) là một chuỗi dữ liệu gồm nhiều phần tử có thứ tự. Khác với kiểu dữ liệu truyền thống khi các mẫu là phi thứ tự và độc lập lẫn nhau, với dữ liệu chuỗi, 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.

Xét câu "Tôi thích ăn phở nhưng không thích ăn bún bò", tập hợp từ xuất hiện trong câu bao gồm { "Tôi", "thích", "ăn", "phở", "nhưng", "không", "thích", "ăn", "bún_bò" }.

Giả sử bỏ qua thông tin về thứ tự của câu thì sẽ có các câu cùng tập từ nhưng khác thứ tự như "Tôi thích ăn bún bò nhưng không thích ăn phở" hoàn toàn trái ngược nghĩa của câu ban đầu, hay trong trường hợp khác như "Tôi phở

thích ăn nhưng bún bò không thích ăn " 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 chuỗi thời gian (time series):**

Giả sử có dữ liệu về nhiệt độ đúng 12h trưa mỗi ngày và 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 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,...

Dữ liệu chuỗi thời gian là một trường hợp cụ thể của dữ liệu chuỗi 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 thì cần tiếp nhận chúng (qua tai nghe, mắt nhìn) một cách tuần tự. Con người 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 dữ liệu chuỗi thời gian.

Một đặc điểm đặc biệt của dữ liệu chuỗi là một chuỗi dữ liệu có thể có độ dài tùy ý. 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 chuỗi. Với dữ liệu dạng độ dài xác định, nhiều kỹ 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 chuỗi khá phổ biến trong các ứng dụng thực tế. Các ứng dụng về phân tích dữ liệu nói chung và machine learning nói riêng cũng đã được nghiên cứu và nhiều phương pháp được đề xuất để xử lý kiểu dữ liệu này. Một số bài toán trên dữ liệu chuỗi 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, ...), 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 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 1-1 giữa 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.

Long is handsome → Long [noun] is [verb] handsome [adj]

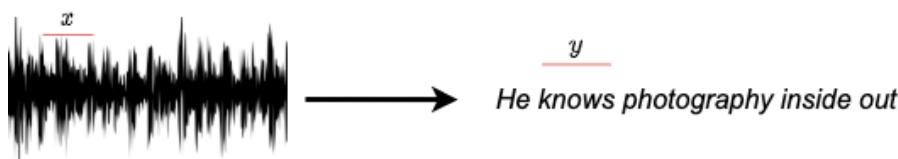
The rail is very long → The [adv] rail [noun] is [verb] very [adv] long [adj]

Hình 6.2: Phân loại từ trong một câu

- **Dạng sequence-to-sequence, hay many-to-many bất đối xứng:**

Dữ liệu đầu vào và đầu ra đều là dạng chuỗi, tuy nhiên có độ dài khác nhau và không có sự tương ứng 1-1 giữa 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 chuỗi đầu vào (audio) và một chuỗi đầu ra (chuỗi văn bản) với độ dài khác nhau. Bài toán speech recognition được liệt vào dạng sequence-to-sequence.



Hình 6.3: 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 dịch máy 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.

Je voudrais essayer ceci ————— I would like to try this on

Hình 6.4: Bài toán dịch máy

- **Dạng many-to-one:**

Dữ liệu đầu vào dạng chuỗi, đầu ra dạng cố định. Bài toán phân tích cảm xúc là một trường hợp cụ thể.

Giả sử, xét một bài đánh giá phim của khách hàng, input sẽ là chuỗi văn bản đánh giá của khách hàng, output là cảm xúc của khách hàng tích cực/tiêu cực.

Quả là một bộ phim hay —————> Tích cực

Hình 6.5: Bài toán phân tích cảm xúc

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 mạng đa tầng và mạng tích chập) được xếp vào nhóm mạng feed-forward. 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 chuỗi, đầu ra y_i tại một vị trí bất kỳ 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 chuỗi. Một số ví 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).

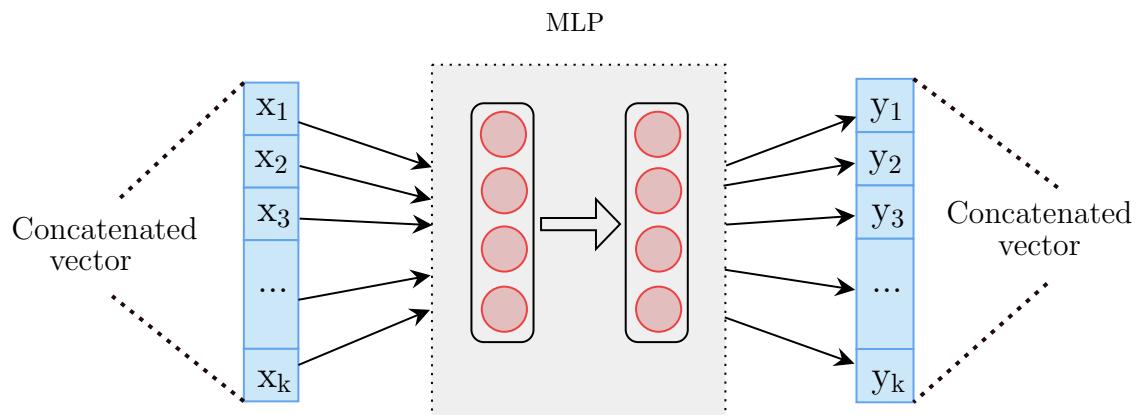
Dữ liệu dạng chuỗi đôi khi còn nằm ở dạng chuỗi dữ liệu thuần túy không dựa trên quan sát:

- 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 số xổ 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 chuỗi có độ dài tùy ý. Tuy nhiên, dữ liệu chuỗi 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 mô hình tiếp cận theo hướng này được xếp vào nhóm "auto regressive models". Về tổng quát, một mô hình auto regressive bậc k (k^{th} -order) tổng quát có thể được hiểu là một ánh xạ:

$$\hat{y}_i = f(x_i, y_{i-1}, y_{i-2}, \dots, y_{i-k}) \quad (6.1)$$

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 mạng đa tầng cho bài toán dữ liệu chuỗi theo dạng mô hình regressive

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 mạng đa tầng không phù hợp cho dữ liệu dạng chuỗi

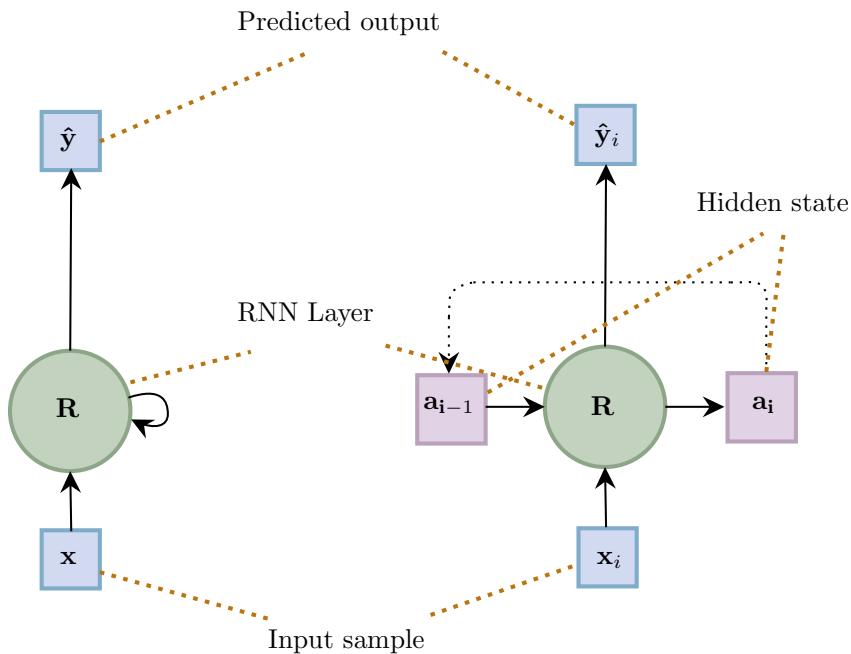
- Mô hình chưa có khả năng ghi nhớ trạng thái từ các dữ liệu trước đó để hỗ trợ ra quyết định, vẫn phụ thuộc hoàn toàn vào đầu vào. Trong trường hợp cần ghi nhớ dữ liệu ở mức lớn hơn bậc k của mô hình, mạng feed-forward sẽ không thể đưa ra dự đoán chính xác. Thật vậy, cách mà mô hình regressive nêu trên lấy dữ liệu từ các mẫu trước đó trong chuỗi hoàn toàn là do giả định của người thiết kế mạng.

- Số lượng tham số là rất lớn: khi sử dụng tầng kết nối hoàn toàn, số lượng tham số của một lớp sẽ bằng tích của chiều đầu vào lẫn chiều đầu 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 riêng cho từng đầu vào. Vì mỗi đầu vào x_i đều mang tính chất là một điểm dữ liệu trong chuỗi, 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 mạng tích chập cho bài toán xử lý ảnh, vấn đề của mạng đa tầng khi giải quyết bài toán liên quan đến ảnh cũng tương tự như với dữ liệu dạng chuỗi. 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), mạng tích chập 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 chuỗi, 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

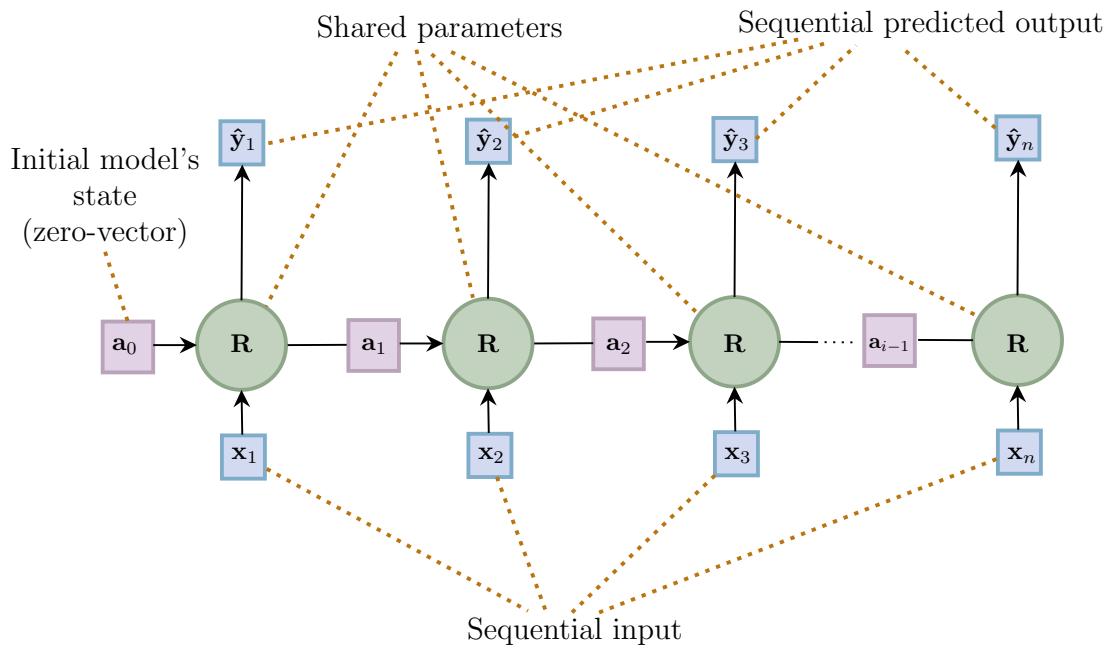
Năm 1986, Rumelhart và các cộng sự của mình đã công bố một bài nghiên cứu mang tên “*Học biểu diễn nội bộ bằng lan truyền lỗi*” (Learning internal representations by error propagation) (Rumelhart et al., 1986). Trong bài nghiên cứu có trình bày một kiến trúc mạng kinh điển để xử lý dữ liệu dạng chuỗi, có tên là mạng nơ-ron truy hồi. Mạng nơ-ron truy hồi là một kiến trúc mạng neural 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 mạng nơ-ron truy hồi 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 mạng nơ-ron truy hồi được biểu diễn với đường nối vòng (trái) và biểu diễn với hidden state (phải)

Với mạng nơ-ron 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 kỳ 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 là "gỡ" đường nối vòng (unfold) để đưa ra kiến trúc tính toán cụ thể hơn khi truyền dữ liệu chuỗi vào mạng.

