ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

Quản Thành Thơ

CÁC MÔ HÌNH HỌC SÂU NÂNG CAO

NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA
TP HỒ CHÍ MINH - 2023

To laoshi, Bao Bao and Dong Dong

Lời nói đầu

Trong thời đại ngày nay, *Trí Tuệ Nhân Tạo* (Artificial Intellgence) đang ngày càng trở nên phổ biến và ứng dụng rộng rãi trong nhiều lĩnh vực của cuộc sống. Đặc biệt, *học sâu* (Deep Learning) là một hướng tiếp cận đang phát triển mạnh mẽ, với nhiều ứng dụng đang được áp dụng trong thực tế. Trong các kiến trúc mạng học sâu, *Mạng nơ-ron tích chập* (Convolutional Neural Network - CNN) và *Mạng nơ-ron hồi quy* (Recurrent Neural Network - RNN) là hai dạng mạng học sâu phổ biến nhất và được sử dụng rộng rãi trong nhiều ứng dụng thực tế. CNN thường được sử dụng để học cách trích xuất đặc trưng của dữ liệu, đặc biệt là dữ liệu ảnh. Trong khi đó, RNN được sử dụng để xử lý dữ liệu chuỗi, chẳng hạn như ngôn ngữ tự nhiên, vì có khả năng lưu trữ thông tin từ quá khứ và sử dụng nó để dự đoán tương lai. Cả hai loại mạng này đều có ứng dụng rộng rãi trong nhiều lĩnh vực, bao gồm xử lý ngôn ngữ tự nhiên, dịch thuật, xử lý ảnh và video, và nhiều ứng dụng khác.

Được phát triển từ hai mô hình học sâu đang dần được coi là kinh điển trên, nhiều mô hình học sâu nâng cao khác cũng dần được giới thiệu và áp dụng thành công trong nhiều bài toán thực tế. Hiện nay, Trí Tuệ Nhân Tạo đang được áp dụng thành công nhất đối với dữ liệu có thể biểu diễn dạng *chuỗi* (sequence) như văn bản, hình ảnh, âm thanh và dữ liệu chuỗi thời gian. Theo hướng nghiên cứu này, có thể kể đến các dạng mở rộng của RNN như LSTM (Long Short-term Memory) và GRU (Gated Recurrent Unit), các mô hình Sequence-to-Sequence và cơ chế attention, kiến trúc Transformer và các *mô hình ngôn ngữ lớn* (large language model) như BERT (Bidirectional Encoder Representations from Transformers) và GPT (Generative Pre-trained Transformer). Đặc biệt, GPT chính là tiền thân của ứng dụng ChatGPT đang gây xôn xao trên toàn thế giới hiện nay. Hành trình nghiên cứu trong lĩnh vực xử lý ngôn ngữ tự nhiên để dẫn đến ChatGPT sẽ được tái hiện trong cuốn sách này.

Bên cạnh dữ liệu dạng chuỗi, dữ liệu dạng *đồ thị* (graph) cũng đang thu hút nhiều nghiên cứu vì tính ứng dụng thực tế. Mô hình *học đối kháng*

(adversarial learning) cũng được xem là một trong những phát kiến quan trọng trong học sâu và được áp dụng rộng rãi với nhiều dạng dữ liệu khác nhau. Trong sách này, hai dạng mô hình học sâu đại diện cho xử lý dữ liệu đồ thị và học đối kháng là GNN (Graph Neural Network) và GAN (Generative Adversarial Networks) cũng được trình bày. Qua đó, độc giả sẽ được giới thiệu những kiến thức mới đang có những đóng góp mới mang tính đột phá cho ngành.

Ngoài ra, trong cuốn sách này cũng thảo luận về việc ứng dụng các dạng mạng học sâu này vào các bài toán chuyển đổi giọng nói thành văn bản và tổng hợp giọng nói từ văn bản. Đây là hai bài toán rất quan trọng và được áp dụng trong nhiều lĩnh vực như truyền thông, chăm sóc sức khỏe, và giải trí. Việc kết hợp xử lý tiếng nói cũng được xem là tương lai gần của các hệ thống chatbot. Hy vọng cuốn sách này sẽ là một tài liệu tham khảo hữu ích cho những ai quan tâm đến lĩnh vực học sâu và ứng dụng của nó trong cuộc sống.

Các chủ đề này sẽ được trải dài trong 7 chương với giả thiết là người đọc đã có kiến thức về các kiến trúc mạng học sâu cơ bản như CNN và RNN. Các thảo luận liên quan đến xử lý ngôn ngữ tự nhiên cũng đòi hỏi người đọc có kiến thức về *mô hình ngôn ngữ* (language model) và *phép nhúng từ* (word embedding). 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), đặc biệt là xử lý ngôn ngữ tự nhiên và xử lý tiếng nói. 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 sinh viên và cộng sự trong nhóm nghiên cứu URA: Nguyễn Quang Đức, Nguyễn Đăng Cương, Nguyễn Huỳnh Minh, Bùi Ngô Hoàng Long, Tô Duy Hưng, Trần Khánh Tùng, Đỗ Lê Thanh Bình, Trần Thị Thắm, Bùi Công Tuấn và Mai Đức Trung đã đóng góp nhiều công sức và ý kiến quý báu trong

quá trình hoàn thiện nội dung sách. Cuối cùng, tác giả xin chân thành gửi lời cảm ơn đến các tác giả của các tài liệu tham khảo về các kiến thức quý báu giúp hoàn thành quyển sách này.

Tóm tắt

Sách *"Các mô hình học sâu nâng cao"* trình bày lý thuyết của các mô hình học sâu nâng cao và ứng dụng vào các bài toán thực tế liên quan. Cấu trúc sách bao gồm hai phần chính. Phần đầu của sách bao gồm năm chương, trình bày chủ yếu các mô hình học sâu hiện đại trong những năm gần đây. 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. Phần thứ hai gồm hai chương, trình bày cách ứng dụng các mô hình học sâu vào giải quyết bài toán chuyển văn bản thành giọng nói và chuyển giọng nói thành văn bản. Nội dung của mỗi chương được mô tả tóm tắt như bên dưới.

Chương 1: **Các mô hình GRU, LSTM và BiLSTM.** Đây là các biến thể mở rộng từ kiến trúc mạng RNN. GRU (Gated Recurrent Unit) và LSTM (Long Short-term Memory) sử dụng các *cổng* (gates) để kiểm soát luồng thông tin qua các thời điểm trong quá trình xử lý dữ liệu chuỗi, trong đó GRU có các bước xử lý và tính toán đơn giản hơn. BiLSTM (Bidirectional LSTM) sử dụng mô hình LSTM trên cả hai hướng của chuỗi đầu vào để tăng cường hiệu quả xử lý và cải thiện kết quả. Các kiến trúc mạng này đã được sử dụng rộng rãi trong các tác vụ như dịch máy, nhận dạng giọng nói và xử lý ngôn ngữ tự nhiên.

Chương 2: **Mô hình Sequence-to-Sequence và cơ chế attention**. Mô hình *Sequence-to-Sequenc*e (Seq2Seq) hoạt động bằng cách chuyển đổi một chuỗi đầu vào sang một chuỗi đầu ra có ý nghĩa khác nhau, thường là chuỗi có độ dài khác nhau. Cơ chế *attention* là một thành phần quan trọng trong Seq2Seq, giúp mô hình tập trung vào những phần quan trọng của chuỗi đầu vào khi thực hiện dự đoán chuỗi đầu ra. Mô hình Seq2Seq thường được xây dựng trên kiến trúc mạng RNN hoặc kiến trúc *Transformer*. Chương này sẽ tập trung vào mô hình Seq2Seq dựa trên RNN.

Chương 3: **Transformer và các mô hình ngôn ngữ lớn**. Transformer là một kiến trúc mạng học sâu được giới thiệu vào năm 2017, nổi tiếng nhất với việc sử dụng trong mô hình dịch máy *Google Neural Machine Translation* (GNMT). Đặc điểm quan trọng của Transformer là cơ chế *self-attention*, giúp tập trung vào các phần tử quan trọng trong chuỗi đầu vào và đầu ra. Đặc biệt, trong chương này cũng giới thiệu hai *mô hình ngôn ngữ lớn* (Large Language Model) nổi tiếng được phát triển từ Transformer là BERT và GPT. *Bidirectional Encoder Representations from Transformers* (BERT) sử dụng encoder của Transformer để biểu diễn thông tin được tổng hợp từ cả hai hướng của văn bản đầu vào. BERT đã đạt được kết quả ấn tượng trong nhiều tác vụ xử lý ngôn ngữ tự nhiên, bao gồm phân loại văn bản, trích xuất thông tin và trả lời câu hỏi. Mặt khác, GPT (Generative Pre-trained Transformer) sử dụng decoder của Transformer để phát triển một mô hình sinh văn bản từ đầu vào là một *câu dẫn nhập* (prompt). Đây cũng chính là tiền thân của chương trình ChatGPT nổi tiếng hiện nay.

Chương 4: **Các mô hình Graph Neural Networks**. *Graph Neural Networks* (GNN) là một lớp mô hình học sâu mới, được sử dụng để xử lý dữ liệu trên đồ thị. GNN sử dụng một mô hình truyền thông tin trên đồ thị, nơi thông tin được truyền qua các nút và cạnh của đồ thị để tính toán các đặc trưng đại diện cho các nút. GNN đã có những tiến bộ đáng kể trong thời gian gần đây, và đang được xem là một trong những lĩnh vực nghiên cứu quan trọng và tiềm năng nhất của học sâu.

Chương 5: **Các mô hình Generative Adversarial Networks**. *Generative Adversarial Networks* (GANs) là một lớp mô hình học sâu dùng để sinh ra dữ liệu mới từ một phân phối cụ thể sử dụng kỹ thuật *huấn luyện đối kháng* (adversarial training) từ một *mạng sinh* (generator) và một *mạng phân loại* (discriminator). GANs đã đạt được những thành công đáng kể trong việc tạo ra hình ảnh, âm nhạc và văn bản mới, và đang được xem là một trong những kỹ thuật học sâu quan trọng và tiềm năng nhất.

Chương 6: **Ứng dụng của học sâu trong bài toán xử lý tiếng nói**. Chương này thảo luận vấn đề chuyển đổi tiếng nói thành văn bản (speech-to-text). Chương sẽ mở đầu bằng các định nghĩa cơ bản của bài toán speech-to-text và tiếp tục bằng các mô hình học sâu thông dụng cho vấn đề này, đặc biệt là các mô hình để xử lý tiếng Việt. Chúng ta sẽ thảo luận về ứng dụng của các mô hình dựa trên RNN, Seq2Seq và Transformer đã thảo luận ở các chương trước vào vấn đề này.

Chương 7: **Ứng dụng của học sâu trong bài toán tổng hợp tiếng nói**. Chương này thảo luận vấn đề tổng hợp tiếng nói từ văn bản (text-to-speech). Tương tự chương trước, chương này sẽ mở đầu bằng các định nghĩa cơ bản của bài toán text-to-speech, các mô hình học sâu thông dụng hướng đến xử lý tiếng Việt cho vấn đề này bao gồm các mô hình dựa trên CNN, Transformer, các cơ chế tạo sinh để huấn luyện dựa trên GAN và đặc biệt các hướng tiếp cận mới dựa trên mô hình *Diffusion* cũng được giới thiệu.

Mục lục

I Các mô hình học sâu nâng cao **1**

Chương 1: Các mô hình GRU, LSTM và BiLSTM **3**

1.1 Giới thiệu 3

1.2 Mô hình GRU 4

1.3 Mô hình LSTM 11

1.4 Mô hình BiLSTM 14

1.5 Kết chương 20

1.6 Bài tập 21

Chương 2: Mô hình Sequence-to-Sequence và cơ chế Attention **23**

2.1 Giới thiệu 23

2.2 Mô hình Sequence-to-sequence cơ bản 24

2.3 Kiến trúc Encoder-Decoder 24

2.4 Mô hình ngôn ngữ có điều kiện 26

2.4.1 Mô hình ngôn ngữ 26

2.4.2 Mô hình ngôn ngữ có điều kiện 27

2.5 Mô hình Seq2Seq với RNN 27

2.6 Các chiến lược huấn luyện mô hình Seq2Seq 29

2.6.1 Chiến lược huấn luyện teacher forcing 29

2.6.2 Chiến lược huấn luyện non-teacher forcing 30

2.7 Cơ chế attention 31

2.8 Kết chương 39

2.9 Bài tập 40

Chương 3: Transformer và các mô hình ngôn ngữ lớn **42**
3.1 Giới thiệu 42
3.2 Kiến trúc mô hình Transformer 43
3.2.1 Các khối Encoder và Decoder 45
3.2.2 Kỹ thuật scaled dot-product attention 45
3.2.3 Khối Feed Forward 54
3.2.4 Khối Positional Encoding 56
3.2.5 Nhận xét 57
3.3 Tiền huấn luyện và các mô hình ngôn ngữ lớn dựa trên Transformer 58
3.4 Mô hình BERT 60
3.4.1 Giới thiệu 60
3.4.2 Các chiến lược huấn luyện của BERT 60
3.4.3 Một số phiên bản khác của BERT 65
3.5 Mô hình GPT 65
3.5.1 Giới thiệu 65
3.5.2 Cơ chế hoạt động của mô hình GPT 67
3.5.3 Các phiên bản của mô hình GPT 69
3.5.4 Chương trình ChatGPT 70
3.6 Kết chương 74
3.7 Bài tập 76
Chương 4: Các mô hình Graph Neural Networks **80**
4.1 Giới thiệu 80
4.2 Mô hình GNN truyền thống 81
4.2.1 Mô hình hóa đồ thị đầu vào 81
4.2.2 Trích xuất đặc trưng đỉnh 83
4.2.3 Hàm mục tiêu và tối ưu 92
4.3 Các biến thể của GNNs 102
4.3.1 Mô hình Graph Convolution Networks 102
4.3.2 Mô hình GraphSAGE 106
4.3.3 Mô hình Graph Attention Networks 111
4.3.4 Mô hình Gated Graph Neural Networks 117
4.3.5 Các mô hình đồ thị khác 117

4.4 Các dạng bài toán áp dụng GNNs điển hình 118
4.4.1 Các mô hình dự đoán toàn đồ thị 118
4.4.2 Các mô hình dự đoán trên đỉnh 121
4.4.3 Các mô hình dự đoán trên cạnh 123
4.5 Kết chương 125
4.6 Bài tập 125
Chương 5: Các mô hình Generative Adversarial Networks 129
5.1 Giới thiệu 129
5.2 Kiến trúc mạng GAN 130
5.3 Hạn chế của mạng GAN 135
5.4 Các cột mốc quan trọng 136
5.4.1 Mô hình GAN gốc 136
5.4.2 Mô hình Conditional GANs 138
5.4.3 Mô hình DCGANs 139
5.5 Kết chương 144
5.6 Bài tập 145

II Ứng dụng các mô hình học sâu vào một số bài toán thực tế 147
Chương 6: Ứng dụng của học sâu trong bài toán xử lý tiếng nói 149
6.1 Giới thiệu 149
6.1.1 Lịch sử phát triển 149
6.1.2 Tổng quan về nhận dạng giọng nói 152
6.2 Kiến trúc cơ bản của hệ thống nhận dạng giọng nói . . . 153
6.2.1 Bộ trích chọn đặc trưng 154
6.2.2 Mô hình âm học 158
6.2.3 Mô hình ngôn ngữ 159
6.2.4 Mô hình phát âm 159
6.2.5 Bộ giải mã 160
6.3 Một số phương pháp xây dựng mô hình nhận dạng . . . 160

6.3.1 Mô hình Markov ẩn 160
6.3.2 Kết hợp mô hình Markov ẩn và mạng thần kinh nhân tạo 163
6.3.3 Mô hình end-to-end 166
6.4 Các phương pháp đánh giá 178
6.4.1 Tốc độ xử lý 178
6.4.2 Độ chính xác 179
Chương 7: Ứng dụng của học sâu trong bài toán tổng hợp tiếng nói 181
7.1 Giới thiệu 181
7.2 Các thành phần chính trong hệ thống Text-To-Speech . . 184
7.2.1 Text Analysis 184
7.2.2 Acoustic Model 185
7.2.3 Vocoder 189
7.3 Các hệ thống Text-To-Speech cho Tiếng Việt 191

Danh sách hình vẽ

1.1 Một đơn vị của mạng RNN 5
1.2 Một đơn vị của mạng GRU 6
1.3 Kiến trúc một đơn vị LSTM 12
1.4 Kiến trúc mô hình BiLSTM 15

2.1 Kiến trúc Encoder-Decoder 25
2.2 Minh họa về mô hình Seq2Seq dựa trên RNN 28
2.3 Minh họa về chiến lược teacher forcing 29
2.4 Minh họa về chiến lược non-teacher forcing 30
2.5 Cơ chế Attention 33
2.6 Quá trình tính toán của attention 34
2.7 Các hàm để tính attention scores 35
2.8 Ví dụ minh họa việc sử dụng Seq2Seq cho bài toán dịch máy. 35

3.1 Kiến trúc mô hình Transformer (Vaswani et al., 2017) . . 44
3.2 Scaled Dot-Product Attention (Vaswani et al., 2017) . . 46
3.3 Scaled Dot-Product Attention dưới góc nhìn là một mô hình tính toán 47
3.4 Self-attention 51
3.5 Multi-head Attention (Vaswani et al., 2017) 53
3.6 Feed Forward Network 54
3.7 Tổng quan quá trình dự đoán và huấn luyện của mô hình BERT 61

3.8 Nhờ vào mục tiêu MLM và kiến trúc encoder của Transformer, BERT có thể được dùng như một khối embedding để trích xuất thông tin ngữ cảnh từ hai phía 62
3.9 Minh họa các bước sinh văn bản của mô hình GPT . . . 67
3.10 Tổng quan kiến trúc Encoder 68
3.11 Trên GPT, dữ liệu ngõ nhập được xử lý trực tiếp trên khối Decoder 69
3.12 Quá trình tổng quát của việc tiên đoán từ tiếp theo trên GPT 70
3.13 So sánh giữa self-attention và masked self-attention . . . 71
3.14 Quá trình sinh ra từ tiên đoán của GPT 72
3.15 Minh họa few-shot learning của GPT-3 73
3.16 Minh họa one-shot learning của GPT-3 73
3.17 Minh họa zero-shot learning của GPT-3 74
3.18 ChatGPT tạo lời giải lập trình 74
3.19 ChatGPT giả lập kernel của Linux 75
3.20 Các câu trả lời dựa trên các mẫu xác suất khác nhau của ChatGPT đôi khi không nhất quán 76
3.21 ChatGPT trả lời sự kiện không chính xác sau năm 2021 77

4.1 Kiến trúc tổng quan của một mô hình GNN 84
4.2 Khái quát về ý tưởng mã hóa đỉnh 85
4.3 Ví dụ minh họa đồ thị tính toán của một đỉnh 89
4.4 Ví dụ về việc tính toán đặc trưng cho các đồ thị tính toán ở một tầng GNN 92
4.5 Ví dụ trực quan về Bước đi ngẫu nhiên và sự giống nhau giữa hai đỉnh bất kỳ 93
4.6 Đồ thị ví dụ mẫu 95
4.7 Đồ thị mẫu dành cho các bài tập 4.5.5 và 4.5.9 126

5.1 Kiến trúc mạng Vanilla GAN - mô hình mạng GAN đầu tiên 131

5.2 Minh họa cho hiện tượng Mode Collapse khi huấn luyện mạng GAN 136
5.3 Kiến trúc mạng cGAN 139
5.4 Ví dụ cho cách tính tích chập chuyển vị 141
5.5 Tích chập chuyển vị kiểu 1 bước 1 142
5.6 Tích chập chuyển vị kiểu 1 bước 2 142
5.7 Tích chập chuyển vị kiểu 1 bước 3 142
5.8 Tích chập chuyển vị kiểu 1 bước 4 142
5.9 Tích chập chuyển vị kiểu 1 bước 5 143
5.10 Mở rộng ma trận đầu vào trong tích chập chuyển vị kiểu 2 143
5.11 Kết quả đầu ra của phép tích chập chuyển vị kiểu 2 . . . 143

6.1 Radio Rex 150
6.2 Mô hình Markov ẩn 4 trạng thái 151
6.3 Kiến trúc cơ bản của hệ thống nhận dạng giọng nói . . . 153
6.4 Framing và windowing 155
6.5 Mel-scale power spectrum 157
6.6 Lược đồ MFCC 158
6.7 Mô hình HMM 4 trạng thái 161
6.8 Kiến trúc chung của mô hình DNN-HMM 164
6.9 Ví dụ về alignment 168
6.10 Các bước thu gọn alignment 169
6.11 Nhiều alignment khác nhau có thể cho ra cùng kết quả sau thu gọn 169
6.12 Giải thuật beam search 173
6.13 Kiến trúc tổng quát cho mô hình sequence-to-sequence . 174
6.14 Kiến trúc mô hình RNN-Transducer 178

7.1 3 thành phần chính trong hệ thống TTS (Tan et al., 2021) 184
7.2 Tổng quát các hướng tiếp cận khác nhau của mạng sinh[1] 189

Danh sách bảng

1.1 Giá trị lý tưởng của Γ_u và $c^{<t>}$ 7

3.1 Tập từ vựng 63
3.2 Xác suất xuất hiện của các từ trong tập từ vựng 64
3.3 Các phiên bản của mô hình GPT 69
3.4 Vocabulary 78

Phần I

Các mô hình học sâu nâng cao

Chương 1

Các mô hình GRU, LSTM và BiLSTM

1.1 Giới thiệu

Trên lý thuyết, các mô hình *mạng nơ-ron hồi quy* (Recurrent Neural Network hay RNN) mang lại hiệu quả lớn trên các dữ liệu có tính tuần tự như văn bản, giọng nói. Tuy nhiên, RNN lại gặp phải các vấn đề trong quá trình huấn luyện như triệt tiêu gradient hay bùng nổ gradient, dẫn đến việc ghi nhớ các phụ thuộc dài hạn bị hạn chế (đồng nghĩa với việc mô hình này chỉ ghi nhớ được một lượng hạn chế thông tin trong quá khứ). Đặc biệt, trong khi việc sử dụng hàm kích hoạt sigmoid làm cho hiện tượng triệt tiêu gradient rất dễ xảy ra, hàm kích hoạt ReLU mặc dù có thể giải quyết được hiện tượng này tuy nhiên lại dẫn tới hiện tượng bùng nổ gradient. Chương này sẽ trình bày về các mô hình được phát triển để vượt qua những khó khăn đó của RNN.

Mô hình *Long Short-term Memory* (LSTM) (Hochreiter and Schmidhuber, 1997) đề xuất việc sử dụng các tầng ẩn như một bộ nhớ bằng cách sử dụng các cổng giúp mô hình có thể quyết định ghi nhớ hay quên lãng một thông tin. Thực nghiệm cho thấy LSTM mang lại hiệu quả vượt trội

so với kiến trúc RNN truyền thống, mô hình có thể ghi nhớ được những thông tin trong quá khứ dài hơn nhiều so với RNN. Vì vậy, LSTM đã được sử dụng như một mô hình tiêu chuẩn cho các loại dữ liệu có tính tuần tự.

Sau đó, một mô hình đơn giản hơn được thiết kế dựa trên tinh thần của LSTM mang tên *Gated Recurrent Unit* (GRU) được công bố cũng mang lại hiệu quả tương đương (Cho et al., 2014). Ở một góc nhìn khác, BiLSTM (Schuster and Paliwal, 1997) là một phiên bản cải tiến của LSTM, mô hình này đề xuất sử dụng hai lớp LSTM theo chiều từ trái sang phải và ngược lại, nhằm mục đích trích xuất được nhiều đặc trưng hơn. Theo cách tiếp cận tương tự, mô hình BiGRU cũng được giới thiệu.