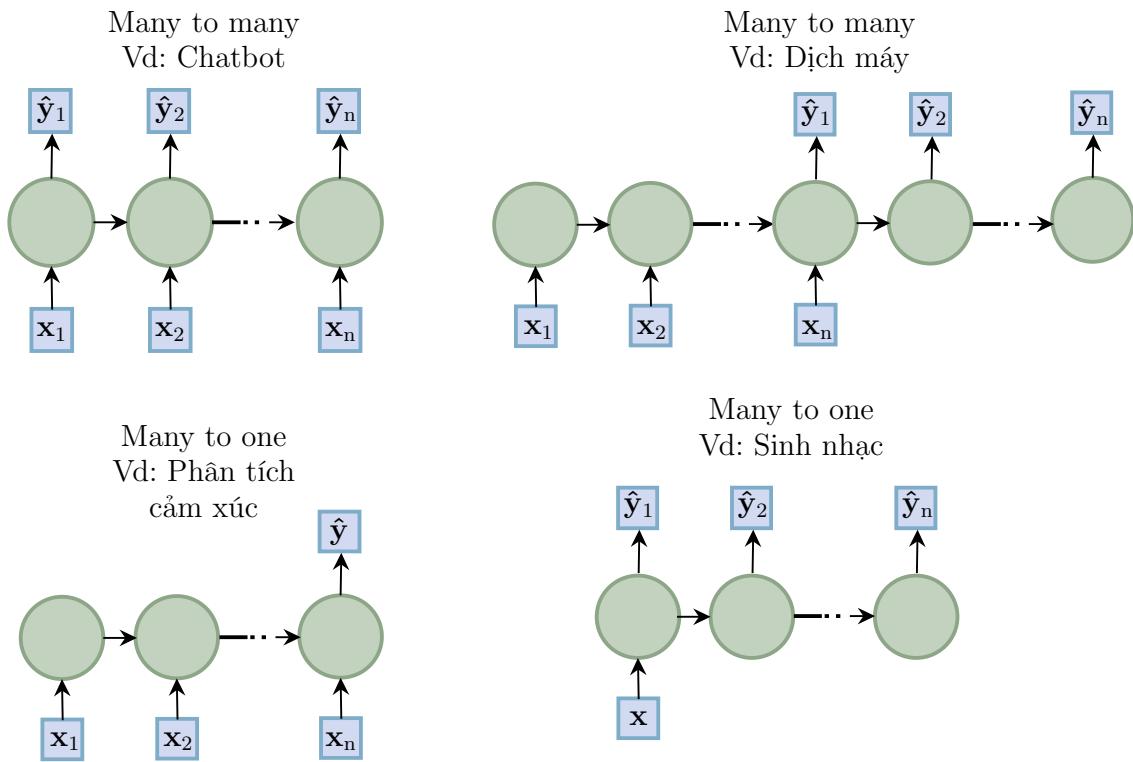
Với mạng nơ-ron truy hồi, việc thông tin từ lần truyền trước được lưu lại (tại hidden state) cho lần tính toán hiện tại, tạo ra một đường nối vòng trên đồ thị tính toán. Nói cách khác, hidden state đóng vai trò như một bộ nhớ của lớp nơ-ron truy hồi. Điều này đặc biệt hữu dụng khi xử lý dữ liệu dạng chuỗi, 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 chuỗi.



Hình 6.8: Luồng tính toán cụ thể trên sequence của RNN

Năm 1982, Hopfield đã đưa ra kiến trúc Hopfield network (Hopfield, 1982b) là một dạng mạng nơ-ron 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 mạng nơ-ron truy hồi. Trọng số trên mạng Hopfield được cập nhật bằng phương pháp tối ưu lồi (khác với gradient descent ở mạng nơ-ron truy hồi hiện đại). Mạng nơ-ron truy hồi 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 mạng nơ-ron truy hồi nêu trên cho mọi bài toán với dữ liệu chuỗi. Một số mẫu biến thể thiết kế (design patterns) từ kiến trúc trên để áp dụng cho từng loại bài toán được mô tả trong hình sau.



Hình 6.9: Các mẫu thiết kế mạng nơ-ron truy hồi cho từng loại bài toán

6.3 Cơ chế lan truyền xuôi của mạng nơ-ron truy hồi

Để tìm hiểu rõ hơn cách hoạt động cụ thể của mạng nơ-ron truy hồi, mục này sẽ trình bày cơ chế để lan truyền một chuỗi dữ liệu qua mạng. Ta xét kiến trúc mạng cụ thể cơ bản của mạng nơ-ron truy hồi, được biểu diễn dưới dạng công thức truy hồi như sau (lưu ý vector được biểu diễn dưới dạng cột):

$$\mathbf{a}_0 = 0 \quad (6.2)$$

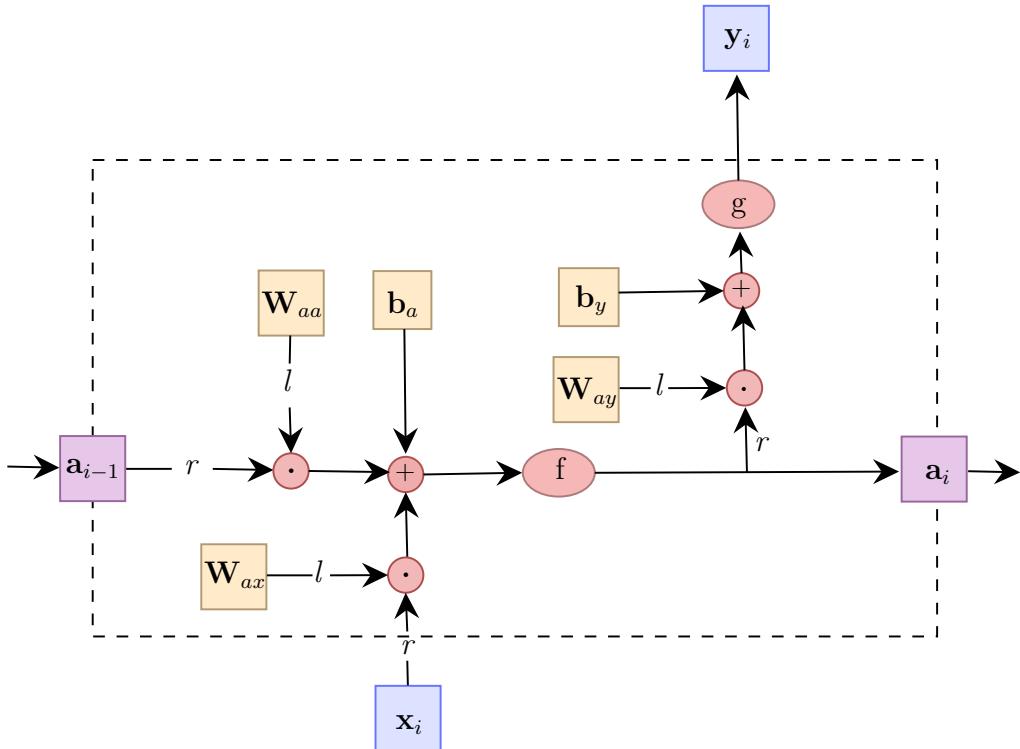
$$\mathbf{a}_i = f(\mathbf{W}_{ax} \cdot \mathbf{x}_i + \mathbf{W}_{aa} \cdot \mathbf{a}_{i-1} + \mathbf{b}_a) \quad (6.3)$$

$$\hat{\mathbf{y}}_i = g(\mathbf{W}_{ya} \cdot \mathbf{a}_i + \mathbf{b}_y) \quad (6.4)$$

Các biến số và tham số trong công thức truy hồi trên được định nghĩa là:

- \mathbf{a}_i là hidden state - trạng thái ẩn của mô hình. $\mathbf{a}_0 = 0$ là quy ước cho trạng thái bắt đầu.
- $\hat{\mathbf{y}}_i$ là giá trị dự đoán đầu ra ứng với bước i .
- \mathbf{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 $g(x) = \text{Softmax}(\mathbf{x})$, tùy vào cách thiết kế và yêu cầu bài toán.
- \mathbf{W} và \mathbf{b} là các trọng số của mô hình, cụ thể:
 - $\mathbf{W}_{\mathbf{ax}}$ là ma trận của phép ánh xạ tuyến tính biến đổi \mathbf{x} thành một phần trong \mathbf{a}
 - $\mathbf{W}_{\mathbf{aa}}$ là ma trận của phép ánh xạ tuyến tính biến đổi trạng thái \mathbf{a} cũ thành một phần trong trạng thái \mathbf{a} mới.
 - $\mathbf{b}_{\mathbf{a}}$ là bias của phép biến đổi tìm \mathbf{a}
 - $\mathbf{W}_{\mathbf{ya}}$ là ma trận của phép ánh xạ tuyến tính biến đổi trạng thái hiện tại \mathbf{a} thành kết quả đầu ra $\hat{\mathbf{y}}$
 - $\mathbf{b}_{\hat{\mathbf{y}}}$ 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 mạng nơ-ron hồi quy

Về mặt lý thuyết, với mỗi nút đầu vào của chuỗi dữ liệu, mạng nơ-ron hồi quy 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. Hãy ví dụ trong các trường hợp sau:

- Với bài toán dự đoán giá cổ phiếu (dạng many-to-many đối xứng): Giả sử chuỗi dữ liệu 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 ý đều tương ứng với một ngày nào đó trong chuỗi. 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 phân tích cảm xúc (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 dự 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 "Bộ phim này không quá tệ" khi được truyền toàn bộ vào mô hình, mô hình có thể gán nhãn "tốt" cho câu (phim không tệ là phim hay)
- Tuy nhiên, cùng với câu trên, khi chỉ lấy đoạn "Bộ phim này không", mô hình lại có thể gán nhãn "xấu" cho câu (từ "không" mang nghĩa phủ định, có thể là tiêu cực)