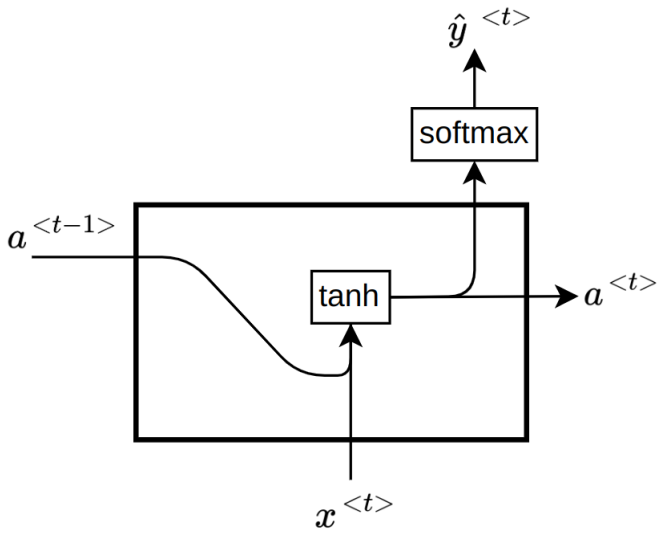
Về mặt thời gian, mặc dù mô hình GRU ra đời sau so với LSTM, tuy nhiên cách tiếp cận lại đơn giản hơn và có tính nối tiếp rõ ràng hơn từ mô hình RNN. Vì vậy chương này sẽ bắt đầu bằng việc trình bày mô hình GRU, với giả định rằng người đọc đã có kiến thức về mô hình RNN. Tiếp sau đó, các mô hình LSTM và BiLSTM sẽ được trình bày.

1.2 Mô hình GRU

Chúng ta bắt đầu bằng việc xét một câu phức thường được giới thiệu cho các mô hình tuần tự như sau:

"The cat, which already ate so much, was full."

Trước tiên, chúng ta cùng nhìn lại một mô hình RNN xử lý câu nhập này. Để huấn luyện mô hình ngôn ngữ, nhiệm vụ của mạng RNN là dự đoán từ sẽ xuất hiện tiếp theo sau khi đã có thông tin về các từ trước đó. Mô hình hoạt động của một *đơn vị* (unit) RNN có thể mô tả như Hình 1.1.



Hình 1.1: Một đơn vị của mạng RNN

Như vậy, ở bước xử lý thứ t (tức là xử lý véc-tơ nhập $x^{<t>}$), mạng RNN sẽ tạo ra một véc-tơ *bộ nhớ* (memory) $a^{<t>}$, thường được gọi là *trạng thái ẩn* (hidden state). Véc-tơ $a^{<t>}$ được tính toán dựa trên giá trị nhập hiện tại $x^{<t>}$ và giá trị bộ nhớ $a^{<t-1>}$ ở bước xử lý thứ $t-1$ trước đó như Công thức 1.1

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a), \tag{1.1}$$

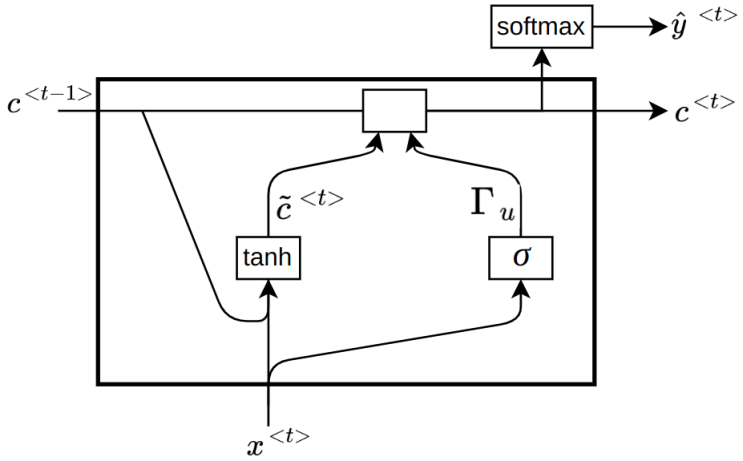
trong đó g_1 là một hàm kích hoạt (thường là *ReLU* hoặc tanh), các ma trận W_{aa} và W_{ax} là các ma trận chứa các trọng số học sẽ được huấn luyện trong quá trình học, b_a là giá trị bias.
Giá trị bộ nhớ $a^{<t>}$ cũng có thể được sử dụng để tính giá trị xuất tương ứng ở bước thứ t này như Công thức 1.2

$$\hat{y}^{<t>} = g_2(W_{ya}a^{<t>} + b_y), \tag{1.2}$$

trong đó g_2 là một hàm kích hoạt khác. Đối với bài toán phân loại nhãn thì g_2 thường là *sigmoid* hoặc *softmax*. W_{ay} và b_y là các ma trận trọng số và giá trị bias tương ứng. Khi huấn luyện, giá trị $\hat{y}^{<t>}$ được mong đợi

càng giống với giá trị được gán nhãn $y^{<t>}$ càng tốt.
Như vậy, véc-tơ $a^{<t>}$ được sử dụng để ghi nhớ lại thông tin đã xử lý ở những bước trước đó. Theo lý thuyết, nếu thông tin được ghi nhớ càng đầy đủ, giá trị $\hat{y}^{<t>}$ sẽ càng hội tụ về giá trị $y^{<t>}$. Ví dụ, khi xử lý chữ *cat* trong câu nhập trên, nếu $a^{<\cdot>}$ có thể ghi nhớ được thông tin rằng đây là danh từ số ít, thì sau khi xử lý đến chữ *much* và dấu phẩy sau đó, mạng RNN sẽ có xu hướng dự đoán từ tiếp theo là *was* (cho danh từ số ít) thay vì *were* (cho danh từ số nhiều).

Tuy nhiên, do trong câu nhập có khá nhiều từ sẽ được xử lý sau chữ *cat*, nên trước khi tiến hành việc dự đoán chữ *was* như vừa thảo luận, thông tin được ghi nhớ trong $a^{<\cdot>}$ sẽ được cập nhật lại sau mỗi bước. Điều này dẫn đến việc thông tin lưu trữ về danh từ số ít của *cat* sẽ bị suy biến và không còn giữ được nguyên trạng sau đó, ảnh hưởng đến kết quả dự đoán của mô hình RNN.
Mạng GRU giải quyết vấn đề trên bằng cách giới thiệu cơ chế *giữ cổng* (gating mechanism). Ý tưởng của cơ chế này là quyết định xem thông tin lưu trữ trong véc-tơ bộ nhớ sẽ được cập nhật hay giữ nguyên sau mỗi bước xử lý. Để hiện thực cơ chế này, một đơn vị của mạng GRU sẽ được mở rộng so với một đơn vị của mạng RNN như Hình 1.2.



Hình 1.2: Một đơn vị của mạng GRU

Trên mạng GRU, khái niệm véc-tơ memory $a^{<.>}$ sẽ được thay bằng memory cell $c^{<.>}$. Để đơn giản hóa vấn đề, chúng ta giả định rằng giá trị $c^{<.>}$ sẽ có giá trị 1 hoặc 0, tương ứng với việc danh từ đang xem xét là số ít hoặc số nhiều. Trong trường hợp lý tưởng, khi xử lý đến chữ *cat*, giá trị $c^{<.>}$ sẽ là 1. Giá trị này sẽ không được cập nhật cho đến khi xử lý xong động từ thể hiện hành động đi kèm với danh từ này. Việc giữ giá trị $c^{<.>}$ được quyết định bằng một *cổng* (gate) Γ_u theo các Công thức 1.3, 1.4 và 1.5

$$\Gamma_u = \sigma(W_{cu}c^{<t-1>} + W_{xu}x^{<t>} + b_u), \tag{1.3}$$

$$\widetilde{c}^{<t>} = \tanh(W_{cc}c^{<t-1>} + W_{xc}x^{<t>} + b_c), \tag{1.4}$$

$$c^{<t>} = \Gamma_u \odot \widetilde{c}^{<t>} + (1 - \Gamma_u) \odot c^{<t-1>}, \tag{1.5}$$

ý nghĩa của các ma trận trọng số và giá trị bias tương tự như đã thảo luận trước đó. Theo Công thức 1.3, giá trị của Γ_u sẽ được quyết định bởi một hàm *sigmoid*, tức là một giá trị nằm giữa 0 và 1. Công thức 1.4 chính là công thức để cập nhật memory cell, tương tự RNN. Từ Công thức 1.5, chúng ta sẽ thấy khi Γ_u bằng 0, giá trị $c^{<t>}$ sẽ bằng $c^{<t-1>}$, tức là không được cập nhật. Ngược lại, nếu Γ_u bằng 1, giá trị $c^{<t>}$ sẽ bằng $\widetilde{c}^{<t>}$, tức là được cập nhật mới hoàn toàn.
Trong trường hợp lý tưởng, giá trị của Γ_u và $c^{<t>}$ theo thời gian được biểu diễn theo Bảng 1.1.

Bảng 1.1: Giá trị lý tưởng của Γ_u và $c^{<t>}$

	The	cat	which	already	ate	so	much	,	was	full
Γ_u		1	0	0	0	0	0	0	0	
$c^{<t>}$		1	1	1	1	1	1	1	1	

Trong thực tế, $c^{<.>}$ sẽ mã hóa nhiều thông tin khác nhau của dữ liệu đã được xử lý cho đến bước hiện hành, và giá trị của Γ_u sẽ là một giá trị nằm giữa 0 và 1. Ở mức ý niệm, giá trị Γ_u sẽ quyết định có bao nhiêu

phần trăm thông tin trước đó của memory cell $c^{<t-1>}$ ở trạng thái trước sẽ được giữ lại và bao nhiêu thông tin sẽ được cập nhật bởi thông tin $\widetilde{c}^{<t>}$. Các giá trị của Γ_u sẽ không đạt 0 tuyệt đối. Tuy nhiên, do đặc trưng của hàm *sigmoid* nên sẽ rất dễ đạt được giá trị gần 0 (khi giá trị hàm tuyến tính có giá trị rất âm), làm cho $c^{<t>}$ theo Công thức 1.5 cũng sẽ đạt giá trị rất gần với $c^{<t-1>}$. Bằng cách này, GRU có thể giải quyết được vấn đề triệt tiêu gradient, giá trị $c^{<.>}$ sẽ được truyền đi xa hơn. Ngoài ra, GRU còn sử dụng một cổng Γ_r để quyết định mức độ tương quan giữa $c^{<t-1>}$ và $c^{<t>}$. Khi đó, biểu thức (1.4) được viết lại thành

$$\widetilde{c}^{<t>} = \tanh(W_{cc}(\Gamma_r \odot c^{<t-1>}) + W_{xc}x^{<t>} + b_c),$$

trong đó,

$$\Gamma_r = \sigma(W_{cr}c^{<t-1>} + W_{xr}x^{<t>} + b_r).$$

Ví dụ 1.2.1. Giả sử ta có một kho ngữ liệu (*vocab*) gồm các từ *{"H", "E", "L", "O"}* và muốn huấn luyện mô hình RNN và GRU để có thể phân tích chuỗi nhập vào *"HE"* và dự đoán được từ tiếp theo là *"L"* (để thu được từ *"HELLO"*). Một số dữ kiện đầu vào:

- Các từ trong kho ngữ liệu đã được mã hóa bằng kỹ thuật *one-hot encoding* như sau:

$$e_H = \begin{bmatrix} 1 & ; & 0 & ; & 0 & ; & 0 \end{bmatrix}, e_E = \begin{bmatrix} 0 & ; & 1 & ; & 0 & ; & 0 \end{bmatrix},$$

$$e_L = \begin{bmatrix} 0 & ; & 0 & ; & 1 & ; & 0 \end{bmatrix}, e_O = \begin{bmatrix} 0 & ; & 0 & ; & 0 & ; & 1 \end{bmatrix},$$

$$W_{xu} = \begin{bmatrix} 0.4 & 0.9 & 0.9 & 0.3 \\ 0.3 & 0.6 & 0.2 & 0.9 \\ 0.2 & 0.7 & 0.4 & 0.8 \end{bmatrix}, W_{cu} = \begin{bmatrix} 1.0 & 0.9 & 0.2 \\ 0.3 & 0.7 & 0.1 \\ 0.1 & 0.6 & 0.2 \end{bmatrix},$$

$$W_{xr} = \begin{bmatrix} 0.1 & 0.2 & 0.8 & 0.2 \\ 0.5 & 0.7 & 0.6 & 0.1 \\ 0.3 & 0.6 & 0.1 & 0.1 \end{bmatrix}, W_{cr} = \begin{bmatrix} 0.5 & 0.3 & 0.1 \\ 0.2 & 0.9 & 0.2 \\ 0.2 & 0.9 & 0.7 \end{bmatrix},$$

$$W_{xa} = W_{xc} = \begin{bmatrix} 0.8 & 0.5 & 0.1 & 0.9 \\ 0.1 & 0.7 & 0.2 & 0.8 \\ 0.7 & 0.5 & 0.6 & 0.3 \end{bmatrix}, W_{aa} = W_{cc} = \begin{bmatrix} 0.2 & 0.5 & 0.3 \\ 0.1 & 0.6 & 0.5 \\ 0.3 & 0.7 & 0.9 \end{bmatrix},$$

$$W_{ay} = W_{cy} = \begin{bmatrix} 0.1 & 0.6 & 0.3 \\ 0.3 & 0.7 & 0.5 \\ 0.2 & 0.4 & 0.1 \\ 0.8 & 0.9 & 0.2 \end{bmatrix},$$

- Chúng ta bỏ qua các giá trị bias và có giá trị khởi tạo $a^{<0>} = c^{<0>} = \begin{bmatrix} 0 & ; & 0 & ; & 0 \end{bmatrix}$.

Trả lời.

Kết quả xử lý của mô hình RNN

Xử lý từ đầu tiên $(t = 1, x_t = e_H)$:

$$a^{<1>} = \tanh(W_{aa}a^{<0>} + W_{xa}x^{<1>}) = \begin{bmatrix} 0.6640367 & ; & 0.0996679 & ; & 0.6043677 \end{bmatrix}$$
$$\hat{y}^{<1>} = softmax(W_{ay}a^{<1>}) = \begin{bmatrix} 0.21 & ; & 0.27 & ; & 0.19 & ; & \mathbf{0.33} \end{bmatrix} \qquad (O)$$

Xử lý từ thứ hai $(t = 2, x_t = e_E)$:

$$a^{<2>} = \tanh(W_{aa}a^{<1>} + W_{xa}x^{<2>}) = \begin{bmatrix} 0.6982880 & ; & 0.8104669 & ; & 0.8650097 \end{bmatrix}$$
$$\hat{y}^{<2>} = softmax(W_{ay}a^{<2>}) = \begin{bmatrix} 0.19 & ; & 0.29 & ; & 0.15 & ; & \mathbf{0.37} \end{bmatrix} \qquad (O)$$

Kết quả mô hình GRU

Xử lý từ đầu tiên $(t = 1, x_t = e_H)$:

$$\Gamma_u = \sigma(W_{cu}c^{<0>} + W_{xu}x^{<1>}) = \begin{bmatrix}0.5986877 & ; & 0.5744425 & ; & 0.549834\end{bmatrix}$$
$$\Gamma_r = \sigma(W_{cr}c^{<0>} + W_{xr}x^{<1>}) = \begin{bmatrix}0.5249792 & ; & 0.6224593 & ; & 0.5744425\end{bmatrix}$$
$$\widetilde{c}^{<1>} = \tanh(W_{cc}(\Gamma_r \odot c^{<0>}) + W_{xc}x^{<1>}) = \begin{bmatrix}0.66403677 & ; & 0.09966799 & ; & 0.60436778\end{bmatrix}$$
$$c^{<1>} = \Gamma_u \odot \widetilde{c}^{<1>} + (1 - \Gamma_u) \odot c^{<0>} = \begin{bmatrix}0.3975506 & ; & 0.0572535 & ; & 0.3323020\end{bmatrix}$$
$$\hat{y}^{<1>} = softmax(W_{cy}c^{<1>}) = \begin{bmatrix}0.23 & ; & 0.26 & ; & 0.22 & ; & \mathbf{0.29}\end{bmatrix} \qquad (O)$$

Xử lý từ thứ hai $(t = 2, x_t = e_E)$:

$$\Gamma_u = \sigma(W_{cu}c^{<1>} + W_{xu}x^{<2>}) = \begin{bmatrix}0.8046381 & ; & 0.6883835 & ; & 0.6985847\end{bmatrix}$$
$$\Gamma_r = \sigma(W_{cr}c^{<1>} + W_{xr}x^{<2>}) = \begin{bmatrix}0.6104446 & ; & 0.7104352 & ; & 0.7238519\end{bmatrix}$$
$$\widetilde{c}^{<2>} = \tanh(W_{cc}(\Gamma_r \odot c^{<1>}) + W_{xc}x^{<2>}) = \begin{bmatrix}0.56560399 & ; & 0.70083614 & ; & 0.67384932\end{bmatrix}$$
$$c^{<2>} = \Gamma_u \odot \widetilde{c}^{<2>} + (1 - \Gamma_u) \odot c^{<1>} = \begin{bmatrix}0.593467578 & ; & 0.616006402 & ; & 0.671436139\end{bmatrix}$$
$$\hat{y}^{<2>} = softmax(W_{cy}c^{<2>}) = \begin{bmatrix}0.20 & ; & 0.28 & ; & 0.17 & ; & \mathbf{0.35}\end{bmatrix} \qquad (O)$$

Với các ma trận trọng số ở trên, RNN và GRU đều cho ra kết quả dự đoán ký tự tiếp theo là *"O"* thay vì *"L"* như mong đợi. Tuy nhiên, kết quả này có thể được cải thiện bằng cơ chế lan truyền ngược trong quá trình huấn luyện mô hình. Đối với những trường hợp ngõ nhập dài và nhiều thông tin cần "ghi nhớ", cơ chế gating của GRU sẽ giúp ghi nhớ các thông tin phù hợp thông qua các ma trận trọng số được huấn luyện qua quá trình học.

1.3 Mô hình LSTM

Như đã đề cập trước đó, GRU là một mô hình được giản lược hóa từ mô hình *Long Short-term Memory* (LSTM). So với GRU, mô hình gốc LSTM sử dụng nhiều gates để lọc và cập nhật thông tin hơn, cụ thể như sau:

- LSTM sử dụng một biến $c^{<t>}$ riêng biệt ($c^{<t>} \neq a^{<t>}$) chuyên dùng để quản lý thông tin "ghi nhớ" (short-term memory)thay vì dùng chung với trạng thái ẩn (hidden state) như mạng RNN truyền thống ($c^{<t>} = a^{<t>}$) mà GRU cũng sử.
- LSTM sử dụng một cổng riêng Γ_f cho việc quyết định quên lãng thông tin cũ. GRU nhập các cổng quên lãng và cập nhật thông tin thành một cổng Γ_u cho cả việc cập nhật và quên lãng như đã trình bày trên.

Trong một đơn vị LSTM, việc cập nhật các thông tin memory và hidden state sẽ được tiến hành theo các các Công thức 1.6), (1.7), (1.8), (1.9), (1.10) và (1.11), cụ thể như sau:

$$\widetilde{c}^{<t>} = \tanh(W_{ac}a^{<t-1>} + W_{xc}x^{<t>} + b_c) \tag{1.6}$$

$$\Gamma_u = \sigma(W_{au}a^{<t-1>} + W_{xu}x^{<t>} + b_u) \tag{1.7}$$

$$\Gamma_f = \sigma(W_{af}a^{<t-1>} + W_{xf}x^{<t>} + b_f) \tag{1.8}$$

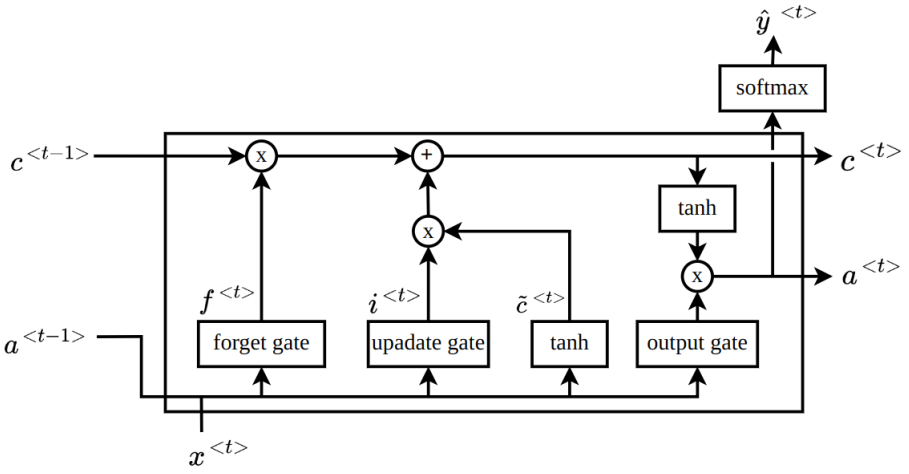
$$\Gamma_o = \sigma(W_{ao}a^{<t-1>} + W_{xo}x^{<t>} + b_o) \tag{1.9}$$

$$c^{<t>} = \Gamma_u \odot \widetilde{c}^{<t>} + \Gamma_f \odot c^{<t-1>} \tag{1.10}$$

$$a^{<t>} = \Gamma_o \odot \tanh(c^{<t>}) \tag{1.11}$$

$$\hat{y}^{<t>} = softmax(W_{ay}a^{<t>}) \tag{1.12}$$

Trong đó ý nghĩa của các cổng, các ma trận cập nhật trọng số và các biến bias tương tự như các khái niệm tương tự trên RNN và GRU. Một cách trực quan, một đơn vị LSTM được biểu diễn như Hình 1.3.



Hình 1.3: Kiến trúc một đơn vị LSTM

Cơ chế hoạt động của LSTM được diễn giải cụ thể như sau:

1. Mô hình sử dụng biến đầu vào $x^{<t>}$, biến nhớ $c^{<t-1>}$ và biến trạng thái $a^{<t-1>}$ ở bước trước để tính toán các giá trị $f^{<t>}$, $i^{<t>}$, $o^{<t>}$ và $\widetilde{c}^{<t>}$ qua các cổng *"forget gate"*, *"update gate"*, *"output gate"* và *"tanh"* bằng (1.8), (1.7), (1.9) và (1.6). Các "gates" này đều được hiện thực bằng các hàm sigmoid, tức là sẽ cho ra một giá trị nằm giữa 0 và 1, dựa vào đó LSTM sẽ quyết định cập nhật/quên lãng thông tin tương ứng,

2. Cổng *forget gate* sẽ quyết định có bao nhiêu phần trăm giá trị của bộ nhớ cũ $c^{<t>}$ sẽ được giữ lại,

3. Cổng *update gate* sẽ quyết định có bao nhiêu phần trăm giá trị của bộ nhớ mới $\widetilde{c}^{<t>}$ sẽ được dùng để cập nhật cùng với phần bộ nhớ cũ đã được giữ lại. Cách tính $\widetilde{c}^{<t>}$ tương tự như đã trình bày ở RNN và GRU. Giá trị $c^{<t>}$ được tính bằng cách quên lãng (do $f^{<t>}$ quyết định) và cập nhật (do $i^{<t>}$ quyết định) giá trị $c^{<t-1>}$ trước đó theo Công thức 1.10,

4. Cuối cùng, cổng *output gate* sẽ quyết định có bao nhêu phần trăm các giá trị của hidden state cũ $a^{<t-1>}$ và input hiện hành $x^{<t>}$ được

dùng để giá trị đầu ra $a^{<t>}$, được tính theo Công thức 1.11,

5. Các giá trị $c^{<t>}$ và $a^{<t>}$ sẽ được truyền đến đơn vị kế tiếp theo tinh thần của mạng RNN truyền thống. $a^{<t>}$ có thể sử dụng để tính giá trị output tương tự như đã trình bày đối với RNN và GRU.

LSTM giải quyết vấn đề triệt tiêu gradient bằng biến nhớ c, việc quên lãng hay cập nhật biến nhớ này trong suốt quá trình xử lý tuần tự từ trái sang phải cho phép thông tin có thể truyền đi rất xa. GRU dựa trên ý tưởng này của LSTM nhưng đơn giản hóa các cổng và bộ nhớ lưu trữ để có thể đẩy nhanh quá trình huấn luyện.

Ví dụ 1.3.1. Hãy tính toán lại vấn đề đặt ra ở Ví dụ 1.2.1, ta cần huấn luyện một mô hình tương tự sử dụng LSTM thay cho GRU với bộ tham số sau:

$$W_{xf} = W_{xr}, W_{au} = W_{cu}, W_{af} = W_{cr},$$
$$W_{ac} = \begin{bmatrix} 0.1 & 0.6 & 0.3 \\ 0.3 & 0.7 & 0.5 \\ 0.2 & 0.4 & 0.1 \end{bmatrix}, W_{xo} = \begin{bmatrix} 0.1 & 0.6 & 0.3 & 0.8 \\ 0.3 & 0.7 & 0.5 & 0.6 \\ 0.2 & 0.4 & 0.1 & 0.2 \end{bmatrix}, W_{ao} = \begin{bmatrix} 0.1 & 0.6 & 0.3 \\ 0.3 & 0.7 & 0.5 \\ 0.2 & 0.4 & 0.1 \end{bmatrix}$$

Trả lời.
Xử lý từ đầu tiên $(t = 1, x_t = e_H)$:

$$\widetilde{c}^{<1>} = \tanh(W_{ac}a^{<0>} + W_{xc}x^{<1>}) = \begin{bmatrix} 0.6640367 & ; & 0.0996679 & ; & 0.6043677 \end{bmatrix}$$
$$\Gamma_u = \sigma(W_{au}a^{<0>} + W_{xu}x^{<1>}) = \begin{bmatrix} 0.5986876 & ; & 0.5744425 & ; & 0.5498339 \end{bmatrix}$$
$$\Gamma_f = \sigma(W_{af}a^{<0>} + W_{xf}x^{<1>}) = \begin{bmatrix} 0.5249791 & ; & 0.6224593 & ; & 0.5744425 \end{bmatrix}$$
$$\Gamma_o = \sigma(W_{ao}a^{<0>} + W_{xo}x^{<1>}) = \begin{bmatrix} 0.5249791 & ; & 0.5744425 & ; & 0.5498339 \end{bmatrix}$$
$$c^{<1>} = \Gamma_u \odot \widetilde{c}^{<1>} + \Gamma_f \odot c^{<0>} = \begin{bmatrix} 0.3975505 & ; & 0.05725347 & ; & 0.3323018 \end{bmatrix}$$
$$a^{<1>} = \Gamma_o \odot \tanh(c^{<1>}) = \begin{bmatrix} 0.1983639 & ; & 0.0355990 & ; & 0.1841590 \end{bmatrix}$$

$$\hat{y}^{<1>} = softmax(W_{ay}a^{<1>}) = \begin{bmatrix} 0.24 & ; & 0.26 & ; & 0.23 & ; & \mathbf{0.27} \end{bmatrix} \qquad (O)$$

Xử lý từ thứ hai $(t = 2, x_t = e_E)$:

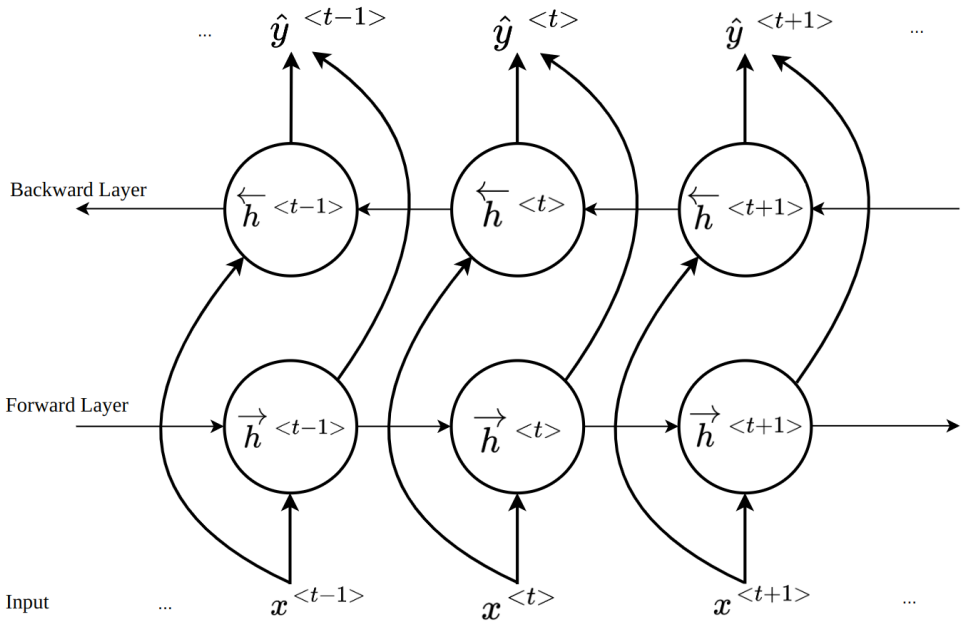
$$\widetilde{c}^{<2>} = \tanh(W_{ac}a^{<1>} + W_{xc}x^{<2>}) = \begin{bmatrix} 0.5345140 & ; & 0.7046655 & ; & 0.5170671 \end{bmatrix}$$
$$\Gamma_u = \sigma(W_{au}a^{<1>} + W_{xu}x^{<2>}) = \begin{bmatrix} 0.7626448 & ; & 0.6688181 & ; & 0.6852548 \end{bmatrix}$$
$$\Gamma_f = \sigma(W_{af}a^{<1>} + W_{xf}x^{<2>}) = \begin{bmatrix} 0.5813402 & ; & 0.6917990 & ; & 0.6901077 \end{bmatrix}$$
$$\Gamma_o = \sigma(W_{ao}a^{<1>} + W_{xo}x^{<2>}) = \begin{bmatrix} 0.6673987 & ; & 0.7060980 & ; & 0.6159346 \end{bmatrix}$$
$$c^{<2>} = \Gamma_u \odot \widetilde{c}^{<2>} + \Gamma_f \odot c^{<1>} = \begin{bmatrix} 0.6387564 & ; & 0.5109010 & ; & 0.5836468 \end{bmatrix}$$
$$a^{<2>} = \Gamma_o \odot \tanh(c^{<2>}) = \begin{bmatrix} 0.3764477 & ; & 0.3323228 & ; & 0.3235572 \end{bmatrix}$$
$$\hat{y}^{<2>} = softmax(W_{ay}a^{<2>}) = \begin{bmatrix} 0.22 & ; & 0.26 & ; & 0.20 & ; & \mathbf{0.31} \end{bmatrix} \qquad (O)$$

1.4 Mô hình BiLSTM

Một đặc điểm dễ thấy là mô hình RNN cùng các biến thể của nó như GRU và LSTM xử lý thông tin theo một chiều. Trong các ví dụ bên trên, các mô hình xử dụng các thông tin đã tích lũy từ trái qua phải cho phần tử input cần xử lý để sinh ra output tương ứng. Trong nhiều trường hợp, nếu mô hình có thể biết được thông tin ở phía sau (tức là bên phải của phần tử cần xử lý trong các ví dụ trên) sẽ giúp ích cho việc trích xuất đặc trưng của phần tử input hiện hành. Điều này đặc biệt quan trọng trong việc xử lý văn bản vì tập hợp các *từ lân cận từ hai phía trái và phải* của một từ sẽ giúp hình thành ngữ cảnh (context) của từ cần xem xét.

Từ ý tưởng đó, mô hình BiLSTM được đề xuất để cải tiến mô hình LSTM mạnh mẽ hơn. Kiến trúc mô hình được mô tả như Hình 1.4, theo đó BiLSTM sử dụng hai tầng LSTM theo hai chiều gọi là *Forward Layer* và *Backward Layer*. Foward Layer là tầng LSTM sẽ xử lý dữ liệu đầu vào từ trái sang phải như LSTM thông thường, Backward Layer là tầng

LSTM xử lý dữ liệu đầu vào từ phải sang trái. Mỗi giá trị đầu vào $x^{<t>}$ được truyền đồng thời qua cả hai tầng này và thu được hai véc-tơ trạng thái tương ứng là $\overrightarrow{a}^{<t>}$ và $\overleftarrow{a}^{<t>}$. Sau đó, hai véc-tơ này được tổng hợp để tính giá trị đầu ra $\hat{y}^{<t>}$.



Hình 1.4: Kiến trúc mô hình BiLSTM

Một cách trực quan, có thể thấy rằng giá trị đầu ra $\hat{y}^{<t>}$ vừa nhận được thông tin từ bên trái ngõ nhập hiện hành theo hướng từ trái qua phải $x^{<t-1>} \rightarrow \overrightarrow{a}^{<t-1>} \rightarrow \overrightarrow{a}^{<t>} \rightarrow \hat{y}^{<t>}$, đồng thời vừa nhận được thông tin từ bên phải ngõ nhập hiện hành theo hướng từ phải qua trái $x^{<t+1>} \rightarrow \overleftarrow{a}^{<t+1>} \rightarrow \overleftarrow{a}^{<t>} \rightarrow \hat{y}^{<t>}$.

Cụ thể, giá trị đầu ra được tính theo (1.13) như bên dưới

$$\hat{y}^{<t>} = softmax(W_{ay}(\overrightarrow{a}^{<t>} \odot \overleftarrow{a}^{<t>})), \tag{1.13}$$

trong đó,

$$\overrightarrow{a}^{<t>} = LSTM(x^{<t>}, \overrightarrow{c}^{<t-1>}, \overrightarrow{a}^{<t-1>}),$$

$$\overleftarrow{a}^{<t>} = LSTM(x^{<t>}, \overleftarrow{c}^{<t+1>}, \overleftarrow{a}^{<t+1>}),$$

với toán tử $\odot$ có thể thay thành các hàm tổng hợp khác tùy theo hiện thực của mô hình.

Tuy nhiên, việc sử dụng thông tin ở theo hướng từ phải sang trái (vốn trái với tự nhiên của con người) để đưa ra kết quả dự đoán cũng đem lại khó khăn cho mô hình BiLSTM trong một số trường hợp. Với những bài toán cần đưa ra kết quả lập tức như sinh ra phụ đề cho một đoạn phim, mô hình BiLSTM phải chờ đến khi người nói kết thúc một câu mới có thể sinh ra được kết quả.

Ví dụ 1.4.1. Xét bài toán ở Ví dụ 1.3.1, ta cần huấn luyện một mô hình tương tự sử dụng BiLSTM thay cho LSTM, giả sử giá trị của bộ trọng số ở hai tầng LSTM là như nhau.

Trả lời.
Tầng LSTM từ trái sang phải:
Xử lý từ thứ nhất $(t = 1, x_t = e_H)$:

$$\widetilde{c}_{lr}^{<1>} = \tanh(W_{ac}a_{lr}^{<0>} + W_{xc}x^{<1>}) = \begin{bmatrix}0.6640367 & ; & 0.0996679 & ; & 0.6043677\end{bmatrix}$$
$$\Gamma_u = \sigma(W_{au}a_{lr}^{<0>} + W_{xu}x^{<1>}) = \begin{bmatrix}0.5986876 & ; & 0.5744425 & ; & 0.5498339\end{bmatrix}$$
$$\Gamma_f = \sigma(W_{af}a_{lr}^{<0>} + W_{xf}x^{<1>}) = \begin{bmatrix}0.5249791 & ; & 0.6224593 & ; & 0.5744425\end{bmatrix}$$
$$\Gamma_o = \sigma(W_{ao}a_{lr}^{<0>} + W_{xo}x^{<1>}) = \begin{bmatrix}0.5249791 & ; & 0.5744425 & ; & 0.5498339\end{bmatrix}$$
$$c_{lr}^{<1>} = \Gamma_u \odot \widetilde{c}_{lr}^{<1>} + \Gamma_f \odot c^{<0>} = \begin{bmatrix}0.3975506 & ; & 0.0572535 & ; & 0.3323019\end{bmatrix}$$
$$a_{lr}^{<1>} = \Gamma_o \odot \tanh(c_{lr}^{<1>}) = \begin{bmatrix}0.1983640 & ; & 0.0328529 & ; & 0.1762699\end{bmatrix}$$

Xử lý từ thứ hai $(t = 2, x_t = e_E)$:

$$\widetilde{c}_{lr}^{<2>} = \tanh(W_{ac}a_{lr}^{<1>} + W_{xc}x^{<2>}) = \begin{bmatrix}0.5316404 & ; & 0.7016998 & ; & 0.5156830\end{bmatrix}$$
$$\Gamma_u = \sigma(W_{au}a_{lr}^{<1>} + W_{xu}x^{<2>}) = \begin{bmatrix}0.7619110 & ; & 0.6682173 & ; & 0.6845587\end{bmatrix}$$

$$\Gamma_f = \sigma(W_{af}a_{lr}^{<1>} + W_{xf}x^{<2>}) = \begin{bmatrix}0.5809476 & ; & 0.6909350 & ; & 0.6883956\end{bmatrix}$$
$$\Gamma_o = \sigma(W_{ao}a_{lr}^{<1>} + W_{xo}x^{<2>}) = \begin{bmatrix}0.6665070 & ; & 0.7048791 & ; & 0.6154881\end{bmatrix}$$
$$c_{lr}^{<2>} = \Gamma_u \odot \widetilde{c}_{lr}^{<2>} + \Gamma_f \odot c^{<1>} = \begin{bmatrix}0.6360188 & ; & 0.5084464 & ; & 0.5817705\end{bmatrix}$$
$$a_{lr}^{<2>} = \Gamma_o \odot \tanh(c_{lr}^{<2>}) = \begin{bmatrix}0.3746987 & ; & 0.3304007 & ; & 0.3224857\end{bmatrix}$$

Xử lý từ thứ ba $(t = 3, x_t = e_L)$:

$$\widetilde{c}_{lr}^{<3>} = \tanh(W_{ac}a_{lr}^{<2>} + W_{xc}x^{<3>}) = \begin{bmatrix}0.4073717 & ; & 0.6074896 & ; & 0.6854638\end{bmatrix}$$
$$\Gamma_u = \sigma(W_{au}a_{lr}^{<2>} + W_{xu}x^{<3>}) = \begin{bmatrix}0.8370658 & ; & 0.6401323 & ; & 0.6682337\end{bmatrix}$$
$$\Gamma_f = \sigma(W_{af}a_{lr}^{<2>} + W_{xf}x^{<3>}) = \begin{bmatrix}0.7537508 & ; & 0.7382316 & ; & 0.6677531\end{bmatrix}$$
$$\Gamma_o = \sigma(W_{ao}a_{lr}^{<2>} + W_{xo}x^{<3>}) = \begin{bmatrix}0.6530461 & ; & 0.7320273 & ; & 0.5840322\end{bmatrix}$$
$$c_{lr}^{<3>} = \Gamma_u \odot \widetilde{c}_{lr}^{<3>} + \Gamma_f \odot c^{<2>} = \begin{bmatrix}0.8203967 & ; & 0.7642250 & ; & 0.8465292\end{bmatrix}$$
$$a_{lr}^{<3>} = \Gamma_o \odot \tanh(c_{lr}^{<3>}) = \begin{bmatrix}0.4409927 & ; & 0.4711026 & ; & 0.4025453\end{bmatrix}$$

Xử lý từ thứ tư $(t = 4, x_t = e_L)$:

$$\widetilde{c}_{lr}^{<4>} = \tanh(W_{ac}a_{lr}^{<3>} + W_{xc}x^{<4>}) = \begin{bmatrix}0.4986625 & ; & 0.6979757 & ; & 0.7244248\end{bmatrix}$$
$$\Gamma_u = \sigma(W_{au}a_{lr}^{<3>} + W_{xu}x^{<4>}) = \begin{bmatrix}0.8635972 & ; & 0.6687028 & ; & 0.6915274\end{bmatrix}$$
$$\Gamma_f = \sigma(W_{af}a_{lr}^{<3>} + W_{xf}x^{<4>}) = \begin{bmatrix}0.7688949 & ; & 0.7672236 & ; & 0.7097093\end{bmatrix}$$
$$\Gamma_o = \sigma(W_{ao}a_{lr}^{<3>} + W_{xo}x^{<4>}) = \begin{bmatrix}0.6786390 & ; & 0.7619395 & ; & 0.6027398\end{bmatrix}$$
$$c_{lr}^{<4>} = \Gamma_u \odot \widetilde{c}_{lr}^{<4>} + \Gamma_f \odot c^{<3>} = \begin{bmatrix}1.0614425 & ; & 1.0530698 & ; & 1.1017493\end{bmatrix}$$
$$a_{lr}^{<4>} = \Gamma_o \odot \tanh(c_{lr}^{<4>}) = \begin{bmatrix}0.5335564 & ; & 0.5965963 & ; & 0.4828708\end{bmatrix}$$

Xử lý từ thứ năm $(t = 5, x_t = e_O)$:

$$\widetilde{c}_{lr}^{<5>} = \tanh(W_{ac}a_{lr}^{<4>} + W_{xc}x^{<5>}) = \begin{bmatrix}0.8969070 & ; & 0.9244964 & ; & 0.6003133\end{bmatrix}$$

$$\Gamma_u = \sigma(W_{au}a_{lr}^{<4>} + W_{xu}x^{<5>}) = \begin{bmatrix}0.8126112 & ; & 0.8214161 & ; & 0.7871594\end{bmatrix}$$
$$\Gamma_f = \sigma(W_{af}a_{lr}^{<4>} + W_{xf}x^{<5>}) = \begin{bmatrix}0.6668659 & ; & 0.6985120 & ; & 0.7468075\end{bmatrix}$$
$$\Gamma_o = \sigma(W_{ao}a_{lr}^{<4>} + W_{xo}x^{<5>}) = \begin{bmatrix}0.7951372 & ; & 0.8052003 & ; & 0.6441991\end{bmatrix}$$
$$c_{lr}^{<5>} = \Gamma_u \odot \widetilde{c}_{lr}^{<5>} + \Gamma_f \odot c^{<4>} = \begin{bmatrix}1.4366766 & ; & 1.4949782 & ; & 1.2953370\end{bmatrix}$$
$$a_{lr}^{<5>} = \Gamma_o \odot \tanh(c_{lr}^{<5>}) = \begin{bmatrix}0.7100788 & ; & 0.7280917 & ; & 0.5543449\end{bmatrix}$$

Tầng LSTM từ phải sang trái:
Xử lý từ thứ nhất $(t = 1, x_t = e_O)$:

$$\widetilde{c}_{rl}^{<1>} = \tanh(W_{ac}a_{rl}^{<0>} + W_{xc}x^{<1>}) = \begin{bmatrix}0.7162978 & ; & 0.6640367 & ; & 0.2913126\end{bmatrix}$$
$$\Gamma_u = \sigma(W_{au}a_{rl}^{<0>} + W_{xu}x^{<1>}) = \begin{bmatrix}0.5744425 & ; & 0.7109495 & ; & 0.6899744\end{bmatrix}$$
$$\Gamma_f = \sigma(W_{af}a_{rl}^{<0>} + W_{xf}x^{<1>}) = \begin{bmatrix}0.5498339 & ; & 0.5249791 & ; & 0.5249791\end{bmatrix}$$
$$\Gamma_o = \sigma(W_{ao}a_{rl}^{<0>} + W_{xo}x^{<1>}) = \begin{bmatrix}0.6899744 & ; & 0.6456563 & ; & 0.5498339\end{bmatrix}$$
$$c_{rl}^{<1>} = \Gamma_u \odot \widetilde{c}_{rl}^{<1>} + \Gamma_f \odot c^{<0>} = \begin{bmatrix}0.4114719 & ; & 0.4720966 & ; & 0.2009982\end{bmatrix}$$
$$a_{rl}^{<1>} = \Gamma_o \odot \tanh(c_{rl}^{<1>}) = \begin{bmatrix}0.2688980 & ; & 0.2840189 & ; & 0.1090510\end{bmatrix}$$

Xử lý từ thứ hai $(t = 2, x_t = e_L)$:

$$\widetilde{c}_{rl}^{<2>} = \tanh(W_{ac}a_{rl}^{<1>} + W_{xc}x^{<2>}) = \begin{bmatrix}0.3185355 & ; & 0.4884390 & ; & 0.6517254\end{bmatrix}$$
$$\Gamma_u = \sigma(W_{au}a_{rl}^{<1>} + W_{xu}x^{<2>}) = \begin{bmatrix}0.8094322 & ; & 0.6201977 & ; & 0.6500164\end{bmatrix}$$
$$\Gamma_f = \sigma(W_{af}a_{rl}^{<1>} + W_{xf}x^{<2>}) = \begin{bmatrix}0.7370244 & ; & 0.7173200 & ; & 0.6191005\end{bmatrix}$$
$$\Gamma_o = \sigma(W_{ao}a_{rl}^{<1>} + W_{xo}x^{<2>}) = \begin{bmatrix}0.6294869 & ; & 0.6972017 & ; & 0.5691275\end{bmatrix}$$
$$c_{rl}^{<2>} = \Gamma_u \odot \widetilde{c}_{rl}^{<2>} + \Gamma_f \odot c^{<1>} = \begin{bmatrix}0.5610978 & ; & 0.6415731 & ; & 0.5480703\end{bmatrix}$$
$$a_{rl}^{<2>} = \Gamma_o \odot \tanh(c_{rl}^{<2>}) = \begin{bmatrix}0.3202776 & ; & 0.3945951 & ; & 0.2840359\end{bmatrix}$$

Xử lý từ thứ ba $(t = 3, x_t = e_L)$:

$$\widetilde{c}_{rl}^{<3>} = \tanh(W_{ac}a_{rl}^{<2>} + W_{xc}x^{<3>}) = \begin{bmatrix}0.4251778 & ; & 0.6133772 & ; & 0.6912246\end{bmatrix}$$
$$\Gamma_u = \sigma(W_{au}a_{rl}^{<2>} + W_{xu}x^{<3>}) = \begin{bmatrix}0.8364735 & ; & 0.6458172 & ; & 0.6738372\end{bmatrix}$$
$$\Gamma_f = \sigma(W_{af}a_{rl}^{<2>} + W_{xf}x^{<3>}) = \begin{bmatrix}0.7515546 & ; & 0.7457359 & ; & 0.6721700\end{bmatrix}$$
$$\Gamma_o = \sigma(W_{ao}a_{rl}^{<2>} + W_{xo}x^{<3>}) = \begin{bmatrix}0.6579102 & ; & 0.7338643 & ; & 0.5866896\end{bmatrix}$$
$$c_{rl}^{<3>} = \Gamma_u \odot \widetilde{c}_{rl}^{<3>} + \Gamma_f \odot c^{<2>} = \begin{bmatrix}0.7773457 & ; & 0.8745737 & ; & 0.8341694\end{bmatrix}$$
$$a_{rl}^{<3>} = \Gamma_o \odot \tanh(c_{rl}^{<3>}) = \begin{bmatrix}0.4284184 & ; & 0.5164133 & ; & 0.4005379\end{bmatrix}$$

Xử lý từ thứ tư $(t = 4, x_t = e_E)$:

$$\widetilde{c}_{rl}^{<4>} = \tanh(W_{ac}a_{rl}^{<3>} + W_{xc}x^{<4>}) = \begin{bmatrix}0.7499545 & ; & 0.8832333 & ; & 0.6817105\end{bmatrix}$$
$$\Gamma_u = \sigma(W_{au}a_{rl}^{<3>} + W_{xu}x^{<4>}) = \begin{bmatrix}0.8668394 & ; & 0.7558515 & ; & 0.7563547\end{bmatrix}$$
$$\Gamma_f = \sigma(W_{af}a_{rl}^{<3>} + W_{xf}x^{<4>}) = \begin{bmatrix}0.6477553 & ; & 0.7909337 & ; & 0.8070309\end{bmatrix}$$
$$\Gamma_o = \sigma(W_{ao}a_{rl}^{<3>} + W_{xo}x^{<4>}) = \begin{bmatrix}0.7451387 & ; & 0.8006375 & ; & 0.6753104\end{bmatrix}$$
$$c_{rl}^{<4>} = \Gamma_u \odot \widetilde{c}_{rl}^{<4>} + \Gamma_f \odot c^{<3>} = \begin{bmatrix}1.1536200 & ; & 1.3593232 & ; & 1.1888155\end{bmatrix}$$
$$a_{rl}^{<4>} = \Gamma_o \odot \tanh(c_{rl}^{<4>}) = \begin{bmatrix}0.6102312 & ; & 0.7015474 & ; & 0.5606502\end{bmatrix}$$

Xử lý từ thứ năm $(t = 5, x_t = e_H)$:

$$\widetilde{c}_{rl}^{<5>} = \tanh(W_{ac}a_{rl}^{<4>} + W_{xc}x^{<5>}) = \begin{bmatrix}0.8957218 & ; & 0.7835411 & ; & 0.8206256\end{bmatrix}$$
$$\Gamma_u = \sigma(W_{au}a_{rl}^{<4>} + W_{xu}x^{<5>}) = \begin{bmatrix}0.8524256 & ; & 0.7369580 & ; & 0.6887070\end{bmatrix}$$
$$\Gamma_f = \sigma(W_{af}a_{rl}^{<4>} + W_{xf}x^{<5>}) = \begin{bmatrix}0.6618713 & ; & 0.7966633 & ; & 0.8093657\end{bmatrix}$$
$$\Gamma_o = \sigma(W_{ao}a_{rl}^{<4>} + W_{xo}x^{<5>}) = \begin{bmatrix}0.6792106 & ; & 0.7780740 & ; & 0.6589750\end{bmatrix}$$
$$c_{rl}^{<5>} = \Gamma_u \odot \widetilde{c}_{rl}^{<5>} + \Gamma_f \odot c^{<4>} = \begin{bmatrix}1.5270843 & ; & 1.6603599 & ; & 1.5273572\end{bmatrix}$$

$$a_{rl}^{<5>} = \Gamma_o \odot \tanh(c_{rl}^{<5>}) = \begin{bmatrix} 0.6180302 & ; & 0.7238155 & ; & 0.5996484 \end{bmatrix}$$

Tổng hợp kết quả:

$$\hat{y}^{<1>} = softmax(W_{ay}(a_{lr}^{<1>} \odot a_{rl}^{<5>})) = \begin{bmatrix} 0.24 & ; & 0.25 & ; & 0.24 & ; & \mathbf{0.26} \end{bmatrix}$$

$$\hat{y}^{<2>} = softmax(W_{ay}(a_{lr}^{<2>} \odot a_{rl}^{<4>})) = \begin{bmatrix} 0.23 & ; & 0.26 & ; & 0.22 & ; & \mathbf{0.29} \end{bmatrix}$$

Như vậy, $\hat{y}^{<1>}$ và $\hat{y}^{<2>}$, sau khi giải mã sẽ ra ký tự "O".