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.

```
1 import numpy as np
2
3 dim_x = 5 # Input dimension
4 dim_y = 5 # Output dimension
5 dim_a = 3 # Hidden state dimension
6
7 n = 10 # Input sequence length
8
9 x = np.random.randn(n, dim_x)
10
11 W_ax = np.random.rand(dim_a, dim_x) # Shape(W_ax) = (3, 5)
12 W_aa = np.random.rand(dim_a, dim_a) # Shape(W_aa) = (3, 3)
13 b_a = np.random.rand(dim_a) # Shape(b_a) = (3,)
14 W_ya = np.random.rand(dim_y, dim_a) # Shape(W_ya) = (5, 3)
15 b_y = np.random.rand(dim_y) # Shape(b_y) = (5,)
16
17 a = np.zeros(dim_a) # Initial state = zeros
18 y = []
19
20 # Forwarding process
21 for i in range(n):
```

```

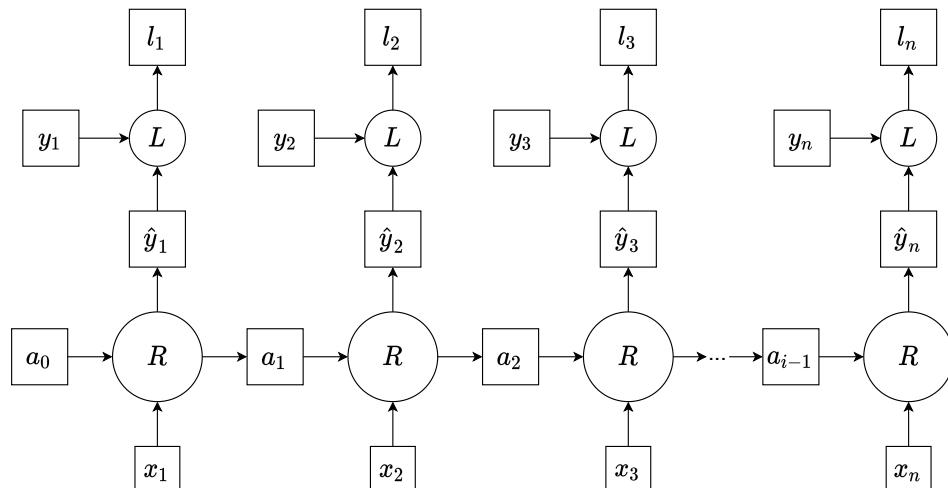
22         xi = X[i].T # Transform into column-vector
23         # Follow equation 6.3 and 6.4 with f is tanh and g is
24         softmax
25         a = np.tanh(np.dot(W_ax, xi) + np.dot(W_aa, a) + b_a)
26         yi = np.dot(W_ya, a) + b_y
27         yi = np.exp(yi) / np.sum(np.exp(yi))
28         y.append(yi)

```

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: Dưa 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 loss $\Delta w = \frac{\delta \mathcal{L}}{\delta w}$ cho từng tham số w trong mạng
4. Bước cập nhật: Cập nhật các tham số theo tốc độ học α hiện tại: $w \leftarrow w - \alpha \Delta w$



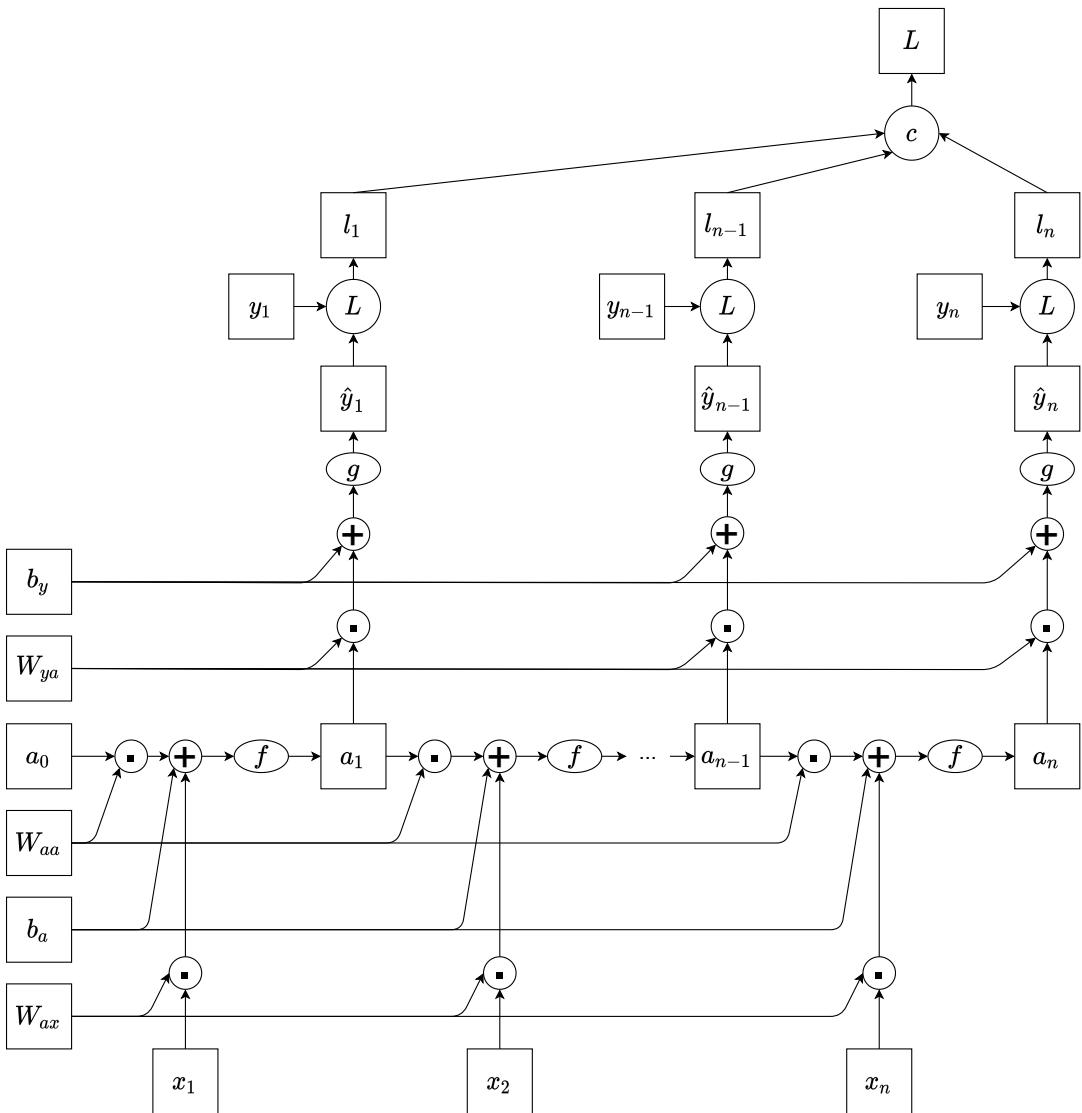
Hình 6.11: RNN với hàm mất mát

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 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.

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 tính toán được giá trị hàm mất mát dựa trên một giá trị đầu ra dự báo \hat{y} và một giá trị nhãn thực 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{y}_i và tất cả các nhãn ở từng bước 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: $\mathcal{L} = \mathcal{L}(\hat{y}_n, y_n)$
- Với các bài toán còn lại: khi có nhiều hơn một output, 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.



Hình 6.12: RNN với hàm mất mát và trọng số cụ thể

với các thành phần mới:

- $l_i = \mathcal{L}(\hat{\mathbf{y}}_i, \mathbf{y}_i)$ là giá trị hàm mất mát cho đầu ra thứ i
- $L = c(l_1, l_2, \dots, l_n)$ là giá trị hàm mất mát tổng; c là một hàm tích hợp nhiều giá trị mất mát (tổng, trung bình, 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 $\mathbf{W}_{ya}, \mathbf{b}_y, \mathbf{W}_{aa}, \mathbf{W}_{ax}, \mathbf{b}_a$:

$$\begin{aligned}\Delta \mathbf{W}_{ya} &= \frac{\partial L}{\partial \mathbf{W}_{ya}} \\ &= \sum_{i=1}^n \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\mathbf{y}}_i} \cdot \frac{\partial \hat{\mathbf{y}}_i}{\partial \mathbf{W}_{ya}} \right) \\ &= \sum_{i=1}^n \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\mathbf{y}}_i} \cdot g'(\mathbf{W}_{ya} \cdot \mathbf{a}_i + \mathbf{b}_y) \cdot \frac{\partial (\mathbf{W}_{ya} \cdot \mathbf{a}_i)}{\partial \mathbf{W}_{ya}} \right)\end{aligned}\quad (6.5)$$

$$\Delta \mathbf{b}_y = \frac{\partial L}{\partial \mathbf{b}_y} = \sum_{i=1}^n \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\mathbf{y}}_i} \cdot \frac{\partial \hat{\mathbf{y}}_i}{\partial \mathbf{b}_y} \right) = \sum_{i=1}^n \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\mathbf{y}}_i} \cdot g'(\mathbf{W}_{ya} \cdot \mathbf{a}_i + \mathbf{b}_y) \right) \quad (6.6)$$

$$\begin{aligned}\Delta \mathbf{W}_{aa} &= \frac{\partial L}{\partial \mathbf{W}_{aa}} \\ &= \sum_{i=1}^n \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\mathbf{y}}_i} \cdot \frac{\partial \hat{\mathbf{y}}_i}{\partial \mathbf{a}_i} \cdot \frac{\partial \mathbf{a}_i}{\partial \mathbf{W}_{aa}} \right) \\ &= \sum_{i=1}^n \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\mathbf{y}}_i} \cdot \mathbf{W}_{ya} \cdot \mathbf{P}_i \right)\end{aligned}\quad (6.7)$$

với

$$\mathbf{P}_i = \frac{\partial \mathbf{a}_i}{\partial \mathbf{W}_{aa}} = f'_i \cdot (\mathbf{W}_{aa} + \mathbf{I}) \cdot \frac{\partial \mathbf{a}_{i-1}}{\partial \mathbf{b}_a} = f'_i \cdot (\mathbf{W}_{aa} + \mathbf{I}) \cdot \mathbf{P}_{i-1} \quad ; \quad \mathbf{P}_0 = 0 \quad (6.8)$$

$$\Delta \mathbf{b}_a = \frac{\partial L}{\partial \mathbf{W}_{ya}} = \sum_{i=1}^n \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\mathbf{y}}_i} \cdot \frac{\partial \hat{\mathbf{y}}_i}{\partial \mathbf{a}_i} \cdot \frac{\partial \mathbf{a}_i}{\partial \mathbf{b}_a} \right) = \sum_{i=1}^n \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\mathbf{y}}_i} \cdot \mathbf{W}_{ya} \cdot \mathbf{Q}_i \right) \quad (6.9)$$

với

$$Q_i = \frac{\partial \mathbf{a}_i}{\partial \mathbf{b}_a} = f'_i \cdot \left(\mathbf{W}_{aa} \cdot \frac{\partial \mathbf{a}_{i-1}}{\partial \mathbf{b}_a} + \mathbf{I} \right) = f'_i \cdot (\mathbf{W}_{aa} \cdot Q_{i-1} + \mathbf{I}) \quad ; \quad Q_0 = 0 \quad (6.10)$$

$$\begin{aligned} \Delta \mathbf{W}_{ax} &= \frac{\partial L}{\partial \mathbf{W}_{ax}} \\ &= \sum_{i=1}^n \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\mathbf{y}}_i} \cdot \frac{\partial \hat{\mathbf{y}}_i}{\partial \mathbf{a}_i} \cdot \frac{\partial \mathbf{a}_i}{\partial \mathbf{W}_{ax}} \right) \\ &= \sum_{i=1}^n \left(\frac{\partial L}{\partial l_i} \cdot \frac{\partial l_i}{\partial \hat{\mathbf{y}}_i} \cdot \mathbf{W}_{ya} \cdot R_i \right) \end{aligned} \quad (6.11)$$

với

$$\begin{aligned} R_i &= \frac{\partial \mathbf{a}_i}{\partial \mathbf{W}_{ax}} \\ &= f'_i \cdot \left(\frac{\partial (\mathbf{W}_{ax} \cdot \mathbf{x}_i)}{\partial \mathbf{W}_{ax}} + \mathbf{W}_{aa} \cdot \frac{\partial \mathbf{a}_{i-1}}{\partial \mathbf{W}_{ax}} \right) \\ &= f'_i \cdot \left(\frac{\partial (\mathbf{W}_{ax} \cdot \mathbf{x}_i)}{\partial \mathbf{W}_{ax}} + \mathbf{W}_{aa} \cdot R_{i-1} \right) \quad ; \quad R_0 = 0 \end{aligned} \quad (6.12)$$

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 bắt kè 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 bắt kè 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.