1.5 Kết chương

Chương này trình bày các biến thể nâng cao của RNN, bao gồm LSTM và GRU. Các mô hình này đều sử dụng cơ chế cổng kiểm soát lưu lượng thông tin truyền qua các đơn vị thời gian. Xét theo thứ tự thời gian thì mô hình LSTM được giới thiệu trước, sử dụng ba cổng "forget", "input" và "output". Đặc biệt LSTM giới thiệu khái niệm memory cell để lưu trữ thông tin về ký ức tạm thời trong một khoảng thời gian dài khi cần thiết. GRU có thể xem như một mô hình thu gọn của LSTM, sử dụng ít cổng hơn và sử dụng hidden state chung với memory cell như mô hình RNN truyền thống. Nhờ vậy, GRU sẽ có thời gian huấn luyện nhanh hơn và dùng ít tài nguyên hơn. Về độ chính xác, GRU được xem là có thể cạnh tranh với LSTM nhưng điều này nhìn chung là phụ thuộc vào bản chất dữ liệu được dùng trong các ứng dụng thực tế.

BiLSTM (ít phổ biến hơn là BiRNN/BiGRU) là một hướng mở rộng khác của các mô hình để cho phép tổng hợp thông tin từ hai chiều khi xử lý dữ liệu đầu vào. Tuy nhiên, cơ chế xử lý trên các mô hình này thực chất vẫn là một "phép cộng cơ học" các thông tin được tính lũy từ hai chiều tại một thời điểm dữ liệu đầu vào được xử lý. Mô hình Transformer đưa ra cơ chế self-attention cho phép tổng hợp thông tin từ hai chiều của dữ liệu nhập một cách tự nhiên và hợp lý hơn. Để hiểu hơn cơ chế hoạt động này của mô hình Transformer, chúng ta cần hiểu về mô hình

Sequence-to-Sequence và cơ chế *attention*. Các khái niệm này sẽ được trình bày chi tiết ở các chương sau.

1.6 Bài tập

Giả sử ta có một kho ngữ liệu (*vocab*) gồm các từ *{"G","O","D","B","Y","E"}* và muốn huấn luyện mô hình có thể phân tích chuỗi nhập vào *"GOO"* và dự đoán được từ tiếp theo là *"D"* (để thu được từ *"GOODBYE"*). Một số dữ kiện đầu vào:

$$W_{xu} = \begin{bmatrix} 0.4 & 0.9 & 0.9 & 0.3 & 0.1 & 0.2 \\ 0.3 & 0.6 & 0.2 & 0.9 & 0.3 & 0.1 \\ 0.2 & 0.7 & 0.4 & 0.8 & 0.5 & 0.6 \end{bmatrix}, W_{au} = W_{cu} = \begin{bmatrix} 1.0 & 0.9 & 0.2 \\ 0.3 & 0.7 & 0.1 \\ 0.1 & 0.6 & 0.2 \end{bmatrix},$$

$$W_{xo} = \begin{bmatrix} 0.1 & 0.6 & 0.3 & 0.8 & 0.4 & 0.7 \\ 0.3 & 0.7 & 0.5 & 0.6 & 0.3 & 0.9 \\ 0.2 & 0.4 & 0.1 & 0.2 & 0.2 & 0.3 \end{bmatrix}, W_{ac} = W_{ao} = \begin{bmatrix} 0.1 & 0.6 & 0.3 \\ 0.3 & 0.7 & 0.5 \\ 0.2 & 0.4 & 0.1 \end{bmatrix},$$

$$W_{xf} = W_{xr} = \begin{bmatrix} 0.1 & 0.2 & 0.8 & 0.2 & 0.8 & 0.1 \\ 0.5 & 0.7 & 0.6 & 0.1 & 0.2 & 0.9 \\ 0.3 & 0.6 & 0.1 & 0.1 & 0.1 & 0.7 \end{bmatrix}, W_{af} = W_{cr} = \begin{bmatrix} 0.5 & 0.3 & 0.1 \\ 0.2 & 0.9 & 0.2 \\ 0.2 & 0.9 & 0.7 \end{bmatrix},$$

$$W_{xa} = W_{xc} = \begin{bmatrix} 0.8 & 0.5 & 0.1 & 0.9 & 0.2 & 0.3 \\ 0.1 & 0.7 & 0.2 & 0.8 & 0.5 & 0.2 \\ 0.7 & 0.5 & 0.6 & 0.3 & 0.6 & 0.1 \end{bmatrix}, W_{aa} = W_{cc} = \begin{bmatrix} 0.2 & 0.5 & 0.3 \\ 0.1 & 0.6 & 0.5 \\ 0.3 & 0.7 & 0.9 \end{bmatrix},$$

$$W_{ay} = W_{cy} = \begin{bmatrix} 0.1 & 0.6 & 0.3 \\ 0.3 & 0.7 & 0.5 \\ 0.2 & 0.4 & 0.1 \\ 0.8 & 0.9 & 0.2 \\ 0.2 & 0.1 & 0.6 \\ 0.5 & 0.3 & 0.2 \end{bmatrix},$$

- Trong bài tập này, chúng ta bỏ qua bias và đặt các giá trị khởi tạo là $a^{<0>} = c^{<0>} = \begin{bmatrix} 0 & ; & 0 & ; & 0 \end{bmatrix}$.

Bài tập 1.6.1. Dùng kỹ thuật one-hot véc-tơ để mã hoá các ký tự trong kho ngữ liệu đã cho, sau đó diễn giải quá trình áp dụng mô hình GRU để dự đoán ký tự tiếp theo cho câu đầu vào là "GOO".

Bài tập 1.6.2. Dùng kỹ thuật one-hot véc-tơ để mã hoá các ký tự trong kho ngữ liệu đã cho, sau đó diễn giải quá trình áp dụng mô hình LSTM để dự đoán ký tự tiếp theo cho câu đầu vào là "GOO".

Bài tập 1.6.3. Dùng kỹ thuật one-hot véc-tơ để mã hoá các ký tự trong kho ngữ liệu đã cho, sau đó diễn giải quá trình áp dụng mô hình BiLSTM để dự đoán ký tự tiếp theo cho câu đầu vào là "GOO". Giả sử các ma trận trọng số là bằng nhau ở hai tầng LSTM.

Chương 2

Mô hình Sequence-to-Sequence và cơ chế Attention

2.1 Giới thiệu

Mô hình *Sequence-to-sequence* (Seq2Seq) được giới thiệu lần đầu tiên trong bài báo "Sequence to Sequence with Neural Networks" (Sutskever et al., 2014) bởi Google. Seq2Seq là một mô hình học sâu với dữ liệu đầu vào là một *chuỗi* (sequence) và tạo ra một chuỗi khác mà trong đó độ dài của 2 chuỗi không nhất thiết phải bằng nhau hoặc có liên quan toán học với nhau, một đặc điểm khác cơ bản với các mô hình dựa trên RNN trước đó như đã trình bày trong Chương 1. Một bài toán phổ biến mà mô hình Seq2Seq được áp dụng nhiều là bài toán dịch tự động hay *dịch máy* (machine translation), thường là từ một ngôn ngữ tự nhiên này sang một ngôn ngữ tự nhiên khác, ví dụ như hiện nay Google Dịch là một ứng dụng nổi tiếng cho bài toán này. Bên cạnh các hệ thống dịch máy nổi tiếng chuyển đổi giữa các ngôn ngữ tự nhiên, bài toán dịch máy còn bao gồm nhiều bài toán khác như chuyển đổi giữa các ngôn ngữ lập trình.

Về mặt trực giác, kiến trúc Encoder-Decoder phù hợp cho bài toán dịch máy vì cơ chế hoạt động khá giống cách con người dịch ngôn ngữ: Thông thường chúng ta sẽ không dịch từng từ một theo cách dùng từ điển mà sẽ đọc từng câu hay từng đoạn của ngôn ngữ nguồn, qua đó tổng hợp thông tin cần thiết rồi mới tiến hành dịch sang ngôn ngữ đích.

2.2 Mô hình Sequence-to-sequence cơ bản

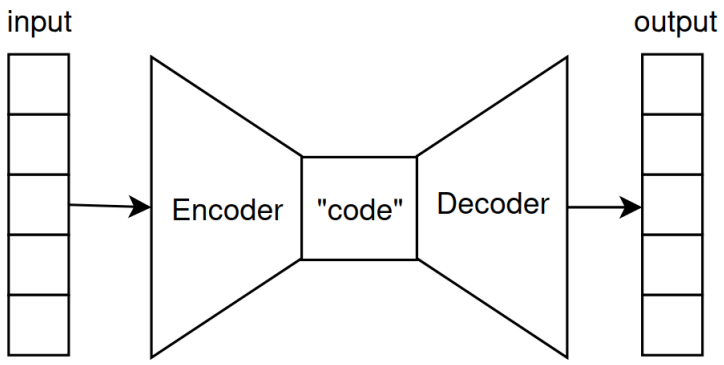
Trong bài toán dịch máy, chúng ta nhận vào đầu vào là một chuỗi $x = \{x_1, x_2, \ldots, x_m\}$, và đầu ra $y = \{y_1, y_2, \ldots, y_n\}$ là một chuỗi mà độ dài có thể khác độ dài của chuỗi đầu vào. Bài toán dịch máy có thể hiểu là đi tìm chuỗi đầu ra phù hợp nhất với chuỗi đầu vào, tức là chuỗi y cần tìm là chuỗi tối đa xác suất điều kiện $y^* = \underset{y}{\operatorname{argmax}}\, p(y|x)$ trong đó y^* là chuỗi *ground truth*, là kết quả dịch được mong đợi.

Trong ngữ cảnh dịch máy sử dụng một kiến trúc mạng học sâu, chúng ta học một hàm $p(y|x, \theta)$ với tham số θ là bộ các *tham số* (parameter) của mạng học sâu tương ứng. Với cùng một dữ liệu huấn luyện như nhau, các chiến lược huấn luyện khác nhau có thể cho ra các giá trị tham số khác nhau dẫn đến các kết quả dịch khác nhau. Vì vậy chúng ta sẽ viết lại bài toàn dịch máy tổng quát là $y^* = \underset{y}{\operatorname{argmax}}\, p(y|x, \theta)$.

2.3 Kiến trúc Encoder-Decoder

Encoder-Decoder (Kiros et al., 2015) là một kiến trúc mạng học sâu tiêu chuẩn cho bài toán dạng Sequence-to-sequence. Về cơ bản, kiến trúc Encoder-Decoder gồm 2 thành phần như ở Hình 2.1:

- Thành phần Encoder: nhận dữ liệu đầu vào để tạo thành một dạng biểu diễn trung gian tương ứng. Quá trình này gọi là encoding, dạng dữ liệu trung gian tạo ra thường là một véc-tơ, gọi là "code" của dữ liệu đầu vào. Sau khi được huấn luyện, thành phần Encoder này có thể sử dụng độc lập để tạo ra "code" của bất kỳ dữ liệu



Hình 2.1: Kiến trúc Encoder-Decoder

nhập nào, kể cả dữ liệu nhập chưa xuất hiện trong tập huấn luyện trước đó.

- Thành phần Decoder: Nhận kết quả đầu ra của Encoder ("code" của dữ liệu đầu vào), từ đó tạo ra chuỗi đầu ra tương ứng. Quá trình này gọi là decoding. Trong bài toán học có giám sát, kết quả đầu ra sẽ được so sánh với nhãn tương ứng và quá trình học back-propagation sẽ được lan truyền qua cả thành phần Decoder lẫn Decoder để cập nhật các trọng số của chúng.

Các thành phần Encoder và Decoder có thể là các kiến trúc mạng nơ-ron bất kỳ (thường là mạng đa tầng). Tuy nhiên khi vận dụng vào bài toán Sequence-to-sequence, do đặc thù của dữ liệu nhập là dạng chuỗi (sequence) nên các thành phần Encoder và Decoder sẽ là các kiến trúc mạng phù hợp để xử lý dạng dữ liệu này. Một trong những hướng tiếp cận phổ biến là sử dụng các kiến trúc dựa trên mạng RNN đã được trình bày ở chương trước cho các thành phần Encoder và Decoder.

2.4 Mô hình ngôn ngữ có điều kiện

2.4.1 Mô hình ngôn ngữ

Mô hình ngôn ngữ cơ bản là một phân phối xác suất trên các chuỗi từ. Trong thực tế, một mô hình ngôn ngữ ước tính xác suất của một từ trong ngôn ngữ của nó. Với một chuỗi có độ dài m, mô hình ngôn ngữ gán xác suất $P(w_1, w_2, ..., w_m)$ tới toàn bộ chuỗi. Để tính xác suất này, chúng ta phân tách thành các xác suất của các phần nhỏ hơn. Ví dụ, chúng ta có câu *I go to school*, và theo tự nhiên chúng ta đọc từ trái qua phải. Tại mỗi bước, chúng ta ước tính của một từ khi đã biết các từ trước đó.

$$p(I\ go\ to\ school) = p(I).p(go|I).p(to|I\ go).p(school|I\ go\ to) \tag{2.1}$$

Một cách tổng quát, ký hiệu $y_1, y_2, ..., y_n$ là các từ trong một chuỗi, và $p(y_1, y_2, ..., y_n)$ là xác suất để thấy tất cả các từ này. Sử dụng chain rule, chúng ta viết lại là

$$p(y_1, y_2, ..., y_n) = p(y_1).p(y_2|y_1).....p(y_n|y_1, ..., y_{n-1}) = \prod_{1}^{n} p(y_t|y_{<t}) \tag{2.2}$$

Tuy nhiên, trong thực tế để tính giá trị $p(y_1, y_2, ..., y_n)$ là khó khăn và tốn nhiều chi phí về thời gian và tính toán. Để khắc phục vấn đề này, mô hình n-gram được đề xuất, và hiện tại đây được xem là một trong những hướng tiếp cận phổ biến để xây dựng mô hình ngôn ngữ . Mô hình n-gram giả định rằng mô hình ngôn ngữ là một chuỗi Markov với giả thiết rằng 1 từ chỉ phụ thuộc vào n từ đứng trước nó. Cụ thể, chúng ta có thể viết lại công thức 2.2 như sau:

$$\begin{aligned} p(y_1, y_2, ..., y_n) &= p(y_1).p(y_2|y_1).p(y_3|y_1, y_2).....p(y_n|y_1, ..., y_{n-1}) \\ &= p(y_1).p(y_2|y_1).p(y_3|y_2)....p(y_n|y_{n-1}) \end{aligned} \tag{2.3}$$

Công thức 2.3 là một mô hình Markov bậc 1, dạng đơn giản nhất. Tuy nhiên, giả định này lại khá đơn giản với n được chọn là 1. Trong thực

tế các mô hình *bigram* (2-gram) hay *trigram* (3-gram) thường được sử dụng hơn.

2.4.2 Mô hình ngôn ngữ có điều kiện

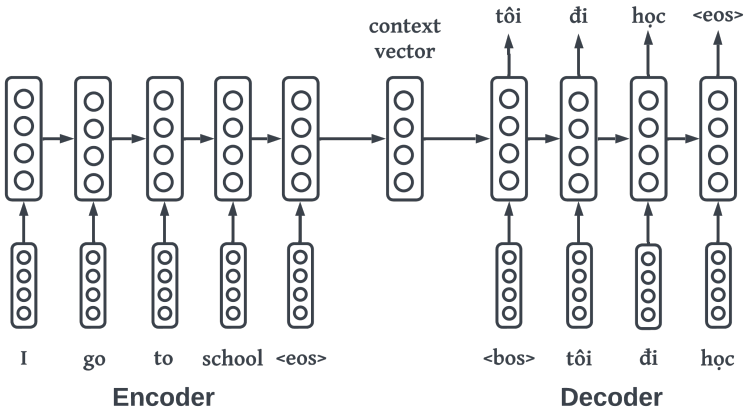
Như đã trình bày ở trên, mô hình ngôn ngữ ước tính xác suất không điều kiện $p(y)$ của chuỗi y, trong khi đó mô hình Sequence-to-sequence ước tính xác suất có điều kiện $p(y|x)$ của chuỗi y khi biết chuỗi x. Điều này nhìn khá giống với mô hình ngôn ngữ nhưng có thêm dữ liệu đầu vào x. Việc biểu diễn thông tin của x và vào quá trình decoding có thể thực hiện theo cơ chế hoạt động của một mạng RNN bình thường, hoặc đặc biệt hơn có thể dùng cơ chế *attention*, như sẽ trình bày ở các phần sau.

$$\begin{aligned} p(y_1, y_2, ..., y_n|x) &= p(y_1|x).p(y_2|y_1, x).....p(y_n|y_1, ..., y_{n-1}, x) \\ &= \prod_{1}^{n} p(y_t|y_{<t}, x) \end{aligned} \tag{2.4}$$

2.5 Mô hình Seq2Seq với RNN

Chúng ta xem xét một mô hình Seq2Seq dạng đơn giản nhất là mô hình dựa trên RNN, cụ thể cả Encoder và Decoder đều hiện thực dưới dạng một mô hình RNN. Mô hình này giải quyết bài toán dịch máy như đã thảo luận ở phần trước. Encoder nhận vào một chuỗi nhập từ ngôn ngữ nguồn và tuần tự cập nhật hidden state ứng với từng từ (gọi là các *token*) trong chuỗi nhập như đã mô tả trong phần trước. Encoder sẽ tạo ra hidden state là đầu ra của từng bước xử lý. Ngoài các token trong chuỗi nhập, chúng ta sẽ thêm vào một token đặc biệt gọi là <eos> (*end-of-string*) vào cuối chuỗi nhập. Token <eos> dùng để chỉ thị đó là token kết thúc của chuỗi, hidden state đầu ra của khối Encoder ứng với token cuối cùng này sẽ đóng vai trò *ngữ cảnh* (context) hoặc là "code" của chuỗi nhập để chuyển cho Decoder bắt đầu quá trình dịch sang câu đích.

Để thực hiện việc dịch chuỗi nhập sang ngôn ngữ đích, Decoder sẽ thực



Hình 2.2: Minh họa về mô hình Seq2Seq dựa trên RNN

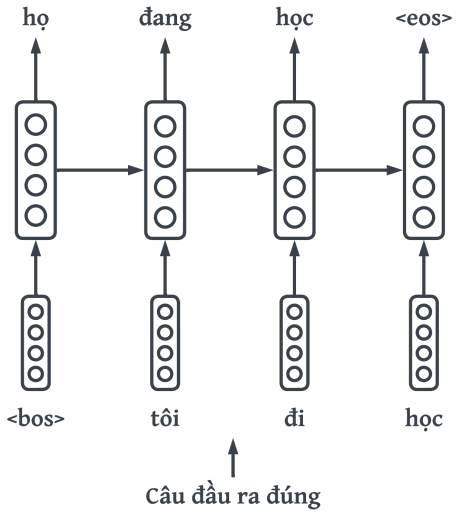
hiện quá trình *sinh chuỗi* (string generation) của một *mô hình ngôn ngữ* (language model). Về cơ bản, ở mỗi bước xử lý, Decoder sẽ xuất ra output ứng với từng từ trong ngôn ngữ đích và đồng thời cập nhật lại hidden state của mình. Quá trình này tương tự với quá trình tiên đoán từ (hoặc token) kế tiếp dựa trên chuỗi nhập hiện hành như đã mô tả ở chương trước. Tuy nhiên, thay vì khởi động hidden state là một véc-tơ rỗng, Decoder sẽ sử dụng véc-tơ context được tạo ra bởi Encoder. Ký tự nhập đầu tiên cho Decoder sẽ là một token đặc biệt, gọi là <bos> (*begin-of-string*). Decoder sẽ thực hiện quá trình sinh chuỗi cho đến khi token <eos> được sinh ra. Việc xuất hiện của token <eos> này sẽ do Decoder quyết định dựa vào xác suất có điều kiện của mô hình ngôn ngữ tương ứng đã được huấn luyện, vì vậy chiều dài của chuỗi ngôn ngữ đích tạo ra sẽ không nhất thiết phải bằng chiều dài hoặc phụ thuộc tuyến tính vào chiều dài của ngôn ngữ nguồn được nhập vào. Toàn bộ quá trình trên được minh họa như Hình 2.2 khi dịch một câu tiếng Anh sang tiếng Việt. Trong quá trình này, các dữ liệu nhập mà Decoder thật sự nhận vào để tạo sinh chuỗi ngôn ngữ đích sẽ tùy thuộc vào chiến lược của mô hình. Các chiến lược này sẽ được trình bày trong phần tiếp theo.

2.6 Các chiến lược huấn luyện mô hình Seq2Seq

Để huấn luyện mô hình Seq2Seq, hàm mất mát thường được sử dụng là *cross entropy* để so sánh giữa các từ được mô hình xuất ra so với các từ tương ứng trong ground truth, tương tự việc huấn luyện các mô hình ngôn ngữ. Có hai chiến lược huấn luyện chính cho mô hình này, bao gồm *teacher forcing* và ít phổ biến hơn là *non teacher forcing*.

2.6.1 Chiến lược huấn luyện teacher forcing

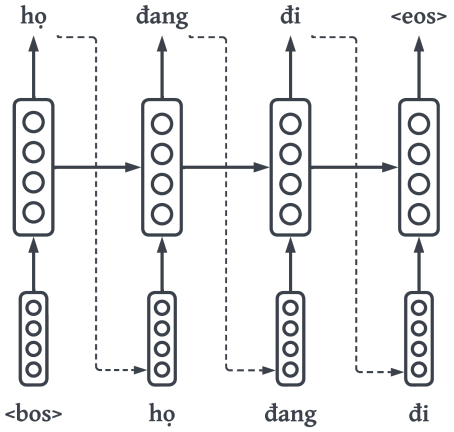
Teacher forcing là kỹ thuật thường được sử dụng trong quá trình huấn luyện các mô hình Sequence-to-sequence. Ý tưởng chính của chiến lược này là đưa giá trị đúng trên ground truth vào Decoder sau mỗi bước, từ đó hy vọng đầu ra của mô hình sẽ khớp với giá trị nhãn. Với chiến lược này, để dự đoán từ tiếp theo, chúng ta sẽ dùng các giá trị đúng ở các thời điểm trước như đầu vào mà không xem xét đầu ra tương ứng của mô hình tại các thời điểm trước, đó cũng là vì sao kỹ thuật này có tên teacher forcing, tức là mô hình bị "forced" phải dùng dữ liệu từ ground truth được cung cấp bởi "teacher" làm dữ liệu ngõ nhập.



Hình 2.3: Minh họa về chiến lược teacher forcing

Hình 2.3 mô tả chiến lược này (ở đây chúng ta chỉ xem xét phần Decoder của mô hình) với ví dụ dịch từ "I go to school" sang tiếng Việt là "tôi đi học". Sau khi dự đoán mô hình dự đoán từ đầu tiên là "họ", từ này không đúng với từ đầu tiên của câu đầu ra là từ "tôi". Khi áp dụng chiến lược teacher forcing, Decoder dùng từ đúng là từ "tôi" để làm ngõ nhập tiếp theo. Tương tự, ở từ thứ hai, Decoder dự đoán là "đang" trong khi từ tương ứng ở câu đầu ra đúng là "đi". Decoder nhận từ "đi" làm ngõ nhập tiếp theo, còn từ dự đoán "đang" sẽ chỉ được dùng để tính giá trị loss và cập nhật trọng số cho mô hình.

2.6.2 Chiến lược huấn luyện non-teacher forcing



Hình 2.4: Minh họa về chiến lược non-teacher forcing

Chiến lược non-teacher forcing khác với teacher forcing là sử dụng đầu ra của mô hình ở bước trước đó để làm đầu vào cho bước tiếp theo. Hình 2.4 minh họa chiến thuật này với ví dụ dịch từ "I go to school" sang tiếng Việt là "tôi đi học". Ở bước đầu tiên, Decoder dự đoán từ tiếp theo là "họ" trong khi từ đúng là "tôi". Khi sử dụng non-teacher forcing thì Decoder dùng từ "họ" làm ngõ nhập tiếp theo thay vì từ "tôi" khi ở chiến thuật teacher forcing. Điều này áp dụng tương tự ở các bước tiếp theo. Trong chiến thuật này, giá trị ground truth thường được sử dụng

để tính giá trị loss và cập nhật trọng số trong quá trình huấn luyện.

Nhìn chung, chiến thuật teacher forcing thường được sử dụng rộng rãi hơn trong các mô hình Seq2seq. Chiến thuật này sử dụng giá trị ground truth làm đầu vào để dự đoán ra kết quả, do đó sẽ giúp tăng tốc độ huấn luyện mô hình. Khi mới bắt đầu huấn luyện một mô hình Seq2Seq, dự báo của mô hình thường là không tốt. Nếu không áp dụng teacher forcing, mô hình phải thực hiện một quá trình lựa chọn thử sai rất lâu cho đến khi tìm được câu dự báo đúng. Mặc dù vậy, teacher forcing lại làm cho mô hình bị phụ thuộc vào giá trị đúng, điều này có thể dẫn tới khi hoạt động trong thực tế hoặc khi đánh giá trên tập test, kết quả của mô hình có thể bị ảnh hưởng một cách đáng kể.

2.7 Cơ chế attention

Như đã trình bày, trên mô hình Seq2seq, quá trình Decoder sinh chuỗi token ứng với ngôn ngữ đích sẽ dựa trên việc tính toán xác suất có điều kiện, trong đó xác suất xuất hiện của một token tại thời điểm hiện hành sẽ dựa vào thông tin của các token đã được tạo ra trước đó cùng với thông tin của chuỗi nhập x của ngôn ngữ nguồn ban đầu.

Trong những mô hình chúng ta đã xem xét, thông tin về chuỗi nhập x mà Decoder nhận được từ Encoder là một véc-tơ "context", thực chất chính là hidden state ở bước xử lý sau cùng của Encoder. Khi áp dụng để xử lý dịch câu dài trong thực tế, phương pháp này có những hạn chế như sau:

- Trong cơ chế xử lý này, Encoder "nén" toàn bộ câu đầu vào thành một véc-tơ context. Khi ngõ nhập là một câu dài, việc sử dụng chỉ một véc-tơ context để biểu diễn thông tin của tất cả từ trong câu sẽ khó khăn.
- Nếu sử dụng hidden state cuối cùng được tạo ra bởi Encoder làm véc-tơ context , thì véc-tơ này sẽ có xu hướng bị ảnh hưởng nhiều bởi những từ ở cuối câu nhập và bị mất mát nhiều thông tin ở

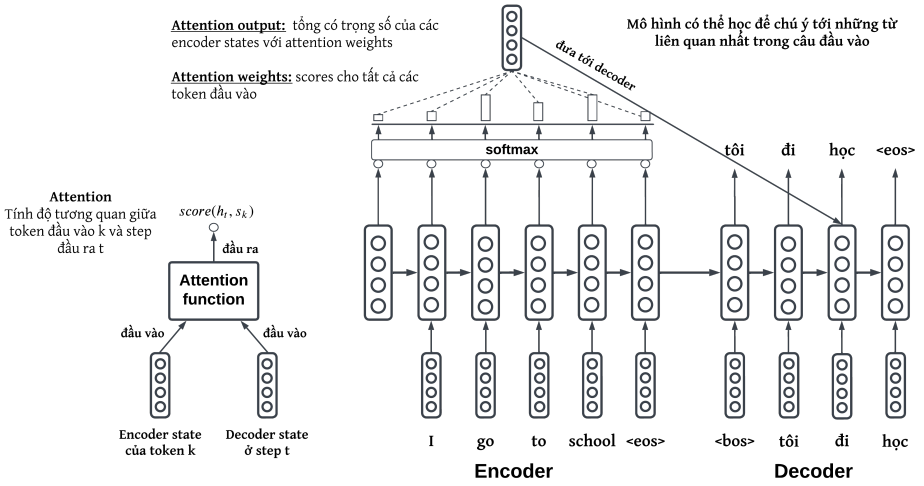
những từ đầu câu nhập.

- Decoder chỉ "thấy" một véc-tơ context của câu đầu vào để cho ra câu đầu ra. Tuy nhiên, tại mỗi bước sinh, các từ của câu đầu vào có thể có ảnh hưởng khác nhau đến kết quả dự báo tại bước đó. Nói cách khác, thông tin của câu đầu vào nên được sử dụng lại ở mỗi bước sinh từ để tạo ra một véc-tơ context mới.

Cơ chế *attention* được giới thiệu ở bài báo "Neural Machine Translation by Jointly Learning to Align and Translate" (Bahdanau et al., 2015a), để giải quyết vấn đề trên. Trong chương này các mình minh họa trong phần này cũng được vẽ lại từ các hình gốc trong bài báo trên. Về mặt tổng quát, cơ chế attention là một phần trong toàn thể kiến trúc mạng nơ-ron của mô hình Seq2seq. Tại mỗi bước sinh từ, cơ chế này sẽ tính toán để tạo ra một véc-tơ context mới, gọi là *véc-tơ attention*, sử dụng hidden state hiện hành và thông tin ban đầu của câu nhập. Đặc biệt, ở mỗi bước tính toán, mô hình sẽ đưa ra một *phân phối xác suất* thể hiện độ quan trọng của từng từ trong câu nhập trong ngữ cảnh hiện hành và dựa vào đó để tổng hợp véc-tơ attention tương ứng.

Về trực giác, ý tưởng này cũng rất phù hợp với cách dịch ngôn ngữ của con người. Ví dụ như dịch câu "I go to school" sang tiếng Việt với câu dịch mong đợi là "Tôi đi học", từ đầu tiên trong câu đích ("Tôi") được dịch tương ứng với từ "I" trong câu nhập. Nói cách khác, từ "I" này đóng vai trò quan trọng nhất khi sinh ra từ đầu tiên của câu đích và cần có được sự *chú ý* (attention) nhiều nhất. Tương tự, tư "go" và "school" sẽ lần lượt đóng vai trò quan trọng nhất khi dịch các từ tiếp theo, còn từ "to" sẽ không có vai trò quan trọng trong bất kỳ bước dịch nào. Tuy nhiên với cách tạo véc-tơ context như các phần trước, thông tin của từ "I" đã bị mất mát khá nhiều khi Encoder xử lý đến từ cuối cùng của câu nhập nên Decoder không có nhiều thông tin của từ này khi quyết định dịch từ đầu tiên.

Cơ chế attention được minh họa ở Hình 2.5. Ở mỗi bước xử lý của Decoder ứng với từng token nhập, các bước tính toán của cơ chế attention sẽ được



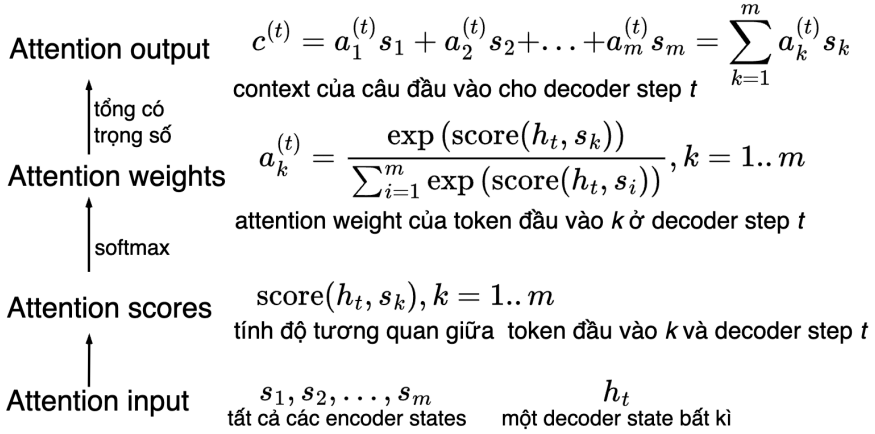
Hình 2.5: Cơ chế Attention

thực hiện như minh họa ở Hình 2.6, cụ thể như sau

1. *Các thông số cho cơ chế attention*: Để thực hiện quá trình tính toán để sinh ra token (từ) thứ t, Decoder trước tiên sẽ tính toán để sinh ra hidden state h_t như một mạng RNN bình thường, đồng thời các hidden state của Encoder $s_1, s_2, ..., s_m$ ứng với m từ của chuỗi nhập cũng sẽ được sử dụng.

2. *Tính attention score.* Với mỗi hidden state của Encoder s_i, Decoder sẽ tính độ *liên quan* (relevance) giữa hidden state này và hidden state hiện hành của Decoder h_t, gọi là *attention score.* Nếu xem như một hàm, *attention score* nhận đầu vào là hidden state hiện hành của Decoder và một hidden state s_k của encoder, sau đó trả về một giá trị số thực $score(h_t, s_k)$. Điểm số *attention score* này sẽ diễn tả **mức độ quan trọng của các từ trong chuỗi nhập** trong thời điểm hiện hành để Decoder quyết định việc sinh ra từ tiếp theo tương ứng.

3. *Tính attention weights* bằng cách áp dụng hàm *softmax* trên các giá trị *attention scores.* Bằng cách này, mô hình tạo ra **một phân phối xác suất ứng với độ quan trọng của các từ** trong chuỗi

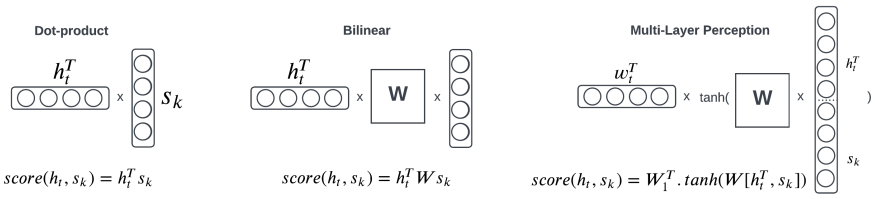


Hình 2.6: Quá trình tính toán của attention

nhập tại thời điểm hiện tại.

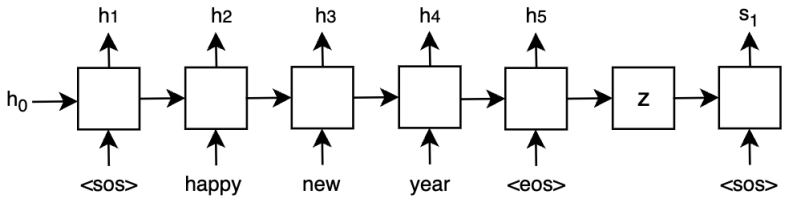
4. *Tính attention output (hay véc-tơ attention)* bằng cách nhân mỗi hidden state của Encoder với *attention weight* tương ứng và tính tổng của chúng. Nói cách khác, véc-tơ attention là tổng có trọng số của các hidden state của Decoder. Véc-tơ attention này sẽ được dùng thay vai trò của hidden state được tạo ra cho các bước tính toán tiếp theo.

Trong các bước trên, quan trọng nhất là bước tính attention score. Hình 2.7 trình bày một số cách tính attention score phổ biến. Cách đơn giản và phổ biến nhất là tính *dot-product* giữa hai véc-tơ $score(h_t, s_k) = h_t^T s_k$. Chú ý rằng về mặt toán học, hàm dot-product được dùng để tính *độ tương tự* (similarity) giữa hai véc-tơ, vì thế độ quan trọng của từng từ trong chuỗi nhập sẽ được tính dựa trên độ tương tự của hidden state tạo ra bởi từ ấy và hidden state hiện hành của Decoder. Đây là một ý tưởng đơn giản nhưng hợp lý và hiệu quả. Các hàm phổ biến khác là bilinear function $score(h_t, s_k) = h_t^T W s_k$, multi-layer perceptron $score(h_t, s_k) = w_2^T \cdot \tanh(W_1[h_t, s_k])$, cũng dựa trên dot-product nhưng sử dụng thêm các trainable parameters để học thêm các tham số hiệu chỉnh trong quá trình huấn luyện.



Hình 2.7: Các hàm để tính attention scores

Ví dụ 2.7.1. Chúng ta xem xét ví dụ sau: giả sử chúng ta muốn xây dựng một mô hình để chuyển từ tiếng Anh sang tiếng Việt. Như Hình 2.8, mô hình nhận đầu vào là câu "happy new year", chúng ta có 2 token đặc biệt là $<bos>$ để chỉ thị cho Decoder là quá trình dịch bắt đầu và $<eos>$ để đánh dấu chuỗi kết thúc.



Hình 2.8: Ví dụ minh họa việc sử dụng Seq2Seq cho bài toán dịch máy.

Sau đó, tất cả các token của chuỗi nhập sẽ được đưa qua một encoder. Đây là một sequence model như RNN, GRU, LSTM đã trình bày trong chương trước. Ứng với mỗi token, chúng ta có được đầu ra hidden state tương ứng cho từng token là $h_i, 1 \leq i \leq 4$, và một véc-tơ context z chứa toàn bộ thông tin của câu đầu vào (tương ứng với hidden state cuối cùng). Sau đó, Decoder nhận véc-tơ z như là hidden state ban đầu. Đồng thời Decoder cũng nhận đầu vào đầu tiên là token $<bos>$ để tạo ra được hidden state ương ứng là s_1, véc-tơ này được sử dụng dự đoán ra token đầu tiên của ngõ xuất. Đồng thời Decoder lại nhận véc-tơ s_1 này làm hidden state và từ nhập tiếp theo để dự đoán tiếp tục. Nếu sử dụng cơ chế teacher forcing, từ nhập tiếp theo sẽ là từ đã được xác định từ ground truth ("Chúc"). Quá trình này diễn ra cho đến khi token được dự đoán

là *<eos>*.

Trong trường hợp mô hình Seq2Seq này có sử dụng cơ chế attention, thay vì trực tiếp sử dụng s_1 để dự đoán ra token tiếp theo, thì lúc này mô hình sẽ tính attention score giữa hidden state hiện tại của decoder và các hidden state của câu đầu vào. Cụ thể là

$$e_{11} = score(s_1, h_1) \tag{2.5}$$

$$e_{12} = score(s_1, h_2) \tag{2.6}$$

$$e_{13} = score(s_1, h_3) \tag{2.7}$$

$$e_{14} = score(s_1, h_4) \tag{2.8}$$

Sau đó, các giá trị $e_{ij}, 1 \leq j \leq 4$ sẽ được chuẩn hóa để có giá trị dương và tổng bằng 1 (tức là tạo ra một phân phối xác suất) thông qua hàm $softmax$. Sau bước này chúng ta thu được giá trị đầu ra tương ứng với các attention weight là a_{11}, a_{12}, a_{13} và a_{14}. Cuối cùng, véc-tơ attention được tính bằng tổng có trọng số $c_1 = a_{11}h_1 + a_{12}h_2 + a_{13}h_3 + a_{14}h_4$. Véc-tơ c_1 này sẽ được sử dụng như là hidden state để dự đoán ra token tiếp theo (một số biến thể khác có thể ghép nối véc-tơ c_1 này và véc-tơ s_1 làm véc-tơ hidden state mới). Quá trình này diễn ra cho đến khi Decoder sinh ra token *<eos>*.

Ví dụ 2.7.2. Giả sử chúng ta muốn dịch câu tiếng Anh "I love you" sang tiếng Việt với chuỗi xuất mong đợi là "Tôi yêu em", và có hidden state tương ứng cho *I*, *love*, *you*, *<eos>* là

$$\begin{bmatrix} 1.1392 \\ -0.1606 \\ -0.6093 \\ 0.5817 \end{bmatrix}, \begin{bmatrix} 0.3265 \\ 0.5722 \\ 0.5170 \\ 0.9927 \end{bmatrix}, \begin{bmatrix} -0.7503 \\ 0.3961 \\ -1.8442 \\ 0.3233 \end{bmatrix}, \begin{bmatrix} -1.5798 \\ -1.0091 \\ 0.5106 \\ 0.0863 \end{bmatrix}.$$

Chúng ta giả sử bộ từ tựng của ngôn ngữ đích gồm có "<bos>", "<eos>", "<Anh>", "Tôi", "yêu", "em". Giả sử chúng ta có hidden state ứng với token đầu tiên "<bos>" trong quá trình decoding là [1.1382, -1.3158, -1.9128, -0.1167]. Đồng thời, ma trận W biến đổi từ hidden state sang

ký tự ngõ xuất của decoder là

$$W = \begin{bmatrix} 0.5828 & 1.8904 & 1.6395 & -0.9807 & 0.2858 & -1.1501 \\ 0.9348 & -0.0655 & -0.8780 & 0.4886 & 1.2109 & -2.2029 \\ 1.0918 & -0.9696 & -0.8569 & 2.0815 & -0.8114 & -1.0813 \\ 2.3575 & 0.5645 & 0.7298 & 0.1513 & 0.2483 & -0.0324 \end{bmatrix}$$

Để dự báo ký tự tiếp theo của *<bos>*, chúng ta sẽ tính phân phối xác suất của các từ trong bộ từ vựng của ngõ xuất như sau:

$$\text{softmax}\left(\begin{bmatrix} 1.1382 \\ -1.3158 \\ -1.9128 \\ -0.1167 \end{bmatrix}^T W \right) \approx \begin{bmatrix} 0.0003 \\ 0.2899 \\ 0.5018 \\ 0.0000 \\ 0.0067 \\ 0.2013 \end{bmatrix}^T$$

Giá trị xác suất lớn nhất trong bộ phân phối xác suất thu được nằm ở vị trí số 2 trong bộ từ vựng, ứng với từ "Anh". Do đó ký tự được dự đoán tiếp theo ký tự "<bos>" là "Anh". Tuy nhiên, với việc giá trị được mong đợi từ ground truth là "Tôi", nếu sử dụng chiến thuật teacher forcing, thì ký tự đầu vào tiếp theo đưa vào decoder là "Tôi". Nếu dùng chiến thuật non-teacher forcing thì decoder nhận đầu vào là "Anh".

Trong trường hợp mô hình sử dụng cơ chế attention, chúng ta sẽ tính attention score cho mỗi hidden state của chuỗi nhập. Để đơn giản, chúng ta dùng hàm score là dot-product, cụ thể:

$$e_{11} = \begin{bmatrix} 1.1382 \\ -1.3158 \\ -1.9128 \\ -0.1167 \end{bmatrix}^T \begin{bmatrix} 1.1392 \\ -0.1606 \\ -0.6093 \\ 0.5817 \end{bmatrix} = 2.61 \tag{2.9}$$

$$e_{12} = \begin{bmatrix} 1.1382 \\ -1.3158 \\ -1.9128 \\ -0.1167 \end{bmatrix}^T \begin{bmatrix} 0.3265 \\ 0.5722 \\ 0.5170 \\ 0.9927 \end{bmatrix} = -1.49 \tag{2.10}$$

$$e_{13} = \begin{bmatrix} 1.1382 \\ -1.3158 \\ -1.9128 \\ -0.1167 \end{bmatrix}^T \begin{bmatrix} -0.7503 \\ 0.3961 \\ -1.8442 \\ 0.3233 \end{bmatrix} = 2.11 \tag{2.11}$$

$$e_{14} = \begin{bmatrix} 1.1382 \\ -1.3158 \\ -1.9128 \\ -0.1167 \end{bmatrix}^T \begin{bmatrix} -1.5798 \\ -1.0091 \\ 0.5106 \\ 0.0863 \end{bmatrix} = -1.46 \tag{2.12}$$

Sau đó, để chuẩn hóa về khoảng $(0, 1)$ tạo thành phân phối xác suất ứng với attention weight, chúng ta sẽ dùng softmax:

$$\text{softmax}([2.61, -1.49, 2.11, -1.46]) = [0.6097, 0.0101, 0.3698, 0.0104]$$

Dễ thấy rằng vào thời điểm này, mô hình sẽ tính toán rằng từ nhập có vai trò quan trọng nhất để dự đoán từ tiếp theo lúc này là "I". Dựa vào đó, véc-tơ attention được tính lại là tổng có có trọng số của các hidden

state đầu vào

$$\begin{bmatrix}0.6097\\0.0101\\0.3698\\0.0104\end{bmatrix}^T \begin{bmatrix}1.1392 & -0.1606 & -0.6093 & 0.5817\\0.3265 & 0.5722 & 0.5170 & 0.9927\\-0.7503 & 0.3961 & -1.8442 & 0.3233\\-1.5798 & -1.0091 & 0.5106 & 0.0863\end{bmatrix}$$
$$= [3.2102, 1.0373, -0.9973, 0.5953]$$

Véc-tơ attention sẽ thay thế vai trò của hidden state hiện hành của decoder để dự báo từ tiếp theo, bằng cách nhân với W rồi đưa qua softmax.

$$\text{softmax}([3.2102, 1.0373, -0.9973, 0.5953]W) \approx [0, 0.9071, 0.0875, 0, 0.0054, 0]$$

Giá trị ngõ ra có xác suất lớn nhất lúc này là "<eos>". Sau một quá trình huấn luyện phù hợp, mô hình sẽ học được cách lựa chọn chính xác hơn là từ "Tôi" khi đang dự đoán từ ở đầu câu và từ quan trọng nhất ở chuỗi nhập lúc này là "I".

2.8 Kết chương

Mô hình Seq2Seq có thể xem là sự kết hợp giữa sequence model và kiến trúc Encoder-Decoder. Mô hình này rất phù hợp với các bài toán dịch máy vì phù hợp với bản chất dữ liệu văn bản và có cơ chế hoạt động khá tương thích với cách con người tiến hành biên dịch. Để giải quyết bài toán phụ thuộc xa, đặc biệt đối với câu nhập dài, cơ chế attention đã được giới thiệu, trong đó độ quan trọng của từng từ trong ngữ liệu nhập sẽ được tính toán lại ở mỗi bước tạo sinh từ ở chuỗi ngôn ngữ đích.

Tuy nhiên, các kiến trúc Seq2Seq dựa trên các mô hình RNN và các biến thể vẫn tiếp tục vấp phải các hạn chế đặc trưng của mô hình RNN. Do các hidden state của RNN cần phải được tạo ra tuần tự nên các kiến trúc này khó huấn luyện song song để tận dụng sức mạnh của GPU. Mặt

khác, các thông tin trên mạng RNN vẫn được xử lý theo một chiều nhất định nên vẫn chưa khai thác hết các quan hệ ngữ nghĩa trong văn bản. Trong chương sau, chúng ta sẽ được giới thiệu mô hình *Transformer* với cơ chế hoạt động khắc phục được các hạn chế trên. Đây cũng được xem là mô hình Seq2Seq tiên tiến nhất hiện nay.