Mạng 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 vanishing/exploding gradient (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 vanishing/exploding gradient 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.

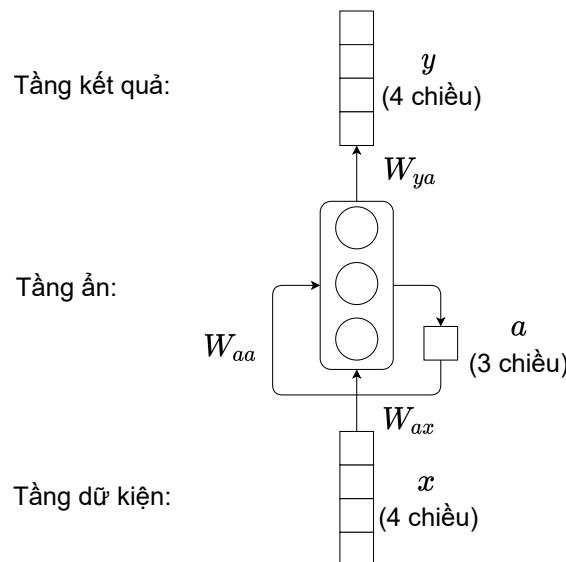
6.6 Ví dụ minh họa mạng RNN

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 một 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:

$$H \rightarrow \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad E \rightarrow \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad L \rightarrow \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad O \rightarrow \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix},$$

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 $\hat{y}^{<t>}$ 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 $\hat{y}^{<t>}$ 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 $a^{<1>}$, $\hat{y}^{<1>}:$

$$a^{<1>} = g_1(W_{aa}a^{<0>} + W_{ax}x^{<1>} + b_a)$$

$$= \tanh\left(\begin{pmatrix} .3 & .8 & .2 \\ .6 & .4 & .1 \\ .4 & .3 & .9 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} .1 & .3 & .2 & .3 \\ .1 & .7 & .3 & .2 \\ .4 & .5 & .8 & .6 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + 0.6\right)$$

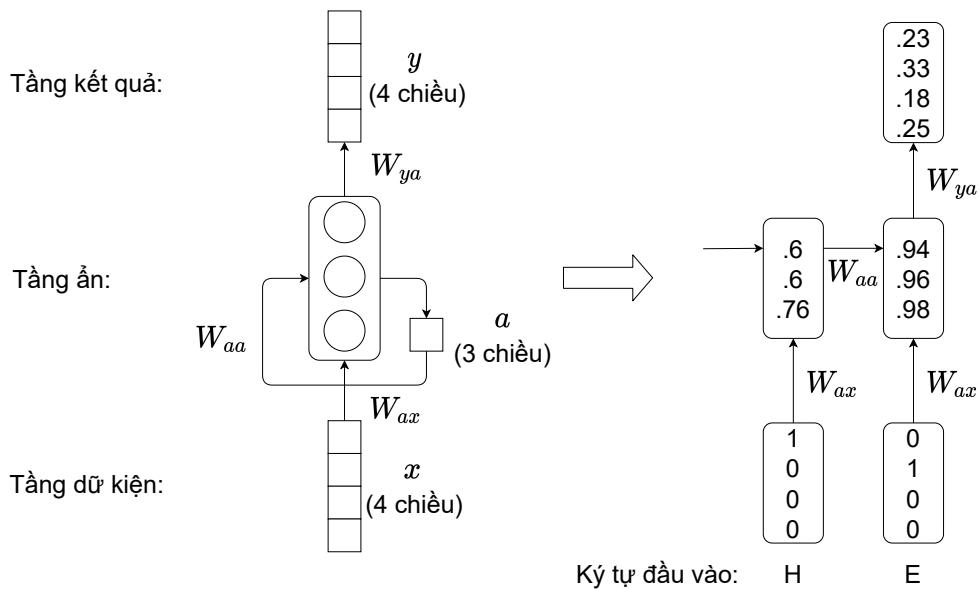
$$= \tanh\left(\begin{pmatrix} .7 \\ .7 \\ 1. \end{pmatrix}\right) = \begin{pmatrix} .6 \\ .6 \\ .76 \end{pmatrix}$$

$$\begin{aligned}\hat{y}^{<1>} &= g_2(W_{ya}a^{<1>} + b_y) \\ &= \text{softmax} \left(\begin{pmatrix} .4 & .2 & .6 \\ .9 & .1 & .5 \\ .2 & .2 & .6 \\ .1 & .8 & .4 \end{pmatrix} \cdot \begin{pmatrix} .6 \\ .6 \\ .76 \end{pmatrix} + .6 \right) = \begin{pmatrix} .24 \\ .29 \\ .21 \\ .26 \end{pmatrix}\end{aligned}$$

Tiếp tục sử dụng tính toán tương tự ta tính được:

$$\begin{aligned}a^{<2>} &= (.94 \quad .96 \quad .98)^T \\ \hat{y}^{<2>} &= (.23 \quad .33 \quad .18 \quad .25)^T\end{aligned}$$

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 $\hat{y}^{<2>} = [.23 \quad .33 \quad .18 \quad .25]$. 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} :

$$W_{ax} \longrightarrow (W_{ax} | b_a) \longrightarrow \begin{pmatrix} 0.1 & 0.3 & 0.2 & 0.3 & 0.6 \\ 0.1 & 0.7 & 0.3 & 0.2 & 0.6 \\ 0.4 & 0.5 & 0.8 & 0.6 & 0.6 \end{pmatrix}$$

Chọn hệ số học $\eta = 1.0$, cập nhật W_{ax} theo công thức:

$$\begin{aligned} W_{ya} &= W_{ya} - \eta(\hat{y}^{<2>} - y^{<2>}) \otimes a^{<2>} \\ &= \begin{pmatrix} 0.4 & 0.2 & 0.6 & 0.6 \\ 0.9 & 0.1 & 0.5 & 0.6 \\ 0.2 & 0.2 & 0.6 & 0.6 \\ 0.1 & 0.8 & 0.4 & 0.6 \end{pmatrix} - 1.0 \left(\begin{pmatrix} .23 \\ .33 \\ .18 \\ .25 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right) \otimes \begin{pmatrix} .94 \\ .96 \\ .98 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 0.4 & 0.2 & 0.6 & 0.6 \\ 0.9 & 0.1 & 0.5 & 0.6 \\ 0.2 & 0.2 & 0.6 & 0.6 \\ 0.1 & 0.8 & 0.4 & 0.6 \end{pmatrix} - \begin{pmatrix} 0.2162 & 0.2208 & 0.2254 & 0.23 \\ 0.3102 & 0.3168 & 0.3234 & 0.33 \\ -0.7708 & -0.7872 & -0.8036 & -0.82 \\ 0.235 & 0.24 & 0.245 & 0.25 \end{pmatrix} \\ &= \begin{pmatrix} 0.1838 & -0.0208 & 0.3746 & 0.37 \\ 0.5898 & -0.2168 & 0.1766 & 0.27 \\ 0.9708 & 0.9872 & 1.4036 & 1.42 \\ -0.135 & 0.56 & 0.155 & 0.35 \end{pmatrix} \end{aligned}$$

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 $\hat{y}^{<t>}$ chính là giá trị dự đoán sau khi có t ký 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 x, a, 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 \hat{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.

Sử dụng những ma trận trọng số sau đây:

$$W_{ax} = \begin{bmatrix} 0.48 & 0.75 & 0.43 & 0.62 & 0.61 & 0.88 \\ 0.17 & 0.91 & 0.08 & 0.33 & 0.5 & 0.33 \end{bmatrix}$$

$$W_{aa} = \begin{bmatrix} 0.35 & 0.26 \\ 0.62 & 0.81 \end{bmatrix}$$

$$W_{ya} = \begin{bmatrix} 0.74 & 0.66 \\ 0.58 & 0.77 \\ 0.22 & 0.38 \\ 0.67 & 0.45 \\ 0.85 & 0.83 \\ 0.34 & 0.43 \end{bmatrix}$$

$$b_y = b_a = 0.65$$

Chương 7

Mô hình ngôn ngữ (Language model)

7.1 Mô hình hóa ngôn ngữ

7.1.1 Giới thiệu

Từ những buổi bình minh của thời đại máy móc và tự động hóa 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 định* (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 suất (*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:

$$\begin{aligned} P(w_1, w_2, \dots, w_n) &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)P(w_4|w_{1:3})\dots P(w_n|w_{1:n-1}) \\ &= \prod_{i=1}^n P(w_i|w_{1:i-1}). \end{aligned} \tag{7.1}$$

Ví dụ:

$$\begin{aligned} P(\text{"chúng ta không thuộc về nhau"}) \\ = P(\text{chúng}) \times P(\text{ta}|\text{chúng}) \times P(\text{không}|\text{chúng ta}) \times P(\text{thuộc}|\text{chúng ta không}) \\ \times P(\text{về}|\text{chúng ta không thuộc}) \times P(\text{nhau}|\text{chúng ta không thuộc về}) \end{aligned}$$

Công thức (7.1) có thể biến đổi thành:

$$P(w_i|w_1, w_2, \dots, w_{i-1}) = \frac{P(w_1, w_2, \dots, w_{i-1}, w_i)}{P(w_1, w_2, \dots, w_{i-1})} \propto P(w_1, w_2, \dots, w_{i-1}, w_i) \quad (7.2)$$

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:

$$P(\text{"chúng ta không thuộc về } \underline{\underline{\underline{\quad}}}\text{"}).$$

Giả sử trong hai từ có khả năng nhất là {"đâu", "nhau"}. Khi đó, so sánh giữa:

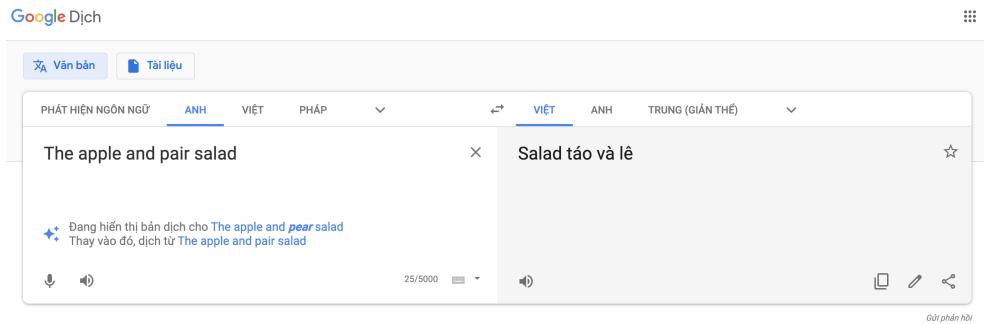
$$P(\text{đâu}| \text{chúng ta không thuộc về}) < P(\text{nhau}| \text{chúng ta không thuộc về}).$$