2.9 Bài tập

Chúng ta muốn dịch câu "tôi đi học" từ ngôn ngữ tiếng Việt sang ngôn ngữ tiếng Anh, chúng ta có state sau khối Encoder là

$$\begin{bmatrix} -1.1761 \\ -1.2025 \\ -0.9259 \\ -1.0914 \end{bmatrix}, \begin{bmatrix} 0.0373 \\ 1.5522 \\ -0.1689 \\ -0.6760 \end{bmatrix}, \begin{bmatrix} 0.8226 \\ -0.5918 \\ -2.0503 \\ -0.1851 \end{bmatrix}, \begin{bmatrix} 0.3200 \\ -1.9870 \\ -1.3753 \\ -0.4789 \end{bmatrix}$$

lần lượt ứng với các từ "tôi", "đi", "học", "<eos>".
Về khối Decoder là một RNN, hai ma trận W_{aa} và W_{xa} lần lượt là

$$\begin{bmatrix} -2.0987 & -0.3064 & -0.8512 & 0.4152 \\ -1.27057 & 0.5998 & -1.9213 & -0.4130 \\ 1.0173 & 1.2102 & 1.3374 & 0.6468 \\ -1.6173 & 1.9206 & 0.2223 & -0.2489 \end{bmatrix}$$

$$\begin{bmatrix} 2.3067 & 1.6139 & -1.4595 & 0.6066 \\ -1.3227 & 0.2369 & 0.2688 & 1.6174 \\ 0.0785 & -0.4541 & 0.7865 & -0.2960 \\ 1.0206 & 1.8557 & -0.1730 & 0.1673 \end{bmatrix}$$

, bỏ qua bias. Giả sử tập vocab đầu ra chúng ta theo thứ tự gồm có "<bos>", "<eos>", "I", "want", "to", "play", "go", "school", ma trận

W_{ay} là

$$\begin{bmatrix} 0.3010 & -1.2205 & 1.4681 & 2.5099 & -0.7856 & 0.8051 & -2.3901 & -0.3655 \\ -0.5234 & -2.4317 & 0.4817 & 0.0347 & 0.5490 & 0.2320 & -0.1764 & -1.4044 \\ -1.0700 & 2.1182 & -0.6537 & 0.3701 & -1.3281 & 1.158 & -1.6236 & 1.3916 \\ -0.8260 & -0.6317 & 0.5921 & -1.2974 & -0.2345 & -1.1528 & -0.7147 & -0.2392 \end{bmatrix}$$

, và ma trận embedding cho các từ là

$$\begin{bmatrix} -0.5140 & 0.0684 & -0.8791 & 0.5001 \\ -0.5057 & -0.8027 & 0.6017 & -1.0996 \\ -0.0883 & 0.5239 & -1.3967 & 1.4880 \\ -1.7672 & -0.1177 & -0.1218 & 0.9873 \\ 1.2944 & -0.2124 & -0.2491 & 0.5711 \\ -0.9535 & -0.1137 & 1.1603 & -0.4135 \\ -0.6904 & 2.0438 & -0.4247 & 0.1462 \\ 0.5053 & 0.0353 & 0.3213 & -0.4196 \\ 0.2955 & -1.7005 & -1.6620 & -1.7246 \end{bmatrix}$$

Bài tập 2.9.1. Chúng ta không dùng cơ chế Attention và sử dụng teacher forcing. Hãy cho biết 5 từ dự báo ra đầu tiên.

Bài tập 2.9.2. Chúng ta không dùng cơ chế Attention và sử dụng non teacher forcing. Hãy cho biết 5 từ dự báo ra đầu tiên.

Bài tập 2.9.3. Chúng ta dùng cơ chế Attention và sử dụng teacher forcing. Hãy cho biết 5 từ dự báo ra đầu tiên.

Bài tập 2.9.4. Chúng ta dùng cơ chế Attention và sử dụng non teacher forcing. Hãy cho biết 5 từ dự báo ra đầu tiên.

Chương 3

Transformer và các mô hình ngôn ngữ lớn

3.1 Giới thiệu

Mặc dù mô hình RNN kết hợp với attention được ứng dụng rất nhiều cho các bài toán Seq2Seq và có được những kết quả khá tốt nhưng kiến trúc này vẫn còn tồn tại một số nhược điểm lớn:

- Một nút mạng RNN chỉ nhận được thông tin ngữ nghĩa từ các nút mạng trước đó (các từ nằm bên trái) mà thiếu thông tin của các nút nằm sau (các từ nằm bên phải). Tuy biến thể *Bi-directional RNN* (Schuster & Paliwal, 1997) có thể phần nào giải quyết vấn đề này nhưng làm tăng thời gian tính toán trong khi thông tin vẫn truyền một cách gián tiếp.
- Mô hình tính toán một cách 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 thể bị "quên" hoặc "mất mát" dữ liệu khi thông tin được truyền

qua nhiều nút mạng trong quá trình tính toán do đạo hàm quá lớn hoặc quá bé (*vanishing / exploding gradient* (Hochreiter, 1998)). Điều này gây cản trở trong việc nhúng thông tin ngữ cảnh của các từ trong câu cho véc-tơ biểu diễn từ. Mặc dù một số biến thể của RNN như GRU (Cho et al., 2014) và *LSTM* (Hochreiter & Schmidhuber, 1997) có khả năng chọn lọc để quyết định giữ/bỏ thông tin tại mỗi nút mạng như đã trình bày ở chương trước, nhưng khi trải qua nhiều nút trung gian thì thông tin vẫn bị mất mát nhiều.

Để giải quyết các nhược điểm này, năm 2017, nhóm nghiên cứu của Google Brain đã giới thiệu mô hình *Transformer* (Vaswani et al., 2017). Mô hình này dần được dùng rộng rãi ở lĩnh vực *Xử lý ngôn ngữ tự nhiên* (Natural Language Processing - NLP (Vaswani et al., 2017) và cả *Xử lý ảnh* (Computer Vision - CV) (Y. Liu et al., 2023). Giống như các mạng RNN, Transformer được thiết kế để xử lý dữ liệu chuỗi tuần tự, chẳng hạn như các tác vụ như dịch máy hay tóm tắt tự động trong lĩnh vực NLP. Tuy nhiên, khác với RNN, Transformer không yêu cầu dữ liệu tuần tự được xử lý theo thứ tự. Ví dụ, nếu dữ liệu đầu vào là một câu ngôn ngữ tự nhiên, Transformer không cần phải xử lý phần đầu câu trước phần cuối câu. Do tính năng này, Transformer cho phép nhiều phép tính toán song song và vì vậy giảm thời gian huấn luyện. Đặc biệt cơ chế hoạt động của Transformer cho phép thông tin tại mỗi nút mạng được tổng hợp cả hai chiều cùng lúc theo một cơ chế khá tự nhiên, qua đó tăng hiệu suất đáng kể khi ứng dụng vào các bài toán học máy.

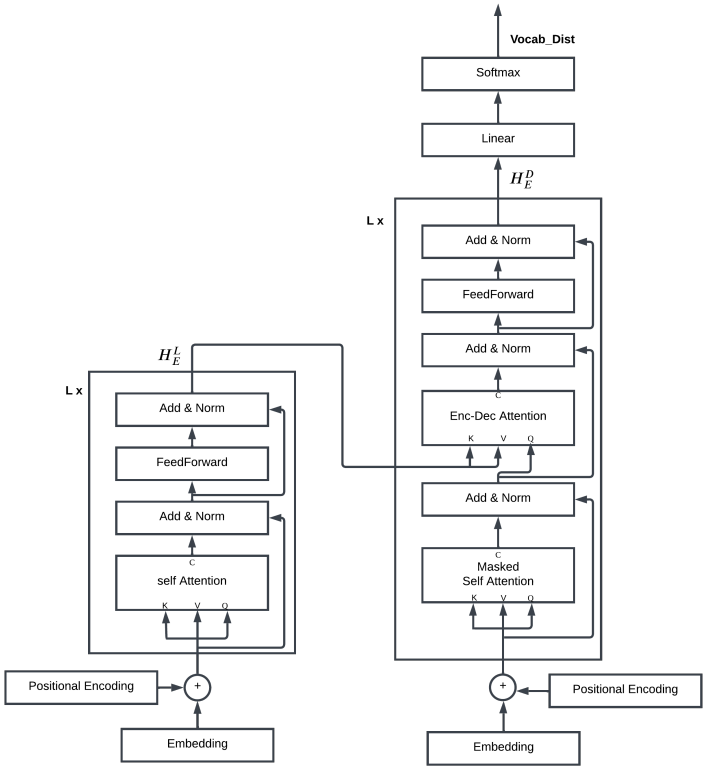
3.2 Kiến trúc mô hình Transformer

Năm 2017, Vaswani và cộng sự đề xuất mô hình Transformer (Vaswani et al., 2017) đã giải quyết các nhược điểm của mô hình RNN. Mô hình Transformer cũng tuân theo kiến trúc seq2seq nhưng với Encoder và Decoder được xây dựng như Hình 3.1. Ngoại trừ các module thường gặp trong các kiến trúc mạng nơ-ron, Transformer giới thiệu ba kỹ thuật mới:

- *Kỹ thuật scaled dot-product attention*: Đây là kỹ thuật cho phép

Transformer thực hiện cơ chế attention theo hai chiều và có khả năng song song hóa. Kỹ thuật này dùng trong các module *Self Attenion*, *Masked Self Attention* và *Encoder-Decoder Attention* (hay *Cross-Attention*).

- *Kỹ thuật Multi-head*: Đây chỉ đơn giản là áp dụng Self-Attention nhiều lần (mỗi lần gọi là một *head*) và tổng hợp kết quả, tương tự cơ chế áp dụng nhiều filter và tổng hợp kết quả của CNN. Nhờ cơ chế attention đặc trưng của mình như đã trình bày, quá trình tính toán Multihead trên Transformer cũng có thể được song song hóa khi tính toán.
- *Cơ chế Positional Encoding*: Cơ chế này cho phép encode thông tin vị trí vào các véc-tơ ngõ nhập.



Hình 3.1: Kiến trúc mô hình Transformer (Vaswani et al., 2017)

3.2.1 Các khối Encoder và Decoder

Khối Encoder: gồm $N = 6$ lớp giống nhau (*encoder layer*) xếp chồng lên nhau. Mỗi lớp được cấu tạo từ 2 lớp con (*sub-layer*): *multi-head self-attention* với chức năng nhúng ngữ nghĩa các từ xung quanh vào các véc-tơ biểu diễn từ và *fully connected feedforward network* với chức năng tính toán, rút trích thông tin. Trên Hình 3.1, hai lớp con này ứng với module *Self Attention*, trong đó hai kỹ thuật *multi-head* và *scaled dot-product attention* sẽ được áp dụng cùng lúc, và module *FeedForward*. Đầu ra của mỗi lớp con này được áp dụng lớp *residual connection* (He et al., 2016) và *layer normalization* (Ba et al., 2016): $LayerNorm(x + Sublayer(x))$ với mục đích hạn chế vanishing gradient, giảm *training error* và giảm thời gian huấn luyện. Để áp dụng residual connection, các lớp con của mô hình đều có đầu ra cùng kích thước $d_{model} = 512$. Trên Hình 3.1, module *Add & Norm* sẽ đảm nhận việc tính toán cho bước residual này.

Khối Decoder: gồm $N = 6$ lớp giống nhau (*decoder layer*) xếp chồng lên nhau. Bên cạnh 2 lớp con như Encoder, Decoder còn có thêm lớp con thứ 3 thực hiện attention với đầu ra của từng từ sau khi qua encoder. Lớp con này đóng vai trò lấy véc-tơ context và *align* giữa chuỗi nguồn và chuỗi đích như trong nghiên cứu của Bahdanau và cộng sự (Bahdanau et al., 2015b). Lớp này tương ứng với module *Decoder-Encoder Attention*. Ngoài ra, ở module này, *Self-Attention* sẽ được biến chuyển thành *Masked Self-Attention*, trong đó Vaswani và cộng sự tiến hành việc che thông tin *masked* để mô phỏng hành vi dự đoán (*predict*) của mô hình ngôn ngữ. Việc này sẽ khiến cho mô hình tránh lấy thông tin ngữ cảnh của các từ tương lai khi huấn luyện.

3.2.2 Kỹ thuật scaled dot-product attention

Đây là cơ chế attention được sử dụng trong Transformer, hoạt động dựa trên nguyên lý biến đổi query và các cặp key-value tương ứng thành

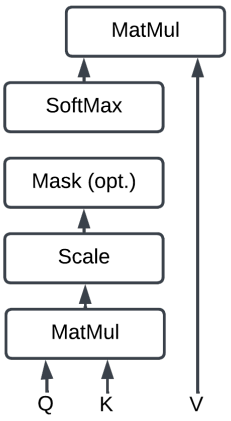
output, với query, key và value là véc-tơ d chiều, do đó output cũng là véc-tơ d chiều. Giá trị output được tính bằng tổng có trọng số của các véc-tơ value, trong đó các trọng số là tích vô hướng của query và key tương ứng chia cho $\sqrt{d}$. Trong thực tế, các véc-tơ query, key và value là các dạng thể hiện khác nhau của từ dưới dạng véc-tơ và được xếp thành các ma trận $Q \in \mathbb{R}^{n \times d}$, $K \in \mathbb{R}^{n \times d}$ và $V \in \mathbb{R}^{n \times d}$ tương ứng, với n là số từ trong chuỗi. Khi đó phương pháp attention được thể hiện bằng Công thức 3.1 và Công thức 3.3.

$$Q = X\mathbf{W}_Q; K = X\mathbf{W}_K; V = X\mathbf{W}_V \tag{3.1}$$

$$A = softmax(\frac{QK^T}{\sqrt{d_k}}) \tag{3.2}$$

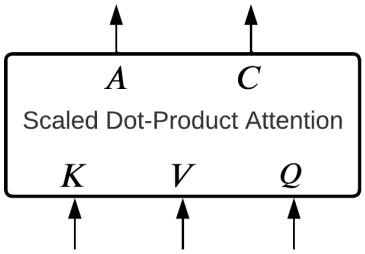
$$C = attn(Q, K, V) = AV \tag{3.3}$$

trong đó $X \in \mathbb{R}^{n \times d'}$ là ma trận chứa đầu vào đã encode, $\mathbf{W}_i \in \mathbb{R}^{d' \times d}$ là các ma trận trọng số, $A \in \mathbb{R}^{n \times n}$ là ma trận attention distribution, $C \in \mathbb{R}^{n \times d}$ là ma trận context.



Hình 3.2: Scaled Dot-Product Attention (Vaswani et al., 2017)

Vì trọng số của mỗi từ được tính bằng phép nhân vô hướng và điều



Hình 3.3: Scaled Dot-Product Attention dưới góc nhìn là một mô hình tính toán

chỉnh bằng hệ số $\frac{1}{\sqrt{d}}$ nên phương pháp attention do Vaswani và cộng sự đề xuất có tên gọi *scaled dot-product attention*. Dưới góc nhìn của mạng nơ-ron nhân tạo, scaled dot-product attention được biểu diễn như Hình 3.2. Trong tài liệu này, scaled dot-product attention còn được biểu diễn dưới dạng một mô hình tính toán như Hình 3.3 gồm đầu vào là 3 ma trận *query* (Q), *key* (K), *value* (V); đầu ra gồm ma trận *attention distribution* (A) và *context* (C).

Ví dụ 3.1 Cho đầu vào của scaled dot-product attention gồm các ma trận:

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

$$\mathbf{W}_Q = \begin{bmatrix} 0.6 & 0.1 & 0.2 \\ 0.4 & 0.3 & 0.7 \\ 0.1 & 0.4 & 0.2 \\ 0.8 & 0.4 & 0.9 \end{bmatrix},$$

$$\mathbf{W}_K = \begin{bmatrix} 0.1 & 0.5 & 0.3 \\ 0.5 & 0.4 & 0.5 \\ 0.6 & 0.2 & 0.1 \\ 0.7 & 0.9 & 0.8 \end{bmatrix},$$

$$\mathbf{W}_V = \begin{bmatrix} 0.8 & 0.9 & 0.2 \\ 0.3 & 0.7 & 0.5 \\ 0.1 & 0.6 & 0.3 \\ 0.2 & 0.4 & 0.1 \end{bmatrix},$$

X biểu diễn ma trận của ngõ nhập bao gồm 4 từ, mỗi từ được biểu diễn trên một hàng. Trong ví dụ này, chúng ta giả sử mỗi từ được embed thành một véc-tơ 4 chiều. Từ mỗi từ đầu vào trên X, chúng ta sẽ tính các véc-tơ *query*, *key* và *value* tương ứng như sau.

$$Q = X\mathbf{W}_Q = \begin{bmatrix} 0.1 & 0.7 & 0.4 & 0.9 \\ 0.3 & 0.8 & 0.6 & 0.1 \\ 0.4 & 0.2 & 0.5 & 0.2 \\ 0.6 & 0.2 & 0.8 & 0.7 \end{bmatrix} \begin{bmatrix} 0.6 & 0.1 & 0.2 \\ 0.4 & 0.3 & 0.7 \\ 0.1 & 0.4 & 0.2 \\ 0.8 & 0.4 & 0.9 \end{bmatrix} = \begin{bmatrix} 1.10 & 0.74 & 1.40 \\ 0.64 & 0.55 & 0.83 \\ 0.53 & 0.38 & 0.50 \\ 1.08 & 0.72 & 1.05 \end{bmatrix}$$

$$K = X\mathbf{W}_K = \begin{bmatrix} 0.1 & 0.7 & 0.4 & 0.9 \\ 0.3 & 0.8 & 0.6 & 0.1 \\ 0.4 & 0.2 & 0.5 & 0.2 \\ 0.6 & 0.2 & 0.8 & 0.7 \end{bmatrix} \begin{bmatrix} 0.1 & 0.5 & 0.3 \\ 0.5 & 0.4 & 0.5 \\ 0.6 & 0.2 & 0.1 \\ 0.7 & 0.9 & 0.8 \end{bmatrix} = \begin{bmatrix} 1.23 & 1.22 & 1.14 \\ 0.86 & 0.68 & 0.63 \\ 0.58 & 0.56 & 0.43 \\ 1.13 & 1.17 & 0.92 \end{bmatrix}$$

$$V = X\mathbf{W}_V = \begin{bmatrix} 0.1 & 0.7 & 0.4 & 0.9 \\ 0.3 & 0.8 & 0.6 & 0.1 \\ 0.4 & 0.2 & 0.5 & 0.2 \\ 0.6 & 0.2 & 0.8 & 0.7 \end{bmatrix} \begin{bmatrix} 0.8 & 0.9 & 0.2 \\ 0.3 & 0.7 & 0.5 \\ 0.1 & 0.6 & 0.3 \\ 0.2 & 0.4 & 0.1 \end{bmatrix} = \begin{bmatrix} 0.51 & 1.18 & 0.58 \\ 0.56 & 1.23 & 0.65 \\ 0.47 & 0.88 & 0.35 \\ 0.76 & 1.44 & 0.53 \end{bmatrix}$$

Như vậy, trong ví dụ này, mỗi véc-tơ *query*, *key* và *value* sẽ là một véc-tơ 3 chiều. Ví dụ, véc-tơ query cho từ đầu tiên của X sẽ là hàng đầu tiên của ma trận Q, tức véc-tơ [1.10 0.74 1.40].

Tiếp theo, giá trị đầu ra của attention được tính như sau:

$$QK^\top = \begin{bmatrix} 1.10 & 0.74 & 1.40 \\ 0.64 & 0.55 & 0.83 \\ 0.53 & 0.38 & 0.50 \\ 1.08 & 0.72 & 1.05 \end{bmatrix} \left(\begin{bmatrix} 1.23 & 1.22 & 1.14 \\ 0.86 & 0.68 & 0.63 \\ 0.58 & 0.56 & 0.43 \\ 1.13 & 1.17 & 0.92 \end{bmatrix} \right)^\top = \begin{bmatrix} 3.85 & 2.33 & 1.65 & 3.39 \\ 2.40 & 1.44 & 1.03 & 2.13 \\ 1.68 & 1.02 & 0.73 & 1.50 \\ 3.40 & 2.07 & 1.48 & 3.02 \end{bmatrix}$$

$$\frac{QK^T}{\sqrt{d_k}} = \begin{bmatrix} 3.85 & 2.33 & 1.65 & 3.39 \\ 2.40 & 1.44 & 1.03 & 2.13 \\ 1.68 & 1.02 & 0.73 & 1.50 \\ 3.40 & 2.07 & 1.48 & 3.02 \end{bmatrix} \cdot \frac{1}{\sqrt{3}} = \begin{bmatrix} 2.22 & 1.34 & 0.95 & 1.96 \\ 1.38 & 0.83 & 0.59 & 1.22 \\ 0.97 & 0.59 & 0.42 & 0.86 \\ 1.96 & 1.20 & 0.85 & 1.74 \end{bmatrix}$$

$$\begin{aligned} A &= softmax(\frac{QK^T}{\sqrt{d_k}}) \\ &= softmax \left(\begin{bmatrix} 2.22 & 1.34 & 0.95 & 1.96 \\ 1.38 & 0.83 & 0.59 & 1.22 \\ 0.97 & 0.59 & 0.42 & 0.86 \\ 1.96 & 1.20 & 0.85 & 1.74 \end{bmatrix} \right) \\ &= \begin{bmatrix} 0.41 & 0.17 & 0.11 & 0.31 \\ 0.34 & 0.20 & 0.16 & 0.30 \\ 0.31 & 0.22 & 0.18 & 0.29 \\ 0.38 & 0.18 & 0.13 & 0.31 \end{bmatrix} \end{aligned}$$

Trong ma trận attention A này, mỗi hàng tương ứng với một từ đầu vào. Giá trị trên mỗi cột tương ứng với *độ tương tự của các từ trong câu đối với từ đang xét*. Các giá trị được chuẩn hóa thành các trọng số tương ứng *một phân bố xác suất*. Ví dụ: hàng thứ nhất của ma trận A có giá trị là [0.41 0.17 0.11 0.31]. Các giá trị này biểu thị độ tương tự của từ thứ

nhất với chính nó là 0.41, độ tương tự của từ này đối với từ thứ hai, ba, tư của ngõ nhập có giá trị lần lượt là 0.17, 0.11 và 0.31. Các giá trị trọng số trong ma trận A này có ý nghĩa giống như các giá trị *attention weight* đã thảo luận ở chương trước, tuy nhiên các giá trị này có thể tính **đồng thời cùng lúc cho tất cả các từ trong câu**.

Tiếp theo, ma trận C (context) sẽ được tính như sau:

$$
\begin{aligned}
C &= AV \\
&= \begin{bmatrix} 0.41 & 0.17 & 0.11 & 0.31 \\ 0.34 & 0.20 & 0.16 & 0.30 \\ 0.31 & 0.22 & 0.18 & 0.29 \\ 0.38 & 0.18 & 0.13 & 0.31 \end{bmatrix} \begin{bmatrix} 0.51 & 1.18 & 0.58 \\ 0.56 & 1.23 & 0.65 \\ 0.47 & 0.88 & 0.35 \\ 0.76 & 1.44 & 0.53 \end{bmatrix} \\
&= \begin{bmatrix} 0.59 & 1.24 & 0.55 \\ 0.59 & 1.22 & 0.54 \\ 0.58 & 1.21 & 0.54 \\ 0.59 & 1.23 & 0.55 \end{bmatrix}
\end{aligned}
$$

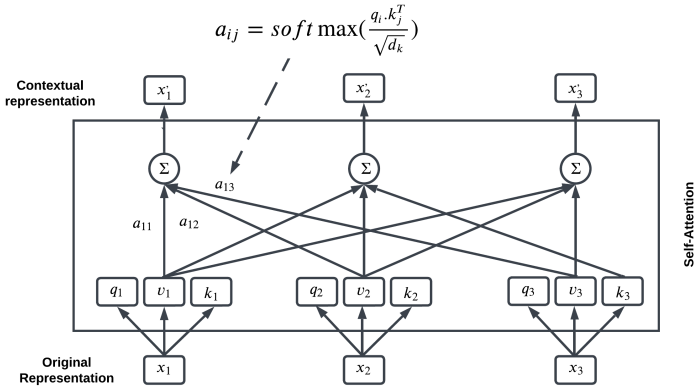
Ma trận này có thể hiểu như kết quả từ việc *học biểu diễn* (representation learning) của các từ đầu vào. Cụ thể, từ thứ nhất sẽ được biểu diễn thành một véc-tơ mới là $[0.59 \quad 1.24 \quad 0.55]$. Thông tin biểu diễn này sẽ chứa đựng thông tin của dữ liệu được tiền huấn luyện thể hiện qua các trọng số của các ma trận $\mathbf{W}_Q$,$\mathbf{W}_K$ và $\mathbf{W}_V$ vừa chứa đựng thông tin ngữ cảnh trong câu nhập.

Trong mô hình Transformer, cơ chế này sẽ được sử dụng với 2 chức năng:

- *Self-attention*: Nhúng thông tin từ các từ xung quanh (context) vào véc-tơ biểu diễn từ. Chức năng này thay cho cơ chế lan truyền dữ liệu giữa các nút mạng RNN.
- *Cross-attention* (Encoder-Decoder attention): Tập trung vào một số từ, cụm từ xung quanh có liên hệ gần gũi khi dịch một từ.

Hình 3.4 mô tả chi tiết cách sử dụng scaled dot-product attention như

một khối self-attention.