Việc này tương đương với so sánh:

$$P(\text{chúng ta không thuộc về đâu}) < P(\text{chúng ta không thuộc về nhau}).$$

Mô hình ngôn ngữ có thể ứng dụng vào rất nhiều việc, lĩnh vực:

- Dịch máy: $P(\text{high winds tonite}) > P(\text{large winds tonite})$.
- Sửa lỗi chính tả: ví dụ câu "The office is about fifteen minuets from my house".
Mô hình ngôn ngữ cho ta biết: $P(\text{about fifteen miinutes from}) > P(\text{about fiIeen minuets from})$. Do đó có thể dùng để sửa được chính tả cho câu.
- Nhận biết giọng nói: $P(\text{The apple and pear salad}) >> P(\text{The apple and pair salad})$. 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:

$$P(\text{"Chúng ta không thuộc về nhau"}) = \frac{\text{count}(\text{Chúng ta không thuộc về nhau})}{\text{count}(\text{Tất cả các câu được nói/viết})}$$

Hoặc với xác suất likelihood thì:

$$P(\text{nhau} | \text{chúng ta không thuộc về}) = \frac{\text{count}(\text{chúng ta không thuộc về nhau})}{\text{count}(\text{chúng ta không thuộc về})}$$

Tuy nhiên, cách ước lượng này 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:

$$P(w_i | w_1, w_2, \dots, w_{i-1}) = P(w_i | w_{i-1})$$

Ví dụ: $P(\text{nhau} | \text{chúng ta không thuộc về}) = P(\text{nhau} | \text{về})$.

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:

$$P(w_i|w_1, w_2, \dots, w_{i-1}) = P(w_i|w_{1:i-1-k})$$

Mặc dù, 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 đến các từ đầu tiên và ngoài khoảng k (**long-distance dependencies**). Tuy nhiên, mô hình hóa ngôn ngữ dựa vào tính chất Markov vẫn là một cách hữu hiệu và đưa ra được một mô hình hóa mạnh với một số k hợp lý.

Mô hình n -gram

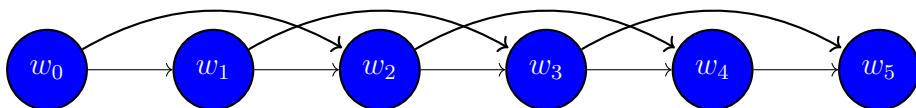
Dựa trên tính chất Markov, ta có mô hình ngôn ngữ n -gram (n -gram language model): gán xác suất dựa vào n từ (n -gram). Ý tưởng về n -gram khởi nguồn từ những công trình Lý thuyết thông tin (*Information Theory*) của *Claude Shannon* khi ông đặt câu hỏi về cho một chuỗi các ký tự thì xác suất xuất hiện (*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.

$$P(w_1 w_2 \dots w_n) \approx \prod_{i=1}^n P(w_i)$$

Tiếp đến là **Bigram model**: từ dự đoán phụ thuộc vào một từ trước đó.

$$P(w_i|w_1 w_2 \dots w_{i-1}) \approx \prod_{i=1}^n P(w_i|w_{i-1})$$



Hình 7.2: Bigram Language Model

Dựa vào Hình 7.2, ta có thấy:

- $P(w_1|w_0) = P(w_1|w_0)$
- $P(w_2|w_1w_0) = P(w_2|w_1)$
- $P(w_3|w_2w_1w_0) = P(w_3|w_2)$
- ...
- $P(w_k|w_{k-1}...w_2w_1w_0) = P(w_k|w_{k-1})$

Từ đó ta có thể tính

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_i, w_{i-1})}{\text{count}(w_{i-1})} \quad (7.3)$$

Ví dụ: Giả sử câu bắt đầu với $\langle s \rangle$ và kết thúc với $\langle /s \rangle$. Xét 3 câu sau:

$\langle s \rangle$ I am Sam $\langle /s \rangle$

$\langle s \rangle$ Sam I am $\langle /s \rangle$

$\langle s \rangle$ I do not like green eggs and ham $\langle /s \rangle$

Ta có:

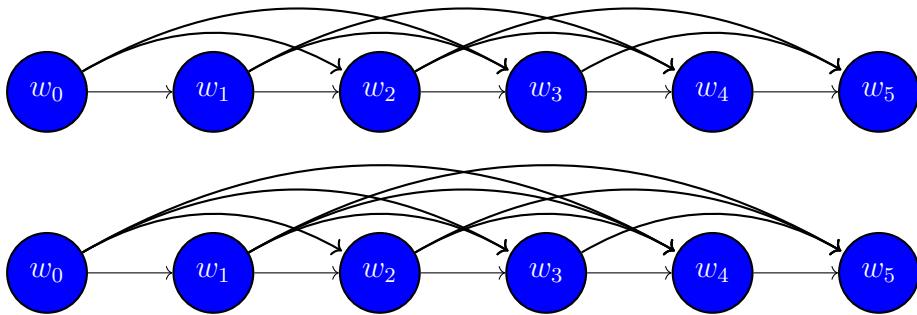
$$P(I|\langle s \rangle) = \frac{2}{3}, P(Sam|\langle s \rangle) = \frac{1}{3}, P(am|I) = \frac{2}{3},$$

$$P(\langle /s \rangle | Sam) = \frac{1}{2}, P(Sam|am) = \frac{1}{2}, P(do|I) = \frac{1}{3}.$$

Đối với **Trigram model** thì:

$$P(w_i|w_{i-1}w_{i-2}) = \frac{\text{count}(w_i, w_{i-1}, w_{i-2})}{\text{count}(w_{i-1}, w_{i-2})}$$

Chúng ta có thể mở rộng ra cho trigram, 4-gram,...như mô tả trong Hình 7.3.



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 variance 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, \dots, w_n và một mô hình ngôn ngữ gán xác suất cho một từ dựa vào lịch sử, Perplexity được tính như sau:

$$PP(p) = 2^{H(p)} = 2^{-\sum_x p_x \log_2 p_x} \quad (7.4)$$

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:

$$H(p) \approx -\frac{1}{N} \log_2 p(W) = -\frac{1}{N} \log_2 p(w_1, w_2, \dots, w_N)$$

Do đó ta có thể biến đổi công thức 7.4 thành:

$$\begin{aligned}
 PP(p) &= 2^{-\frac{1}{N} \log_2 p(w_1, w_2, \dots, w_N)} \\
 &= (2^{\log_2 p(w_1, w_2, \dots, w_N)})^{-\frac{1}{N}} \\
 &= p(w_1, w_2, \dots, w_N)^{-\frac{1}{N}} \\
 &= \sqrt[n]{\frac{1}{p(w_1, w_2, \dots, w_N)}}
 \end{aligned} \tag{7.5}$$

Ví dụ: cho một chuỗi số ngẫu nhiên từ 0->9, xác suất xuất hiện của từ tiếp theo biết xác suất xuất hiện của mỗi từ là 1/10 thì perplexity của mô hình này là:

$$PP(p) = p(w_1, w_2, \dots, w_N)^{\frac{1}{N}} = \left(\frac{1}{10}\right)^{\frac{1}{N}} = \frac{1}{10}^{-1} = 10$$

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):

Bảng 7.1: Perplexity tương ứng với các n -gram khác nhau

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Như chúng ta đã biết, Perplexity 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" lầm 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. Có thể nói rằng mô hình không chắc chắn trong việc dự đoán 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 dù 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 thế 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 có 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:

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}. \quad (7.6)$$

Giả sử chuỗi $\langle w_i, w_{i-1} \rangle$ không xuất hiện trong tập huấn luyện thì $\text{count}(w_{i-1}, w_i) = 0$ 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 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.

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i) + 1}{\text{count}(w_{i-1}) + |V|} \quad (7.7)$$

Trong đó $|V|$ là kích thước tập từ vựng, 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 $0 < \alpha \leq 1$:

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i) + \alpha}{\text{count}(w_{i-1}) + \alpha|V|} \quad (7.8)$$

Với kỹ thuật làm trơn add-1 trong công thức (7.7), ta có thể hoàn nguyên lại số (count) của các từ:

$$\text{count} * (w_{i-1}, w_i) = \frac{(\text{count}(w_{i-1}, w_i) + 1) * \text{count}(w_{i-1})}{\text{count}(w_{i-1}) + |V|} \quad (7.9)$$

	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

Hình 7.4: Thống kê count của bigram chưa được làm trơn

	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 7.5: Thống kê count của bigram đã được làm trơn bằng ước lượng add-1

Dễ dàng nhận thấy rằng Bảng 7.4 xuất hiện nhiều count có giá trị là 0, trong khi đó; Bảng 7.5 chứa các giá trị đã qua quá trình 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:

$$P(w_{i+1}|w_{i-k:i}) = \beta * \frac{\text{count}(w_{i-k:i+1})}{\text{count}(w_{i-k:i})} + (1 - \beta)P(w_{i+1}|w_{i-(k-1):i}). \quad (7.10)$$

Phép làm trơn trong công thức 7.10 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 trộn, ta có:

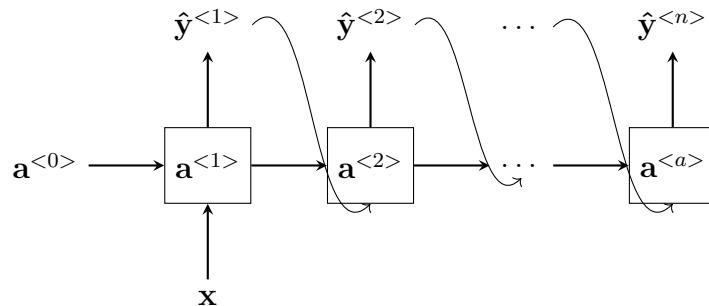
$$\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1 P(w_n|w_{n-1}w_{n-2}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)$$

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

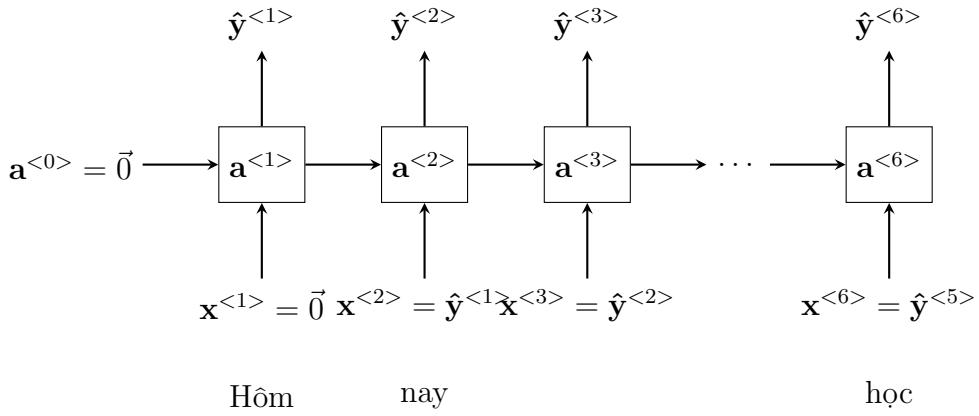
7.2.1 Ý tưởng cơ bản

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. Để thấy mạng RNN được sử dụng với ý tưởng trên là một mạng one-to-many RNN như Hình 7.6.