Hình 3.4: Self-attention

Trong đó:

- Các véc-tơ x_i là các *véc-tơ embedding* của từ thứ i trong câu.
- Các véc-tơ q_i, k_i, v_i lần lượt là query, key, value tương ứng với từ thứ i. Các véc-tơ này được xác định bằng cách cho x_i đi qua một lớp *linear* và có thể xem là các cách embedding khác nhau cho cùng một từ.
- Các giá trị *score* được tính bằng cách nhân vô hướng query của một từ với key của các từ còn lại. Giá trị score này thể hiện khoảng cách ngữ nghĩa của các từ: các từ có ngữ nghĩa liên hệ gần gũi với từ đang xét thì có score lớn, ngược lại các từ không có liên hệ ngữ nghĩa sẽ có score nhỏ hơn. Các score sau đó được nhân với hệ số $\frac{1}{\sqrt{d_k}}$ và qua hàm softmax để chuẩn hóa về miền giá trị $[0, 1]$.
- Các véc-tơ $x_i^{'}$ được tính bằng tổng có trọng số của các véc-tơ value và được xem như là các lớp embedding mới cho các từ. Bằng cách này thông tin ngữ nghĩa của các từ xung quanh được nhúng vào lớp embedding mới một cách trực tiếp mà không bị "mất mát" do quá trình lan truyền với khoảng cách xa như trong mô hình RNN.

Khối self-attention này được dùng trong cả encoder và decoder của mô hình Transformer. Tuy nhiên, trong decoder, khối attention còn có thêm một lớp *mask* như Hình 3.2. Lớp mask này gán giá trị $-\infty$ cho các từ nằm bên phải từ đang xét để sau khi qua softmax các từ này có trọng số bằng 0. Bằng cách này trong quá trình huấn luyện, mô hình không dùng thông tin ngữ cảnh của các từ trong tương lai giống như lúc dự đoán.

Khi dùng scaled dot-product attention vào mục đích cross attention, các véc-tơ query được xây dựng từ các véc-tơ embedding của decoder còn key, value được xây dựng từ các véc-tơ embedding của encoder. Bằng cách này, mô hình có thể tập trung vào một số từ, cụm từ xung quanh có liên hệ gần gũi khi dịch một từ tương tự như cách áp dụng *addtive attention* (Bahdanau et al., 2015b) của Bahdanau và cộng sự trên mô hình RNN.

Thay vì sử dụng một lớp attention với query, key, value với kích thước $d_k = d_v = d_{model}$, Vaswani và cộng sự sử dụng song song $h = 8$ bộ ba query, key và value với kích thước $d_k = d_v = \frac{d_{model}}{h}$ tạo ra h *attention head* (Hình 3.5) với kích thước d_v.

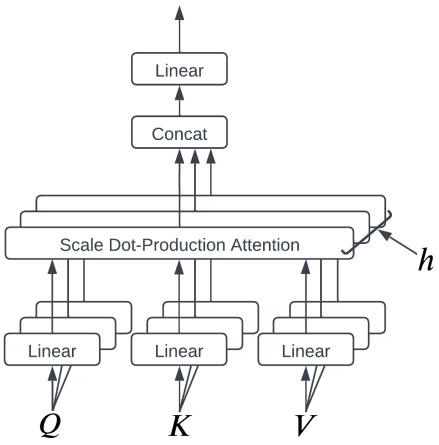
Mỗi head trong Hình 3.5 tương ứng với một góc nhìn khác nhau về ngữ nghĩa của từ, giúp cho mô hình lấy thông tin ngữ cảnh từ nhiều từ khác nhau mà không chỉ tập trung vào một từ nhất định. Sau cùng, các head được nối với nhau tạo thành kết quả output như Công thức 3.4:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{3.4}$$

với $head_i$ được bằng Công thức 3.5:

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{3.5}$$

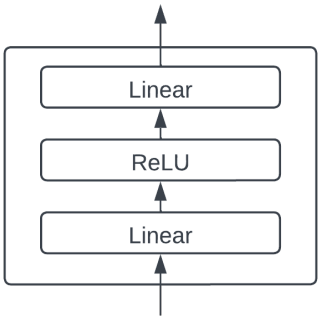
trong đó $W^O \in R^{hd_v \times d_{model}}$, $W_i^Q \in R^{d_{model} \times d_k}$, $W_i^K \in R^{d_{model} \times d_k}$, $W_i^V \in R^{d_{model} \times d_v}$ là các trọng số cần học. Do giảm kích thước mỗi head ($d_v = \frac{d_{model}}{h}$) nên tổng chi phí tính toán vẫn tương tự so với dùng một head với kích thước $d_v = d_{model}$.



Hình 3.5: Multi-head Attention (Vaswani et al., 2017)

3.2.3 Khối Feed Forward

Nếu như cơ chế attention giúp thay thế quá trình lan truyền trạng thái ẩn giữa các nút RNN trong việc nhúng thông tin ngữ nghĩa vào véc-tơ biểu diễn từ thì khối feed forward (Hình 3.6) có tác dụng tính toán và rút trích thông tin như hoạt động tính toán bên trong mỗi nút RNN.



Hình 3.6: Feed Forward Network

Quá trình tính toán của khối feed forward trong Hình 3.6 được biểu diễn bằng Công thức 3.6:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{3.6}$$

với $W_1 \in R^{d_{model} \times d_{ff}}$, $b_1 \in R^{n \times 1}$, $W_2 \in R^{d_{ff} \times d_{model}}$, $b_2 \in R^{n \times 1}$ là các trọng số cần học và n là số từ trong chuỗi, $d_{ff} = 2048$ là kích thước trạng thái ẩn sau khi qua lớp linear đầu tiên.

Ví dụ 3.2 Cho khối Feed Forward trong mô hình Transformer gồm:

$$W_1 = \begin{bmatrix} 0.6 & 0.1 & 0.2 & 0.7 \\ 0.4 & 0.3 & 0.7 & 0.3 \\ 0.8 & 0.4 & 0.9 & 0.4 \end{bmatrix},$$

$$W_2 = \begin{bmatrix} 0.1 & 0.3 & 0.5 \\ 0.5 & 0.5 & 0.6 \\ 0.6 & 0.1 & 0.8 \\ 0.7 & 0.8 & 0.1 \end{bmatrix},$$

$$b_1 = 0.3$$

$$b_1 = 0.2$$

Với đầu vào là ma trận

$$x = \begin{bmatrix} 0.5 & 0.7 & 0.2 \\ 0.4 & 0.2 & 0.1 \\ 0.9 & 0.4 & 0.8 \end{bmatrix}$$

đầu ra của khối Feed Forward được tính như sau:

$$\begin{aligned} l_1 &= xW_1 + b_1 \\ &= \begin{bmatrix} 0.5 & 0.7 & 0.2 \\ 0.4 & 0.2 & 0.1 \\ 0.9 & 0.4 & 0.8 \end{bmatrix} . \begin{bmatrix} 0.6 & 0.1 & 0.2 & 0.7 \\ 0.4 & 0.3 & 0.7 & 0.3 \\ 0.8 & 0.4 & 0.9 & 0.4 \end{bmatrix} + 0.3 \\ &= \begin{bmatrix} 1.0400 & 0.6400 & 1.0700 & 0.9400 \\ 0.7000 & 0.4400 & 0.6100 & 0.6800 \\ 1.6400 & 0.8300 & 1.4800 & 1.3700 \end{bmatrix} \end{aligned}$$

$$r = max(0, l_1) = \begin{bmatrix} 1.0400 & 0.6400 & 1.0700 & 0.9400 \\ 0.7000 & 0.4400 & 0.6100 & 0.6800 \\ 1.6400 & 0.8300 & 1.4800 & 1.3700 \end{bmatrix}$$

$$
\begin{aligned}
FFN(x) &= rW_2 + b_2 \\
&= \begin{bmatrix} 1.0400 & 0.6400 & 1.0700 & 0.9400 \\ 0.7000 & 0.4400 & 0.6100 & 0.6800 \\ 1.6400 & 0.8300 & 1.4800 & 1.3700 \end{bmatrix} . \begin{bmatrix} 0.1 & 0.3 & 0.5 \\ 0.5 & 0.5 & 0.6 \\ 0.6 & 0.1 & 0.8 \\ 0.7 & 0.8 & 0.1 \end{bmatrix} + 0.2 \\
&= \begin{bmatrix} 1.9240 & 1.6910 & 2.0540 \\ 1.3320 & 1.2350 & 1.3700 \\ 2.6260 & 2.3510 & 2.8390 \end{bmatrix}
\end{aligned}
$$

3.2.4 Khối Positional Encoding

Do mô hình Transformer không có kiến trúc tuần tự như mô hình RNN nên cơ chế scaled dot-product attention chưa biểu diễn được thông tin về vị trí của từ khi tạo ra véc-tơ context của từ đó. Nói cách khác, với hai câu "Bạn giỏi hơn tôi" và "Tôi giỏi hơn bạn", các véc-tơ context của các chữ "Bạn", "giỏi", "hơn", "tôi" sẽ giống nhau cho cả hai câu mặc dù về ngữ nghĩa hai câu này có thể xem là ngược hẳn nhau.

Để giải quyết vấn đề trên, Vaswani và cộng sự đã thêm lớp *positional encoding* vào véc-tơ embedding của các từ trước khi đưa vào encoder và decoder. Lớp positional encoding là một ma trận cùng kích thước với ma trận embedding và được hiện thực bằng các hàm *sine* và *cosine* trong Công thức 3.7 và 3.8:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{3.7}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{3.8}$$

với *pos* là vị trí của từ trong câu ứng với hàng thứ *pos* trong ma trận positional encoding, i là chiều thứ i trong ma trận positional embedding. Ma trận positional encoding sau đó được cộng vào ma trận embedding nhân với trọng số $\sqrt{d_{model}}$ để nhúng thêm thông tin về thứ tự.

Ví dụ 3.3 Cho $d_{model} = 512$, và $x_{pos=1} = [0.2, 0.3, 0.6, 0.1]$ và $x_{pos=2} = [0.5, 0.1, 0.7, 0.4]$. Hãy tính véc-tơ của $x_{pos=1}$ và $x_{pos=2}$ sau khi có positional encoding.

Ta có

$$\begin{aligned}
PE_{(1,0)} &= sin(1/10000^{0/512}) = 0.8415 \\
PE_{(1,1)} &= cos(1/10000^{0/512}) = 0.5403 \\
PE_{(1,2)} &= sin(1/10000^{2/512}) = 0.8219 \\
PE_{(1,3)} &= cos(1/10000^{2/512}) = 0.5697
\end{aligned}$$

Suy ra

$$\begin{aligned}
x_{pos=1} &= [0.2, 0.3, 0.6, 0.1] + [0.8415, 0.5403, 0.8219, 0.5697] \\
&= [1.0415, 0.8403, 1.4219, 0.6697]
\end{aligned}$$

Ta lại có

$$\begin{aligned}
PE_{(2,0)} &= sin(2/10000^{0/512}) = 0.9093 \\
PE_{(2,1)} &= cos(2/10000^{0/512}) = -0.4161 \\
PE_{(2,2)} &= sin(2/10000^{2/512}) = 0.9364 \\
PE_{(2,3)} &= cos(2/10000^{2/512}) = -0.3509
\end{aligned}$$

Suy ra

$$\begin{aligned}
x_{pos=2} &= [0.5, 0.1, 0.7, 0.4] + [0.9093, -0.4161, 0.9364, -0.3509] \\
&= [1.4093, -0.3161, 1.6364, 0.0491]
\end{aligned}$$

3.2.5 Nhận xét

Chúng ta có thể xem cơ chế attention được mô tả trong Chương 2 là một trường hợp đơn giản của kỹ thuật scaled dot-product attention. Trong

trường hợp này, các hidden state của Decoder sẽ đóng vai trò query, đồng thời các hidden state của Encoder sẽ vừa đóng vai trò key vừa đóng vai trò value.

Việc tách riêng vai trò của key và value giúp Transformer xử lý thông tin chặt chẽ hơn. Đồng thời, như chúng ta đã thấy, các véc-tơ query, key và value của kiến trúc Transformer có thể được tạo ra cùng lúc trong cả hai cơ chế self-attention và cross-attention. Điều này cho phép Transformer có thể song song hóa khi hạ tầng tính toán cho phép và cùng lúc tổng hợp thông tin attention và context từ cả hai chiều của chuỗi đang xử lý từ vị trí tính toán.

Tóm lại, từ kiến trúc và những tính chất nêu trên, mô hình Transformer hoàn toàn đảm bảo được những tính chất mà mô hình RNN có đồng thời giải quyết một số nhược điểm của mô hình RNN:

- Thông tin ngữ cảnh nhúng vào véc-tơ biểu diễn một từ được lấy trực tiếp từ các từ xung quanh bất kể nằm bên trái hay bên phải từ đang xét mà không cần phải qua quá trình truyền trạng thái ẩn trong đoạn đường dài.
- Loại bỏ kiến trúc tuần tự và tận dụng được khả năng tính toán song song của GPU giúp tăng tốc độ xử lý đặc biệt trong quá trình huấn luyện.

Với những tính chất này, mô hình Transformer đã đạt được nhiều kết quả tốt hơn so với mô hình RNN và dần thay thế mô hình RNN trong một số bài toán NLP nói chung và bài toán NMT nói riêng.

3.3 Tiền huấn luyện và các mô hình ngôn ngữ lớn dựa trên Transformer

Tiền huấn luyện (pre-train) (Han et al., 2021) là một hướng nghiên cứu đã đạt được nhiều thành công trong những năm gần đây nhờ vào khối lượng dữ liệu (*data*) lớn và sự phát triển các mô hình học sâu. Ý tưởng

của phương pháp này là huấn luyện trước một kiến trúc mạng học sâu với một tập dữ liệu có sẵn. Mạng học sâu này, đặc biệt là bộ trọng số thu được, sẽ tiếp tục được dùng để tiếp tục *tinh chỉnh* (fine-tune) hoặc *học chuyển đổi* (transfer learning) cho các ứng dụng và bài toán khác.

Kỹ thuật này thường được dùng để tiền huấn luyện các mô hình ngôn ngữ để tiếp tục xử lý trong các ứng dụng *downstream* tiếp theo trong lĩnh vực xử lý ngôn ngữ tự nhiên. Khi huấn luyện một mô hình ngôn ngữ với một kiến trúc mạng học sâu lớn và với một kho ngữ liệu lớn thì chúng ta có được một *mô hình ngôn ngữ lớn* (large language model).

Kể từ sự ra đời của mô hình Transformer, nhiều mô hình ngôn ngữ lớn được tiền huấn luyện trên Transformer. Tiêu biểu có thể kể đến *BERT* (Devlin et al., 2019) dùng kiến trúc encoder của Transformer, *GPT-2* (Radford et al., 2019) dùng kiến trúc decoder của Transformer, *Transformer XL* (Dai et al., 2019) dùng kiến trúc recurrent decoder dựa trên decoder của Transformer. Việc vận dụng các mô hình ngôn ngữ lớn này vào các ứng dụng downstream đã cho những kết quả ấn tượng trong nhiều bài toán NLP.

Các mô hình ngôn ngữ lớn dựa trên Transformer có thể chia thành hai nhóm chính: *AutoEncoding* và *AutoRegression*. Mô hình ngôn ngữ lớn theo hướng AutoEncoding sẽ "encode" (hoặc "embed") dữ liệu đầu vào để tạo thành một dạng biểu diễn chứa đựng nhiều thông tin ngữ nghĩa cho các ứng dụng khác tiếp tục sử dụng. Các ứng dụng theo hướng này thường sử dụng mô hình ngôn ngữ lớn hướng AutoEncoding như một lớp *embedding* tiếp theo (sau khi đã thực hiện embedding "cơ bản" bằng word2vec hoặc Glove) để tiếp tục xử lý. Trong khi đó AutoRegression sẽ hướng đến việc tạo ra một chuỗi xuất từ chuỗi nhập ban đầu. Ý nghĩa của chuỗi xuất này tùy thuộc vào mục đích huấn luyện.

Như vậy sự khác nhau giữa các mô hình theo hướng AutoEncoding và AutoRegression phụ thuộc vào *chiến lược huấn luyện* hơn là kiến trúc mô hình. Hai đại diện nổi tiếng nhất của các mô hình ngôn ngữ lớn theo hai hướng này có thể là BERT và GPT. Phần tiếp theo sẽ giới thiệu chi

tiết hơn hai mô hình này.

3.4 Mô hình BERT

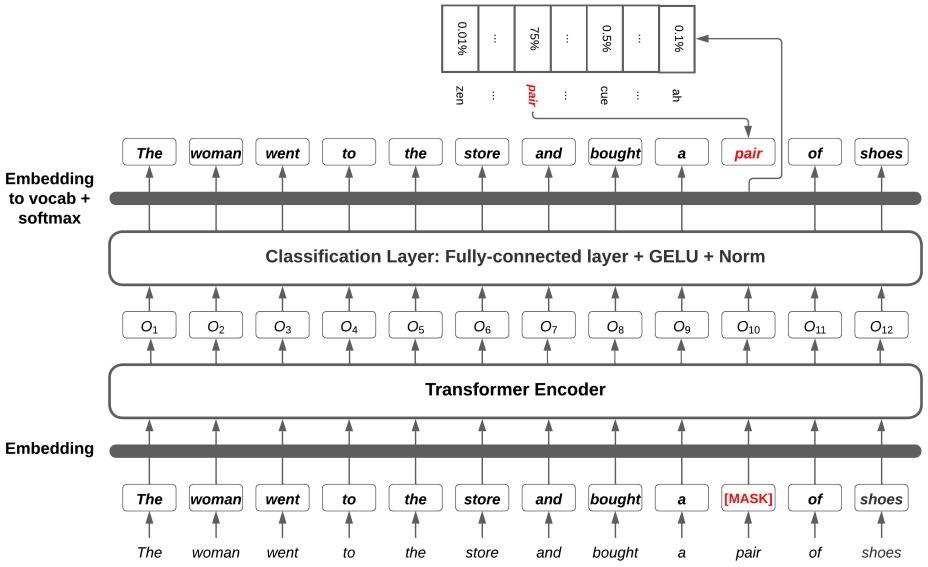
3.4.1 Giới thiệu

Năm 2018, Devlin và cộng sự đề xuất mô hình *Pre-training of Deep Bidirectional Transformers for Language Understanding (BERT)* (Devlin et al., 2019). BERT được huấn luyện trên lượng dữ liệu lớn và có thể xem là một *pre-trained model* thường được *fine-tune* cùng với một số lớp nơ-ron đơn giản để tạo nên những mô hình *state-of-the-art* giải quyết nhiều bài toán khác nhau trong lĩnh vực xử lý ngôn ngữ tự nhiên như *question answering* (Dwivedi & Singh, 2013) hay *language inference* (MacCartney, 2009).

3.4.2 Các chiến lược huấn luyện của BERT

BERT dùng chung một kiến trúc với khối encoder của Transformer, và được huấn luyện trên hai mục tiêu: *masked language modeling (MLM)* và *next sentence prediction (NSP)*

Với mục tiêu MLM, BERT sẽ ngẫu nhiên *che* (mask) 15% token trong chuỗi đầu vào sẽ được *mask* (với 80% trong số đó thực sự bị thay thế bởi token [MASK], 10% số token được thay thế ngẫu nhiên bởi một token khác và 10% còn lại được giữ không đổi), sau đó BERT được huấn luyện để dự đoán những token bị mask dựa vào ngữ cảnh (*context*) của những token xung quanh. Vì vậy đầu ra của BERT có thể được dùng như một lớp embedding đã được nhúng vào thông tin ngữ cảnh từ cả hai hướng nhờ vào cơ chế self-attention trong mỗi lớp Transformer encoder. Hình 3.7 mô tả cách BERT dự đoán và huấn luyện cho mục tiêu này. Sau khi dữ liệu input đi qua các lớp Encoding của Transformer, chúng ta sẽ thu được các véc-tơ embedding sau cùng. Các véc-tơ embedding này tiếp tục đi qua một tầng FC để tạo ra các véc-tơ phân bố xác suất của bộ từ vựng tại vị trí tương ứng trong câu. Đây cũng chính là tiên đoán của BERT



Hình 3.7: Tổng quan quá trình dự đoán và huấn luyện của mô hình BERT

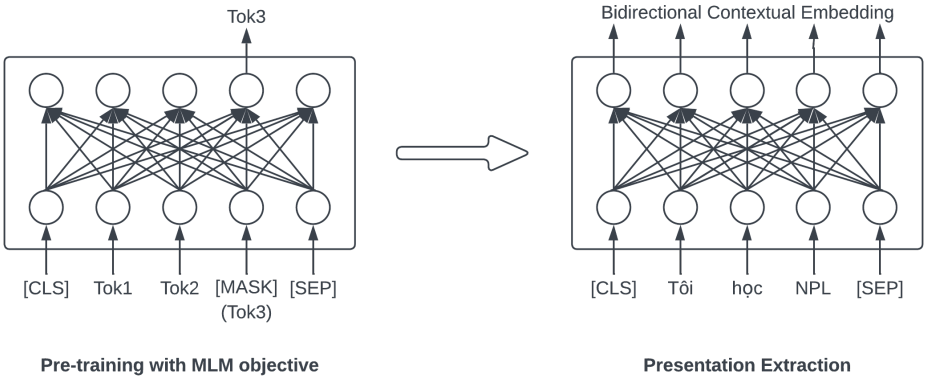
cho xác suất xuất hiện của từng từ tại vị trí đầu ra của mô hình. BERT sẽ so sánh tiên đoán xác suất từ xuất hiện ở vị trí MASK và so sánh với từ tương ứng ở ground truth để xác định giá trị loss cho lần huấn luyện này và cập nhật trọng số tương ứng.

Như vậy, chúng ta có thể hiểu tầng FC sau cùng **chính là decoder của BERT**. Sau khi đã được huấn luyện với một tập corpus tương ứng, tầng FC này sẽ được gỡ ra, phần Encoder còn lại của BERT sẽ **được sử dụng như một tầng embedding** để tạo ra các véc-tơ BERT-embedding cho các tài liệu khác. Đây cũng là cách BERT thường được sử dụng nhiều nhất. Điều này cũng cho chúng ta hai nhận xét quan trọng:

- BERT là một mô hình ngôn ngữ lớn dạng AutoEncoding, thường được sử dụng để tạo ra các dạng biểu diễn cho văn bản để tiếp tục phục vụ cho các bài toán khác.
- Với cách sử dụng này, về kiến thúc thì BERT hoàn toàn giống như tầng Encoding của Transformer. Tuy nhiên, như đã trình bày, sự

khác nhau của các mô hình ngôn ngữ k**hác nhau ở cách huấn luyện**, không nằm ở kiến trúc. Với cách huấn luyện này, BERT không thể dùng cho bài toán NMT của Transformer.

Với mục tiêu NSP, mô hình nhận đầu vào các cặp chuỗi ngăn cách nhau bởi token [SEP] và được huấn luyện để dự đoán đó có phải là hai chuỗi liên tiếp hay không như Hình 3.8. Để huấn luyện, mô hình sẽ chọn một số cặp câu liên tiếp nhau và một số cặp câu ngẫu nhiên với giá trị ground truth được quy định bởi token [CLS]. Bằng cách thay đổi ý nghĩa của token [CLS] với các input tương ứng, chúng ta có thể huấn luyện BERT cho các bài toán *supervised learning* khác. Hai mục tiêu MLM và NSP có thể được huấn luyện đồng thời khi huấn luyện BERT với một tập corpus.



Hình 3.8: Nhờ vào mục tiêu MLM và kiến trúc encoder của Transformer, BERT có thể được dùng như một khối embedding để trích xuất thông tin ngữ cảnh từ hai phía

Ví dụ 3.4 Giả sử bộ vocabulary của BERT gồm các token trong Bảng 3.1. Cho đầu vào của BERT là chuỗi token: ([CLS], tôi, [MASK], thích, học, NLP, [SEP]). BERT sẽ được huấn luyện để tiên đoán từ thích hợp tại vị trí tương ứng với token [MASK].