Hình 7.6: 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 7.7.



Hình 7.7: Mô hình đơn giản thể hiện ý tưởng huấn luyện Language Model với RNN

Trong Ví dụ 7.2.1, ở đầu câu, chúng ta chưa có từ đứng trước, vì thế ta sẽ truyền vào $x^{<1>} = \vec{0}$ và $a^{<0>} = \vec{0}$ là vector 0 để khởi động cho mạng với ý nghĩa đầu vào là từ “mở đầu câu”, như vậy:

- $\hat{y}^{<1>} = \text{xác suất từ “Hôm” là xuất hiện ở đầu câu } P(Hôm),$
- $\hat{y}^{<2>} = \text{xác suất từ “nay” xuất hiện sau từ “Hôm” ở đầu câu } P(nay | Hôm),$
- $\hat{y}^{<3>} = \text{xác suất từ “tôi” xuất hiện sau “Hôm nay” } P(tôi | Hôm nay),$
- $\hat{y}^{<4>} = \text{xác suất từ “đi” xuất hiện sau “Hôm nay tôi” } P(đi | Hôm nay tôi),$
- $\hat{y}^{<5>} = \text{xác suất từ “học” xuất hiện sau “Hôm nay tôi đi” } P(học | Hôm nay tôi đi),$
- $\hat{y}^{<6>} = \text{xác suất câu kết thúc tại “Hôm nay tôi đi học” } P(<EOS> | Hôm nay tôi đi học),$ 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:

$$\begin{aligned} P(y^{<1>}, y^{<2>}, y^{<3>}) &= P(y^{<1>})P(y^{<2>} | y^{<1>})P(y^{<3>} | y^{<1>}, y^{<2>}) \\ &= \hat{y}^{<1>}\hat{y}^{<2>}\hat{y}^{<3>} \end{aligned}$$

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 vector 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 vector 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 $\langle \text{NUM} \rangle$.
- các từ ít gặp thay bằng $\langle \text{UNK} \rangle$.
- thêm các token đặc biệt như $\langle \text{EOS} \rangle$ - 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 $\langle \text{EOS} \rangle$. 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.

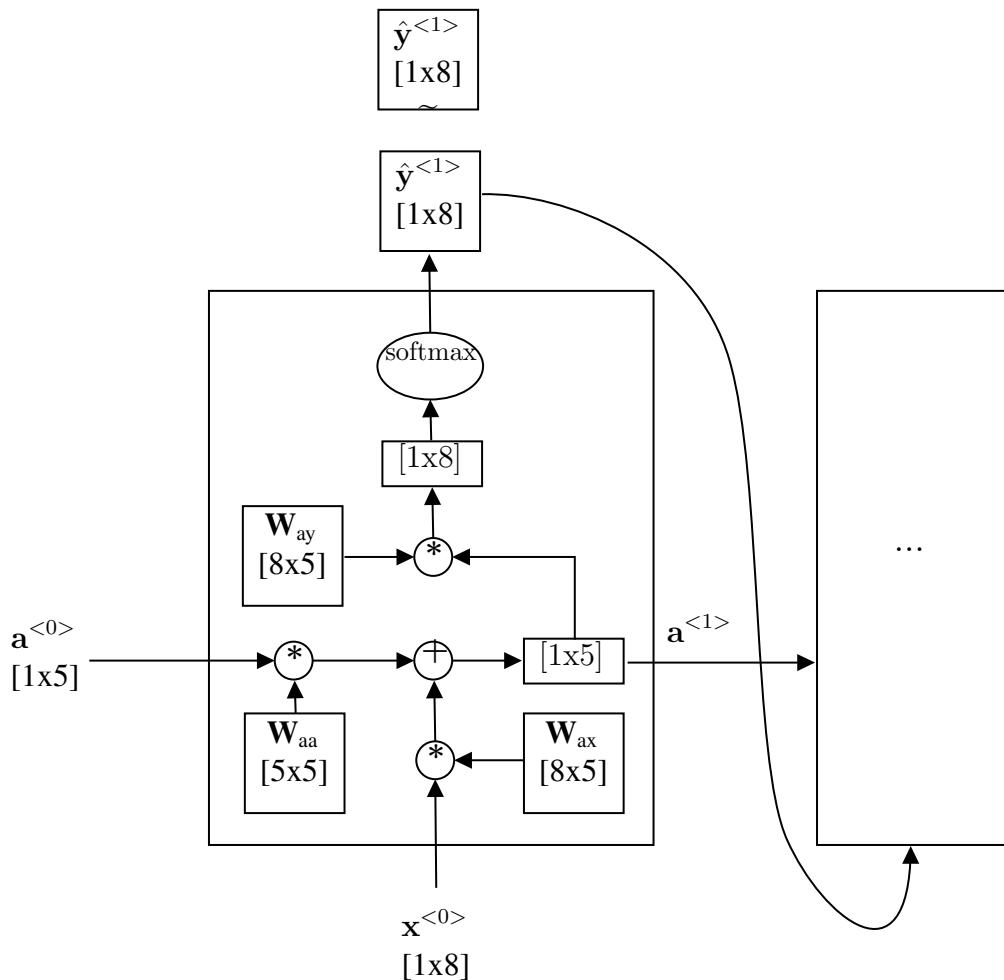
Ví dụ 7.2.2. Huấn luyện Language Model với tài liệu:

"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 1×8 . Input $x^{<0>}$ sẽ là một véc-tơ 0 với số chiều phụ thuộc vào corpus, trong trường hợp này là 1×8 . 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 1×5 .

Khi đó, W_{aa} là một ma trận 5×5 , W_{ax} là một ma trận 8×5 . Đầu ra của phép tính $W_{aa}a^{<i>} + W_{ax}x^{<i>}$ là một vec-tơ 1×5 , trong khi đó ta cần kết quả \hat{y} là một vec-tơ 1×8 để so sánh được với các vec-tơ trong corpus. Do đó, W_{ay} sẽ là ma trận 5×8 .

Ta chọn hàm kích hoạt là hàm softmax để chuyển kết quả về khoảng $[0, 1]$, tức



Hình 7.8: Minh họa Ví dụ 7.2.2, huấn luyện Language Model với RNN

thành xác suất rời rạc 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.

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:

$$\mathcal{L} = \sum_t \mathcal{L}(\hat{y}^{}, y^{})$$

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 Neural Network) đã được huấn luyện trước, có thể kể đến như là: Greedy Search, Beam 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 kỳ

Tạo mẫu bất kỳ 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 phân 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 có 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:

$$F(x) = P(X \leq x)$$

trong đó vé 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.

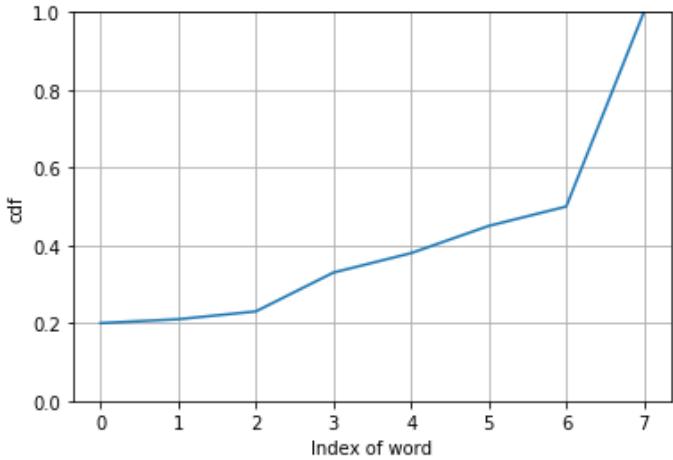
$$P(x) = [0.2, 0.01, 0.02, 0.1, 0.05, 0.07, 0.05, 0.5]$$

với mỗi xác suất trong P tương ứng với các từ

$$["Hom", "nay", "toi", "di", "hoc", "Ngay", "mai", "cung"]$$

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 7.9: 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 nhiên từ phân bố đều trên đoạn $[0, 1]$. Chúng ta dễ dàng phát sinh các từ trên theo công thức.

$$O(x) = \begin{cases} "Hom" & \text{if } 0 \leq x < 0.2 \\ "nay" & \text{if } 0.2 \leq x < 0.21 \\ "toi" & \text{if } 0.21 \leq x < 0.23 \\ "di" & \text{if } 0.23 \leq x < 0.33 \\ "hoc" & \text{if } 0.33 \leq x < 0.38 \\ "Ngay" & \text{if } 0.38 \leq x < 0.45 \\ "mai" & \text{if } 0.45 \leq x < 0.5 \\ "cung" & \text{if } 0.5 \leq x \leq 1.0 \end{cases}$$

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

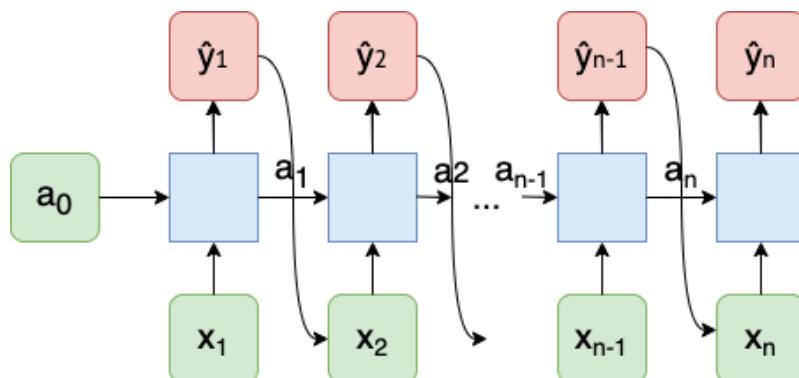
Từ những hiểu biết trên, ta có thể thiết kế một mạng RNN dùng để tự động viết tiếp chuỗi đã được cho trước dựa vào corpus có sẵn và mạng RNN đã được huấn luyện từ trước.

Mô tả bài toán

Dầu 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 7.10: Cấu trúc mạng RNN

Hình 7.10 miêu tả quá trình sinh chuỗi sử dụng mạng RNN được huấn luyện sẵn. Giả sử ta huấn 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: ["Dâu", "lòng", "hai", "ả", "tố", "nga",

"Thúy", "Kiều", $<SOS>$, $<EOS>$]. Trong đó $<SOS>$ là start of string, $<EOS>$ là end of string.

Dầu 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.

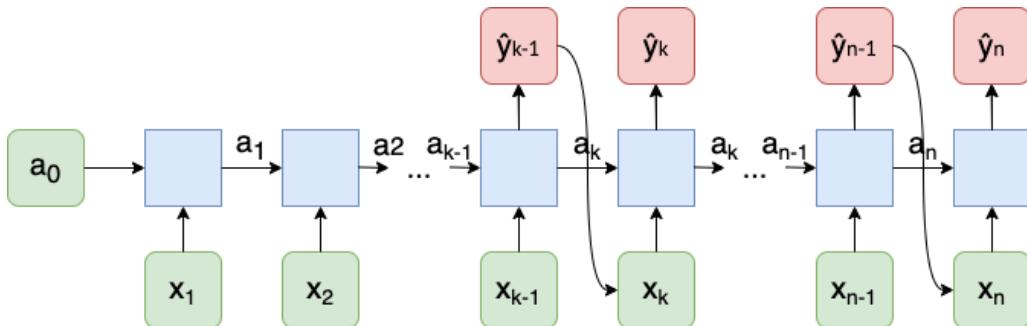
Thực hiện forward bước đầu, ta thu được trạng thái a_1 và một phân phối xác suất ở ngõ ra \hat{y}_1 . Phân phối xác suất này cho ta biết xác suất xuất hiện của từ tiếp theo trong câu. Ví dụ, nếu phân phối xác suất ở ngõ ra \hat{y} có dạng:

$$\hat{y} = [0.2, 0.3, 0.05, 0.05, 0.01, 0.1, 0.09, 0.05, 0.05, 0.1]$$

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 "*Dầu*", ta sẽ gán $x_2 = "Dầu"$ 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: "*Dầu lòng hai ả tố nga <EOS>*". Câu văn kết thúc khi $<EOS>$ được sinh ra.

Sinh chuỗi với các từ được gợi ý



Hình 7.11: Cấu trúc mạng RNN 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 $\langle SOS \rangle$, ta sẽ đưa vào $[x_1, x_2, x_3, \dots, x_k]$ k từ đầu tiên được gọi ý.

Ví dụ ta gọi 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 gọi ý, nên \hat{y}_2 trong trường hợp này sẽ được nhận làm x_3 và a_2 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 gọi 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 gọi 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 *CBO*W để ghi lại ngữ cảnh của từ). Đôi khi, có một khả năng từ được sinh ra là token $\langle UNK \rangle$ (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. Mang 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:

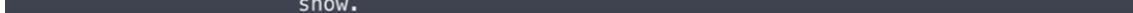
Input text: Malfoy hadn't noticed anything.

GPT2 Continuation: "In that case," said Harry, after thinking over it, "I suggest you return to the library."
He then walked over to a bookcase at the back of the library and pawed through it. For a few seconds he stared at it, almost careening through the counter, then he turned and went back to the bookshelves and started up again, but in a little while a new book had appeared. It was Hogwarts and Its History by Gilderoy Lockhart. He let out a long, soft sigh of relief and seized it with both hands.
Hagrid strode into the room, his large feet gleaming in the lamps, a mixture of shock and awe on his face.
"Professor — yes?"
"Hello, Hagrid, it's good to see you," said Harry, but he also knew that whatever he'd written in the note would be read by everyone in the school — Professor McGonagall, the Headmistress, the rest of the teachers, and even some third-year students who'd just started class.
Hagrid climbed carefully over the stack of books and opened the book, shining gold letters high in

Hình 7.12: 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:

SYSTEM PROMPT (HUMAN-WRITTEN)	<i>In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.</i>
MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)	<p>The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.</p> <p>Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.</p> <p>Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.</p>
	

Hình 7.13: 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'thay được chiến thắng trong cuộc đua vô địch wimbledon N đã có mă
t tại trận chung kết giải quần vợt thế'giới với tỷ số'N N N N N N trong trận đấu này đã có N lận thi đấu vào từ k
ết us open N trước khi giành chiến thắng N N N N trước tay vợt người serbia vừa chiếm tỷ lệ N N N N trong khi đ
ó trận đấu ra sân của mình và đối thủ của federer đã có N lận gặp nhau'

In [22]: generate_text('novak djo', 400)
Out[22]: 'novak djokovic và chuyênl sang thi đấu trên sân cỏ những cú đánh thuận tay khi trở lại thi đấu chuyên nghiệp và đó
là một chán 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 sam querry và colombia 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

In [27]: generate_text('andy murr', 400)
Out[27]: 'andy murray đã có mặt tại các trận đấu như vậy thi chỉ cần thi đấu không thể'trở lại trong năm nay khi thi đấu chỉ
có thể'trở lại thi mới có thể'nhận thấy thực sự cho thấy sự có'gâng 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 chuyênl đổi sô'thuê' thu nhập của tập đoàn là N triệu usd trong tháng N theo báo cáo của uber c
ho biết thêm các tổ'chức tin dụng của hệ thống tài chính tại vietnam 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'thay rằng những thách thức trong các nền kinh tế'
apec cho thấy thị trường chứng khoán vietnam có thể'sẽ tăng'

In [29]: generate_text('apple', 400)
Out[29]: 'apple cho biết chúng tôi cần phải có thêm một sô'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 vietnam cho biết họ đã chọn thời gian tham gia các hoạt động của các nhà trưởn
g trong trường học thuộc các ngành nghề'theo học ban 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 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 7.14: Mạng RNN viết báo tiếng Việt

```

<|startoftext|>Hôm nay Trọng đến tìm Kiều,  

Thiếp hồng tim đến hương khuê gửi vào.<|endoftext|>  

<|startoftext|>Trưởng tơ giáp mặt hoa đào,  

Vè nào chẳng mặn nét nào chẳng ưa?<|endoftext|>  

<|startoftext|>Hải đường mơn mởn cành tơ,  

Ngày xuân càng gió càng mưa càng nồng.<|endoftext|>  

<|startoftext|>Nguyệt hoa hoa nguyệt nao nùng,  

Đêm xuân ai dẽ cầm lòng được chẳng?<|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 thể 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 bể hoa,  

=====  

<|startoftext|>Hôm nay Trọng đến tìm Kiều,  

Thiếp hồng tim 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 húc bén mùi,  

Giêng 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 rääng,  

Tiểu thư trước ♀  

=====  

<|startoftext|>Hôm nay Trọng đến tìm Kiều,  

Thiếp hồng tim đến hương án phòng.<|endoftext|>  

<|startoftext|>Trở về mình bạch nói tường:  

Mặt nàng chẳng thấy việc nàng đã tra.<|endoftext|>  

<|startoftext|>Người này nặng kiếp oan gia,  

Còn nhiều nợ lầm 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 chiến chiến,  

Muốn  

=====  

<|startoftext|>Hôm nay Trọng đến tìm Kiều,  

Cảm 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 thẳng 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 7.15¹⁸⁴. 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 sửa 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:

$$P1 = P(\text{The}) \quad (7.11)$$

$$P2 = P(\text{apple}|\text{The}) * P1 \quad (7.12)$$

$$P3 = P(\text{and}|\text{The apple}) * P2 \quad (7.13)$$

$$P4 = P(\text{pair}|\text{The apple and}) * P3 \quad (7.14)$$

$$P5 = P(\text{salad}|\text{The apple and pair}) * P4 \quad (7.15)$$

$$P(\text{The apple and pair salad}) = P5 \quad (7.16)$$

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:

$$P(The) = \frac{25,000,000}{100,000,000} = 0.25 \quad (7.17)$$

$$P(The\ apple) = \frac{10,000}{25,000,000} * 0.25 = 10^{-4} \quad (7.18)$$

$$P(The\ apple\ and) = \frac{3,000}{10,000} * 10^{-4} = 3 * 10^{-5} \quad (7.19)$$

$$P(The\ apple\ and\ pair) = \frac{5}{3,000} * 3 * 10^{-5} = 5 * 10^{-8} \quad (7.20)$$

$$P(The\ apple\ and\ pear) = \frac{250}{3,000} * 3 * 10^{-5} = 2.5 * 10^{-6} \quad (7.21)$$

$$P(The\ apple\ and\ strawberry) = \frac{400}{3,000} * 3 * 10^{-5} = 4 * 10^{-6} \quad (7.22)$$

$$P(The\ apple\ and\ pair\ salad) = \frac{1}{5} * 5 * 10^{-8} = 10^{-8} \quad (7.23)$$

$$P(The\ apple\ and\ pear\ salad) = \frac{160}{250} * 2.5 * 10^{-6} = 1.6 * 10^{-6} \quad (7.24)$$

$$P(The\ apple\ and\ strawberry\ salad) = \frac{220}{400} * 4 * 10^{-6} = 2.2 * 10^{-6} \quad (7.25)$$

Ta có thể thấy xác suất khi xuất hiện từ "*pair*" (công thức 7.20) khá thấp so với "*pear*" (công thức 7.21) và "*strawberry*" (công thức 7.22). 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:

$$Threshold(The\ apple\ and\ *) = \frac{1}{100} * P(The\ apple\ and) = 3 * 10^{-7} \quad (7.26)$$

Vì xác suất khi xuất hiện của từ "*pair*" thấp hơn threshold đã tính (công thức 7.26) nên ta sẽ nghi ngờ "*pair*" là từ sai. Và ta cũng thấy xác suất của câu "*The apple and pair salad*" (công thức 7.23) thấp hơn hẳn "*The apple and pear salad*" (công thức 7.24) và "*The apple and strawberry salad*" (công thức 7.25).

Với trường hợp tính cho cụm từ ngắn:

$$P(\text{and}) = \frac{85,000,000}{100,000,000} = 0.85 \quad (7.27)$$

$$P(\text{and pair}) = \frac{13,000}{85,000,000} * 0.85 = 1.3 * 10^{-4} \quad (7.28)$$

$$P(\text{and pear}) = \frac{9,000}{85,000,000} * 0.85 = 0.9 * 10^{-4} \quad (7.29)$$

$$P(\text{and strawberry}) = \frac{11,000}{85,000,000} * 0.85 = 1.1 * 10^{-4} \quad (7.30)$$

Ta có thể thấy xác suất của "*and pair*" (công thức 7.28) còn cao hơn cả "*and pear*" (công thức 7.29) và "*and strawberry*" (công thức 7.30). 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 7.22) lớn hơn "*pear*" (công thức 7.21) và xác suất của câu "*The apple and strawberry salad*" (công thức 7.25) lớn hơn "*The apple and pear salad*" (công thức 7.24), 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 đồng 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*:

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases} \quad (7.31)$$

Trong đó:

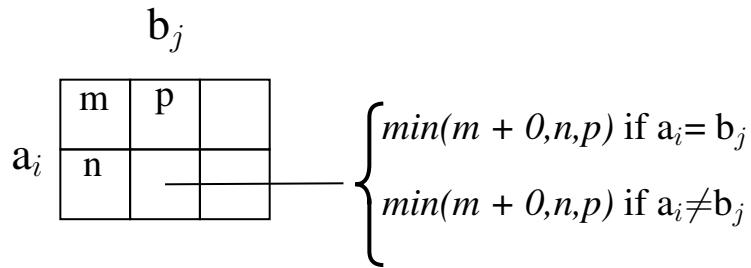
- a, b là hai từ dùng để đo khoảng cách giữa chúng.
- i, j là chỉ số ký tự đang xét đến của a và b .
- $1_{(a_i \neq b_j)}$ là một hàm đặc trưng trả về 0 nếu $a_i = b_j$ và trả về 1 trong trường hợp ngược lại.
- $lev_{a,b}(i,j)$ là khoảng cách giữa i ký tự đầu tiên của a và j ký tự đầu tiên của b .
- 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 7.16).



Hình 7.16: 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

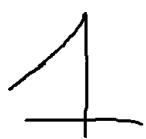
Ta có thể thấy khoảng cách phát âm giữa "*pear*" (pe-a) và "*pair*" (pe-a) là 0 và giữa "*strawberry*" (stro-be-ri) và "*pair*" (pe-a) là 7, vì thế mô hình sẽ chọn "*pear*" là từ thay thế.

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 7.17).



(a) 1



(b) 4

Hình 7.17: Số 1 và số 4 viết tay có nét tương đồng nhau.

- Nhầm lẫn giữa "9", "g" và "q" (Hình 7.18).



Hình 7.18: 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 7.31) 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 7.19).

This is an apple.

Hình 7.19: 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): *example* -> *ezample*
- Lỗi nhập sai thứ tự các ký tự trong từ (transpose): *example* -> *exmapel*
- Lỗi nhập thiếu ký tự trong từ (delete): *example* -> *exampe*
- Lỗi nhập thừa ký tự (insert): *example* -> *exaample*

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 "*example*" để tìm tập các ứng cử viên thay thế như sau:

- Replace -> **a**xampe, **b**xampe, ..., eaampe, ebampe, ..., exampy, exampz
- Transpose -> **x**exampe, eaxmpe, exmape, exapme, examep
- Delete -> xampe (_xampe), eampe (**e_ ampe**), exmpe (**ex_ mpe**), exape (**exa_ pe**), exame (**exam_ e**), examp (**exam_**)
- Insert -> aexampe, bexampe, ..., eaxampe, ebxampe, ..., example, exampme, ..., exampey, exampez

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 tối 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 7.31) để tính khoảng cách từ vựng với từ sai và tìm từ thay thế.

Phụ lục A

Bảng 5 là một số thuật ngữ có đề cập trong sách:

Bảng 5: Bảng các thuật ngữ

Thuật ngữ	Ý nghĩa
activation	kích hoạt
activation function	hàm kích hoạt
Auto-Encoder	một kỹ thuật nhúng từ
average pooling	phép gộp lấy giá trị trung bình
axon	sợi trực
back propagation	lan truyền ngược
bias	độ lệch
binary classification	phân loại nhị phân
branching factor	hệ số phân nhánh
cell body	thân nơ-ron
classification	phân loại
computation graph	đồ thị tính toán
context word	từ ngữ cảnh
contextual information	thông tin ngữ cảnh
Continuous Bag of Word (CBOW)	một phương pháp word2vec
convex function	hàm lồi
convolution	phép tích chập
Convolutional Neural Network	mạng nơ-ron tích chập
corpus	kho ngữ liệu
cost function	hàm chi phí
cumulative distribution function	hàm phân phối tích lũy
dendrit	đuôi gai
design pattern	thiết kế mẫu
distribution pattern	phân bố mẫu
Edit Model	mô hình biến đổi từ
extrinsic evaluation	đánh giá ngoài

feature	đặc trưng
feature map	bản đồ thuộc tính
filter	bộ lọc
focus word	từ mục tiêu
forward propagation	làn truyền xuôi
fully-connected	kết nối đầy đủ
global minimum	cực tiểu toàn cục
GPU	Graphics Processing Unit
gradient descent	—
hidden layer	tầng ẩn
high-dimensional vector	vector nhiều chiều
hyper parameter	siêu tham số
input layer	tầng dữ kiện
intrinsic evaluation	đánh giá trong
local minimum	cực tiểu địa phương
label	nhãn
language model	mô hình ngôn ngữ
Laplace smoothing	kỹ thuật làm tròn Laplace
latent feature extraction	rút trích thuộc tính ẩn
layer	tầng
learning rate	tốc độ học
least square error (LSE)	bình phương cực tiểu
logistic regression	hồi quy logistic
long-distance dependency	sự phụ thuộc khoảng cách dài
loss function	hàm mất mát
low-dimensional vector	vector ít chiều
likelihood	xác suất xuất hiện
Linear Interpolation	nội suy tuyến tính
linear regression	hồi quy tuyến tính
Machine Translation	dịch máy
max pooling	phép gộp lấy giá trị lớn nhất

Maximum likelihood estimation (MLE)	ước lượng hợp lý cực đại
multilayer perceptrons (MLP)	mạng nơ-ron đa tầng
neural	nơ-ron
neural network	mạng nơ-ron
non-convex function	hàm không lồi
one-hot encoding	—
one-hot vector	—
output layer	tầng kết quả
overfitting	sự quá khớp
parameter	thông số
partial derivative	đạo hàm riêng
perceptron	nơ-ron nhân tạo
perceptron learning algorithm (PLA)	—
phrase	cụm từ
pixel	điểm ảnh
pooling	phép gộp
recurrent neural network	mạng nơ-ron truy hồi
regularization	chính quy hóa
sample	mẫu
sentence	câu
sentiment analysis	phân tích cảm xúc
Sequence data	dữ liệu dạng chuỗi
shared weight	chia sẻ trọng số
skip-gram	một phương pháp word2vec
smoothing	kỹ thuật làm trơn
Speech Recognition	nhận biết giọng nói
Spelling Correction	sửa lỗi chính tả
sum pooling	phép gộp lấy giá trị tổng
supervised learning	học có giám sát
synapse	khớp thần kinh
term	—

text	văn bản
text analyzer	phân tích văn bản
text classification	phân loại văn bản
text summarization	tóm tắt văn bản
token	—
threshold	ngưỡng
uniform distribution	phân phối đều
unique word	từ phân biệt
unsupervised learning	học không giám sát
vanishing gradient	đạo hàm suy biến
vocabulary	từ điển
weight	trọng số
weight matrix	ma trận trọng số
weighted branching factor	hệ số phân nhánh có trọng số
window	cửa sổ
window size	kích thước cửa sổ
word	từ
word embedding	kỹ thuật nhúng từ
word2vec	một kỹ thuật nhúng từ
zero-centered	???

Phụ lục B

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:

$$\begin{aligned} a_t &= \mathbf{F}(x_t, a_{t-1}, \theta) \\ &= g(W_{aa}a_{t-1} + W_{ax}x_t + b_a) \end{aligned} \quad (32)$$

g: Hàm truyền (tanh, sigmoid)

$W_{aa} \in \mathbb{R}^{n \times n}$: Ma trận trọng số để chuyển đổi từ trạng thái a_{t-1} qua trạng thái a_t
 $W_{ax} \in \mathbb{R}^{n \times D}$: Ma trận trọng số để chuyển đổi từ không gian đầu vào $x \in \mathbb{R}^D$ tới không gian trạng thái $a_t \in \mathbb{R}^n$

$$\hat{y}_t = f(W_{ya}a_t + b_y) \quad (33)$$

$$L = \sum_{t=1}^T L_t(\hat{y}_t) \quad (34)$$

$$\frac{\partial L}{\partial \theta} = \sum_{t=1}^T \frac{\partial L_t}{\partial \theta} \quad (35)$$

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 (32), đạo hàm của L đối với bộ tham số $\theta(W_{aa}, W_{ax}, ba)$ được tính bằng chain-rule như sau:

$$\frac{\partial L_t}{\partial \theta} = \sum_{k=1}^t \frac{\partial L_t}{\partial a_t} \frac{\partial a_t}{\partial a_k} \frac{\partial^+ a_k}{\partial \theta} \quad (36)$$

$\frac{\partial^+ a_k}{\partial \theta}$ biểu diễn đạo hàm riêng "tức thời" của a_k đối với θ , nghĩa là trong cách tính $\frac{\partial^+ a_k}{\partial \theta}$, giá trị a_{k-1} được xem là hằng số đối với θ

$$\frac{\partial a_t}{\partial a_k} = \prod_{i=k+1}^t \frac{\partial a_i}{\partial a_{i-1}} = \prod_{i=k+1}^t \text{diag}(g'(a_i))W_{aa} \quad (37)$$

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 $\gamma \in \mathbb{R}$ ($\gamma = 1$ 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 $W_a a^\top W_{aa}$), dựa vào định nghĩa và các tính chất của norm đối với ma trận (xem phụ lục phía dưới), dễ dàng chứng minh:

$$\begin{aligned} \left| \frac{\partial a_i}{\partial a_{i-1}} \right| &\leq |diag(g'(a_i))| |W_{aa}| \\ &\leq \gamma \sigma_1 \end{aligned} \tag{38}$$

$$\Rightarrow \left| \frac{\partial a_t}{\partial a_k} \right| = \left| \prod_{i=k+1}^t \frac{\partial a_i}{\partial a_{i-1}} \right| \leq (\gamma \sigma_1)^{t-k} \tag{39}$$

Thay (39) vào (36) (37), sử dụng định nghĩa norm của ma trận vào từng cột của ma trận Jacobian $\frac{\partial^+ a_k}{\partial \theta}$ (tương đương với đạo hàm "tức thời" của a_k với từng chiều θ_j trong bộ tham số θ), ta có:

$$\begin{aligned} \left| \frac{\partial L_t}{\partial \theta} \right| &= \left| \sum_{k=1}^t \frac{\partial L_t}{\partial a_t} \frac{\partial a_t}{\partial a_k} \frac{\partial^+ a_k}{\partial \theta} \right| \\ &\leq \sum_{k=1}^t \left[\left| \frac{\partial L_t}{\partial a_t} \right| \left| \frac{\partial a_t}{\partial a_k} \right| \right] \\ &\leq \sum_{k=1}^t \left[\left| \frac{\partial L_t}{\partial a_t} \right| (\gamma \sigma_1)^{t-k} \right] \end{aligned} \tag{40}$$

Khi $\sigma_1 < \frac{1}{\gamma}$, (i.e $(\gamma \sigma_1)^{t-k} = \eta^{t-k}$ với $\eta < 1$), theo công thức (40), 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 .0.1. Định nghĩa: Norm của ma trận $A \in \mathbb{R}^{m \times n}$ được định nghĩa như sau:

$$|A| = \sup \left\{ \frac{|Ax|}{|x|} : x \in \mathbb{R}^n, x \neq 0 \right\} \quad (41)$$

Định lý .0.1. Định lý

$$|A| = \sigma_{max} = \sigma_1$$

Where σ_1 là singular value lớn nhất của A

Chứng minh. Định lý .0.1 có thể chứng minh trực tiếp bằng khai triển Singular Value Decomposition.

$$\begin{aligned} |A| &= \operatorname{ess\ sup}_{x \neq 0} \frac{|Ax|}{|x|} = \operatorname{ess\ sup}_{x \neq 0} \frac{|U\sigma V^\top x|}{|x|} \\ &= \operatorname{ess\ sup}_{x \neq 0} \frac{|\sigma V^\top x|}{|x|} \\ &= \operatorname{ess\ sup}_{y \neq 0} \frac{|\sigma y|^2}{|V y|^2} \\ &= \operatorname{ess\ sup}_{y \neq 0} \frac{\left(\sum_{i=1}^r \sigma_i^2 |y_i|\right)^{1/2}}{\left(\sum_{i=1}^r |y_i|\right)^{1/2}} \\ &\leq \sigma_1 \end{aligned}$$

Dấu bằng xảy ra khi $y = [1 \ 0 \ \dots \ 0]^\top$, $|\sigma y| = \sigma_1$. Lúc đó $x = v_1$, vector đầu tiên trong tập các vector tạo nên cơ sở trực chuẩn V \square

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 Río, 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. (1982a). 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>
- Hopfield, J. J. (1982b). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8), 2554–2558.
- 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.

- Lecun Y., B. Y., Bottou L., & P., H. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*.
- 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>
- Lu Lu, Y. S., Yeonjong Shin, & Karniadakis, G. E. (2019). Dying relu and initialization: Theory and numerical examples. *arXiv:1903.06733*.
- 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.
- Minsky, M., & Papert, S. A. (1969). *Perceptrons: An introduction to computational geometry*. MIT Press.
- 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.