Index	**Token**
0	[CLS]
1	[SEP]
2	[MASK]
3	không
4	rất
5	tôi
6	NLP
7	học
8	thích

Bảng 3.1: Tập từ vựng

Giả sử đầu ra của BERT như sau:

$$output = \begin{bmatrix} 0.1418 & 0.0594 & 0.0761 & 0.1139 & 0.0825 & 0.1152 & 0.1199 & 0.1451 & 0.1460 \\ 0.0992 & 0.0802 & 0.0727 & 0.1360 & 0.0797 & 0.1476 & 0.1049 & 0.1416 & 0.1380 \\ 0.0877 & 0.0906 & 0.1204 & 0.1092 & 0.1474 & 0.1180 & 0.1345 & 0.1131 & 0.0792 \\ 0.0725 & 0.1376 & 0.1473 & 0.0820 & 0.0870 & 0.0907 & 0.1076 & 0.1269 & 0.1484 \\ 0.1359 & 0.1015 & 0.0960 & 0.0762 & 0.0756 & 0.1047 & 0.1263 & 0.1623 & 0.1216 \\ 0.0649 & 0.1262 & 0.1355 & 0.0773 & 0.1076 & 0.0746 & 0.1618 & 0.1181 & 0.1341 \\ 0.0922 & 0.1624 & 0.0733 & 0.1618 & 0.1026 & 0.0682 & 0.1118 & 0.1077 & 0.1200 \end{bmatrix}$$

Như đã trình bày, ma trận này thể hiện dự đoán của BERT về xác suất xuất hiện của các từ trong tập từ vựng ứng với từng vị trí của chuỗi nhập. Mỗi hàng của ma trận ứng với vị trí của một từ trong chuỗi nhập. Mỗi cột của ma trận ứng với một từ trong tập từ vựng. Ví dụ ở hàng thứ ba (ứng với vị trí của token [MASK] trong ngõ nhập), chúng ta có được véc-tơ sau:

$$\begin{bmatrix} 0.0877 & 0.0906 & 0.1204 & 0.1092 & 0.1474 & 0.1180 & 0.1345 & 0.1131 & 0.0792 \end{bmatrix}$$

Như vậy, xác suất xuất hiện của các từ trong tập từ vựng ứng với vị trí

này sẽ được thể hiện trong Bảng 3.2. Dựa vào bảng này, chúng ta thấy từ có xác suất xuất hiện cao nhất là "rất". Đây cũng chính là dự đoán của mô hình cho từ sẽ xuất hiện ở vị trí [MASK]. Như vậy, từ thông tin ngữ cảnh của các từ: "tôi", "thích", "học", "NLP" trong chuỗi đầu vào, BERT có khả năng dự đoán được từ còn thiếu (token [MASK]) là "rất" để ra được câu hoàn chỉnh là: "tôi rất thích học NLP."

Index	**Token**	**Xác suất**
0	[CLS]:	0.0877
1	[SEP]:	0.0906
2	[MASK]:	0.1204
3	không:	0.1092
4	rất:	0.1474
5	tôi:	0.1180
6	NLP:	0.1345
7	học:	0.1131
8	thích:	0.0792

Bảng 3.2: Xác suất xuất hiện của các từ trong tập từ vựng

Tương tự như nhận xét về lớp embedding của BERT, chúng ta có thể nhận xét về các xác suất tiên đoán như sau:

- Các xác suất này sẽ được tạo ra dựa vào (1) ngữ cảnh của tập corpus đã được tiền huấn luyện trước đó của BERT, được thể hiện qua bộ giá trị các parameters của BERT; và (2) thông tin của chuỗi nhập hiện hành, sẽ được tính bằng kỹ thuật self-attention.
- Nhìn chung, việc dự đoán cho token [MASK] sẽ có đủ thông tin ngữ cảnh, trong khi việc dự đoán cho các vị trí còn lại sẽ bị ảnh hưởng bởi việc ngữ cảnh của chúng có một phần bị che. Do vậy, hàm loss thông thường chỉ được tính tại vị trí ứng với token [MASK] để phục vụ cho việc dự đoán với thông tin hai chiều.

3.4.3 Một số phiên bản khác của BERT

Năm 2019, Liu và cộng sự đưa ra mô hình *RoBERTa (Robustly Optimized BERT Pretraining Approach)* (Y. Liu et al., 2019) áp dụng cùng kiến trúc với BERT với một số thay đổi trong chiến lược huấn luyện: (i) thời gian huấn luyện lâu hơn với nhiều dữ liệu và *batch* lớn hơn; (ii) bỏ mục tiêu NSP; (iii) huấn luyện với chuỗi dài hơn; và (iv) áp dụng *dynamic masking pattern*. RoBERTa cũng đạt được nhiều thành tựu trên GLUE, RACE và SQuAD. Nhìn chung RoBERTa vẫn tuân theo kiến trúc encoder của Transformer và có thể sử dụng với các mục đích tương tự BERT trong mục đích tạo ra lớp embedding cho ngõ nhập.

Năm 2020, Dat Quoc Nguyen và cộng sự đưa ra PhoBERT (D. Q. Nguyen & Nguyen, 2020), một phiên bản khác của RoBERTa được huấn luyện trên dữ liệu tiếng Việt và đạt được những kết quả *state-of-the-art* trên một số bài toán NLP như *part-of-speech tagging*, *dependency parsing*, *named-entity recognition* và *natural language inference* cho tiếng Việt.

3.5 Mô hình GPT

3.5.1 Giới thiệu

Mô hình *Generative Pre-training Transformer* (GPT) được phát triển bởi OpenAI ("OpenAI", 2023) và là một trong những mô hình học sâu nổi tiếng nhất trong lĩnh vực NLP hiện nay. Nếu như BERT được phát triển dựa trên kiến trúc Encoder của Transformer thì GPT được phát triển dựa trên kiến trúc Decoder của Transformer. Mô hình này được huấn luyện trên các tập dữ liệu lớn như Wikipedia và các tài liệu trên Internet, cho phép nó hiểu được cấu trúc ngữ pháp, ngữ nghĩa và văn phong của ngôn ngữ tự nhiên. Mô hình GPT có thể được sử dụng cho nhiều ứng dụng khác nhau, bao gồm tự động phát sinh văn bản, tạo tiêu đề cho bài báo, phân loại văn bản, dịch máy và nhiều ứng dụng NLP khác. Nó đã đạt được kết quả ấn tượng trong nhiều cuộc thi NLP và là một trong những công cụ hữu ích để giải quyết các vấn đề liên quan đến

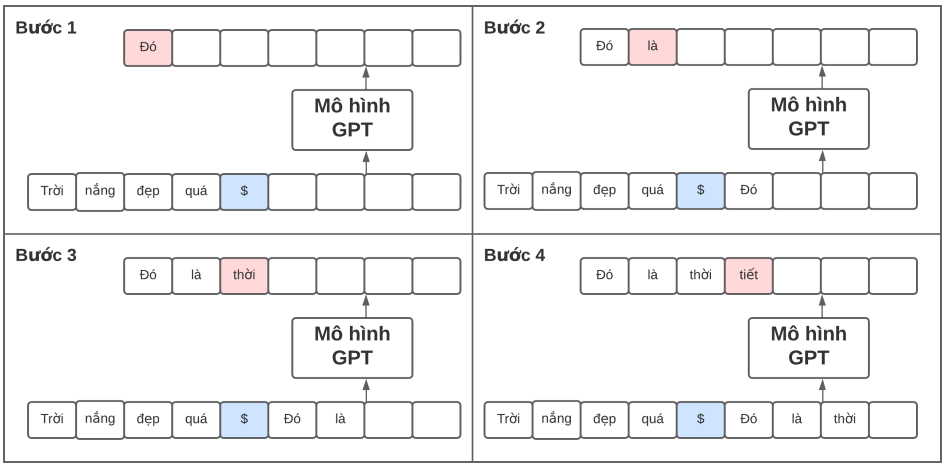
ngôn ngữ tự nhiên.

Dựa trên kiến trúc Decoder của Transformer như đã trình bày, một trong những đặc điểm quan trọng của GPT là cơ chế *prompt-based tuning*. Cơ chế huấn luyện prompt-based của GPT nhằm tìm cách điều chỉnh hoặc tăng cường các dự đoán của mô hình bằng cách cung cấp cho nó các thông tin bổ sung. Các prompt (lời nhắc) là các đoạn văn bản hoặc câu hỏi được đưa vào đầu vào của mô hình để giúp nó tạo ra kết quả đáp ứng cho yêu cầu của người dùng. Một cách để huấn luyện mô hình GPT theo phương pháp prompt-based là sử dụng kỹ thuật *fine-tuning*, trong đó mô hình được tiếp tục huấn luyện trên các tập dữ liệu nhỏ hơn, được cung cấp với các prompt liên quan đến tác vụ mà mô hình cần giải quyết.

Ví dụ, để huấn luyện mô hình GPT để tạo tiêu đề cho một bài báo, các prompt có thể là các đoạn văn bản mô tả nội dung của bài báo. Mô hình được huấn luyện trên các prompt này để học cách tạo ra các tiêu đề tương ứng. Trong một ví dụ khác, các prompt-based GPT cũng có thể được sử dụng để tạo ra các đoạn văn bản mới bằng cách đưa vào các lời nhắc có chứa thông tin hoặc yêu cầu cụ thể. Xét một ví dụ khác, để tạo ra một email yêu cầu hẹn gặp, mô hình GPT có thể được đưa vào prompt có chứa thông tin về người nhận, nội dung và thời gian hẹn gặp. Mô hình sẽ sử dụng các prompt này để tạo ra một email mới, phù hợp với yêu cầu của người dùng.

Cách huấn luyện và hoạt động dựa trên prompt của GPT cũng theo cơ chế sinh chuỗi của mô hình ngôn ngữ, như mô tả trong ví dụ ở Hình 3.9. Dữ liệu nhập của GPT sẽ là một chuỗi prompt. Chuỗi prompt này sẽ được encoded tạo thành một véc-tơ context, đây là trạng thái của decoder sau khi xử lý mô hình ngôn ngữ. Từ trạng thái này, GPT sẽ bắt đầu tạo sinh các từ output theo cơ chế sinh chuỗi của mô hình ngôn ngữ. Như vậy mặc dù chỉ sử dụng phần decoder của Transformer, GPT vẫn hoạt động như một mô hình Seq2Seq trong đó chuỗi nhập là prompt và chuỗi đích là văn bản sinh ra tương ứng. Mô hình GPT sẽ cùng lúc đóng vai trò Encoder và Decoder. Theo cơ chế này, sau khi xử lý chuỗi prompt,

mô hình sẽ tiếp tục tạo sinh chuỗi dựa trên sự tương tự của prompt này với các văn bản mà trước đó nó đã được huấn luyện.

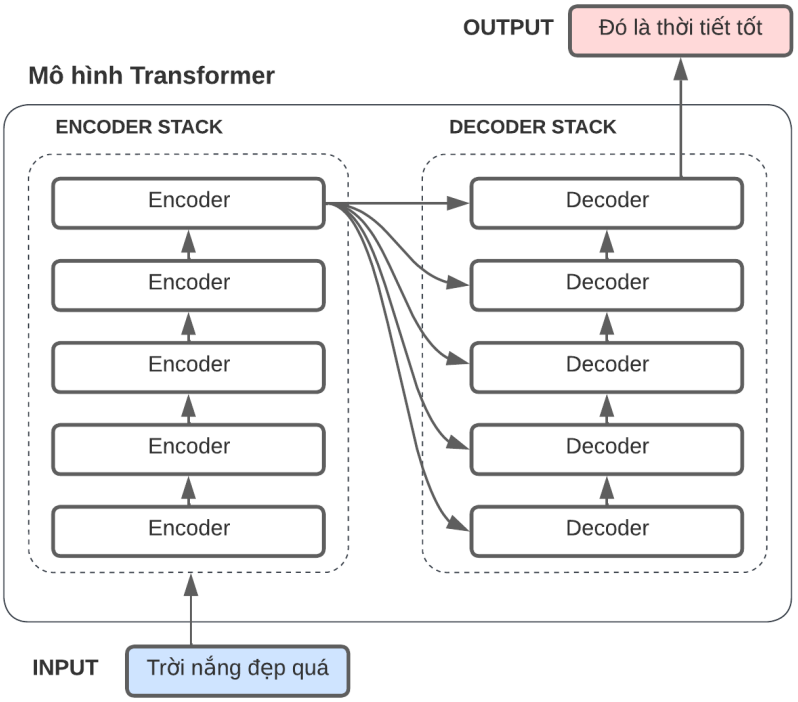


Hình 3.9: Minh họa các bước sinh văn bản của mô hình GPT

3.5.2 Cơ chế hoạt động của mô hình GPT

Để tìm hiểu cơ chế hoạt động của GPT, trước tiên chúng ta cùng nhìn lại tổng quan kiến trúc Transformer ở mức khái quát, như Hình 3.10. GPT sẽ sử dụng lại kiến trúc Decoder của Transformer, như minh họa trong Hình 3.11, trong đó dữ liệu nhập sẽ được trực tiếp đưa vào kiến trúc Decoder này. Việc dự đoán từ tiếp theo bằng khối Transformer-Decoder được minh họa trong Hình 3.12.

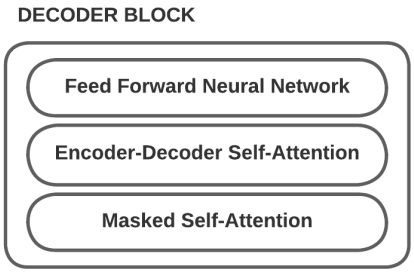
Một đặc điểm đáng chú ý của khối Decoder này là cơ chế *masked self-attention*. Về cơ bản, đây vẫn là kỹ thuật self-attention nhưng có sử dụng cơ chế *che* (masking) tương tự cơ chế MLM của BERT. Trong quá trình huấn luyện phần dữ liệu, khi một token tương ứng được đưa vào ngõ nhập của GPT theo cơ chế non-teacher forcing, phần dữ liệu trong ground truth nằm ở bên phải token này sẽ được che và véc-tơ self-attention sẽ chỉ được tạo ra từ phần bên trái của token này, tức là từ phần dữ liệu đã được tạo sinh (Hình 3.13). GPT dùng cơ chế này để huấn luyện mô hình sinh



Hình 3.10: Tổng quan kiến trúc Encoder

ra dữ liệu mới từ những dữ liệu đã "thấy" được.

Quá trình dự đoán từ tiếp theo của GPT vẫn dựa theo cơ chế "đoán từ" của mô hình ngôn ngữ như Hình 3.14. Sau khi được đi qua các khối Transformer-Decoder trong kiến trúc GPT, chúng ta thu được dữ liệu được encoded của từ nhập hiện hành dưới dạng một véc-tơ. Từ đó, véc-tơ này sẽ được đi qua một ma trận biến đổi để tạo ra xác suất phân bố có thể có cho các từ tiếp theo. GPT sẽ lựa chọn từ tiếp theo bằng cách lấy mẫu theo xác suất. Điều này cũng giải thích vì sao với cùng một prompt, mô hình GPT có thể tạo ra nhiều câu trả lời khác nhau trong những lần gọi khác nhau.



Hình 3.11: Trên GPT, dữ liệu ngõ nhập được xử lý trực tiếp trên khối Decoder

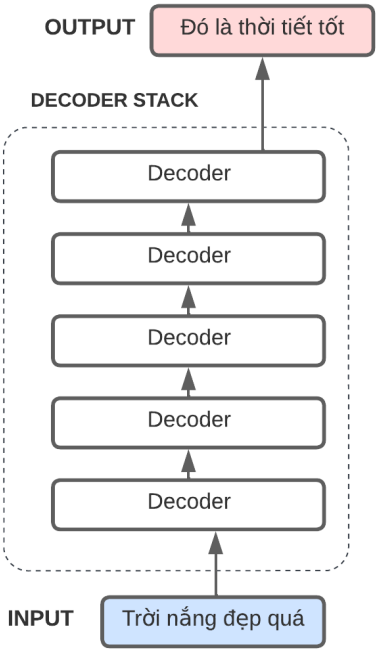
3.5.3 Các phiên bản của mô hình GPT

Từ khi được giới thiệu lần đầu, đã có 3 phiên bản GPT được công bố chính thức, như trình bày trong Bảng 3.3. Các phiên bản của mô hình GPT có cùng kiến trúc nhưng khác nhau về số lượng các lớp và thông số được huấn luyện. Các phiên bản sau cũng được huấn luyện trên các tập dữ liệu lớn hơn nhiều so với phiên bản trước.

Bảng 3.3: Các phiên bản của mô hình GPT

Mô hình	**Số lượng layer**	**Số lượng trọng số**
GPT	12	117 Triệu
GPT-2	48	1 Tỷ 542 Triệu
GPT-3	96	175 Tỷ

Với lượng dữ liệu huấn luyện và số thông số cực lớn, mô hình GPT-3 cũng giới thiệu những khả năng học độc đáo, bao gồm *few-shot learning* (huấn luyện một tác vụ mới chỉ với vài mẫu huấn luyện), *one-shot learning* (huấn luyện một tác vụ mới chỉ với một mẫu huấn luyện) và *zero-shot learning* (thực hiện một tác vụ mới mà không cần huấn luyện). Các Hình 3.15, Hình 3.16 và Hình 3.17 minh họa các khả năng học này của GPT-3.

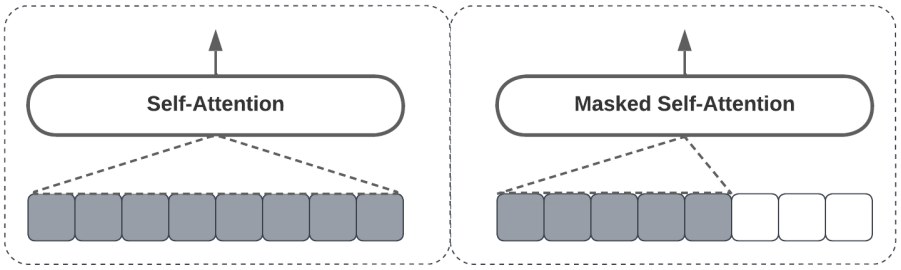


Hình 3.12: Quá trình tổng quát của việc tiên đoán từ tiếp theo trên GPT

Lưu ý rằng trong việc thực hiện các quá trình học như trên, GPT-3 không cập nhật lại các giá trị tham số của mình. Quá trình tạo ra dữ liệu trả lời sẽ dựa trên sự tương tự của các mô tả task và prompt được nhập vào so với kho dữ liệu khổng lồ đã được huấn luyện. Đây cũng chính là kỹ thuật để OpenAI tiếp tục phát triển ứng dụng ChatGPT nổi tiếng với khả năng trả lời rất nhiều dạng câu hỏi (ứng với các task khác nhau) mà không cần huấn luyện lại (tương ứng với zero-shot learning).

3.5.4 Chương trình ChatGPT

ChatGPT là chương trình một *trợ lý ảo* (virtual assistant) dựa trên Trí tuệ Nhân tạo được tạo ra bởi OpenAI và ra mắt công chúng vào tháng 11 năm 2022 ("Introducing ChatGPT", 2023). ChatGPT được fine-tune từ GPT-3.5, một bản nâng cấp của GPT-3. Với cốt lõi là một *mô hình ngôn*

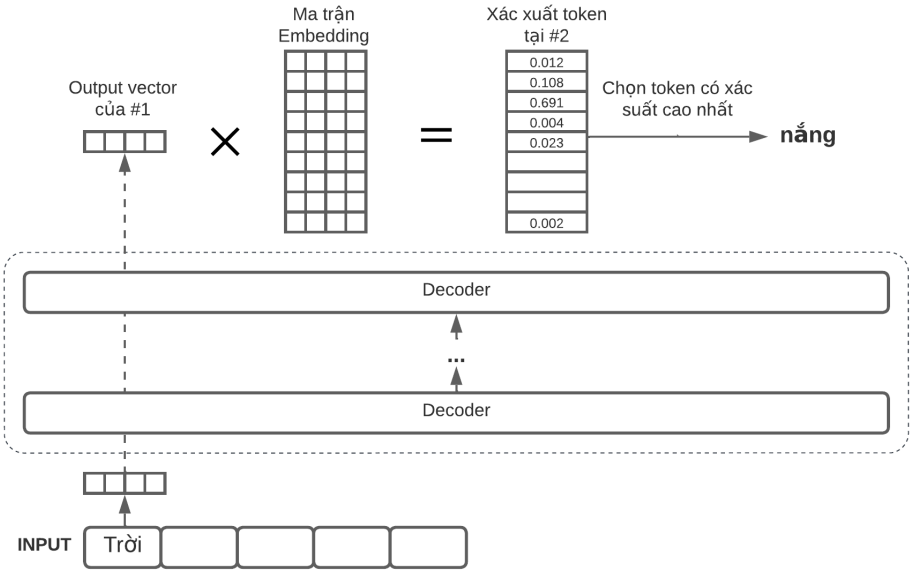


Hình 3.13: So sánh giữa self-attention và masked self-attention

ngữ lớn (large language model) (Manning, 2022), ChatGPT có thể tương tác với người dùng một cách tự nhiên. Như đã trình bày ở trên, dựa vào khả năng của zero-shot learning của GPT-3, chương trình có rất nhiều khả năng: trả lời câu hỏi, giải quyết các vấn đề lập trình, sửa lỗi, viết thơ, tạo bài viết, trả lời email, dịch thuật, v.v. Hình 3.18 minh họa khả năng lập trình các giải thuật cơ bản của ChatGPT. Khả năng zero-shot learning của ChatGPT thậm chí tạo cảm giác như chương trình này được xây dựng dựa trên một máy ảo Linux như Hình 3.19. ChatGPT có thể hiểu nhiều ngôn ngữ, bao gồm cả tiếng Việt và lập kỷ lục có được 1 triệu người dùng chỉ trong vòng 5 ngày!

Bên cạnh việc sử dụng một mô hình nâng cấp của GPT-3, một trong những cải tiến đáng chú ý khác của ChatGPT là sử dụng cơ chế học *tăng cường với phản hồi của con người* (Reinforcement Learning with Human Feedback). Theo như đã trình bày ở trên, hệ thống có thể tạo ra nhiều câu trả lời từ một prompt dựa trên cơ chế hoạt động của mô hình ngôn ngữ. ChatGPT đã sử dụng con người để tối ưu hai quá trình sau:

- Gán nhãn để tạo ra câu trả lời mong đợi từ các prompt có trong tập dữ liệu huấn luyện sẵn có.
- *Xếp hạng* (ranking) các câu trả lời mà hệ thống tạo ra cho một prompt. Dựa vào đó, mô hình sẽ được huấn luyện để tạo ra một cơ chế *phần thưởng* (reward) cho quá trình học tăng cường. Quá trình



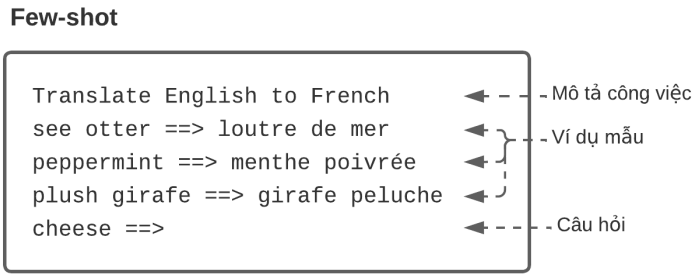
Hình 3.14: Quá trình sinh ra từ tiên đoán của GPT

này cho phép tái huấn luyện nhằm tạo ra câu trả lời chính xác và tự nhiên nhất cho một prompt để từ đó đưa vào sử dụng thực tế.

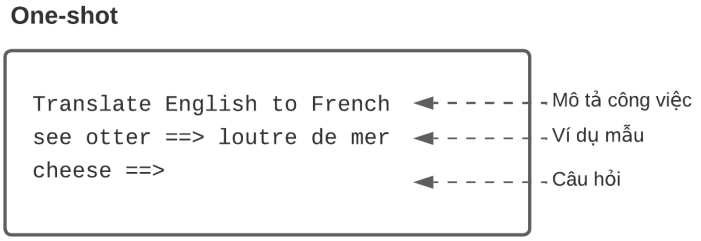
Mặc dù đạt những kết quả đáng kinh ngạc, ChatGPT vẫn có những hạn chế dễ thấy như sau:

- Thông tin được tạo ra có thể thiên vị (bias) và không chính xác dựa vào dữ liệu huấn luyện.
- Do cơ chế tạo câu trả lời dựa trên lấy mẫu xác suất, câu trả lời của ChatGPT có thể không nhất quán và thậm chí không hợp lý, như ví dụ trong Hình 3.20.
- Có kiến thức hạn chế sau năm 2021 vì chỉ được huấn luyện với các tài liệu cho đến thời điểm này, như minh họa trong Hình 3.21.

Nói tóm lại, sau vài tháng ra mắt, các thành tựu và khả năng của ChatGPT đã được cộng đồng thừa nhận như sau:



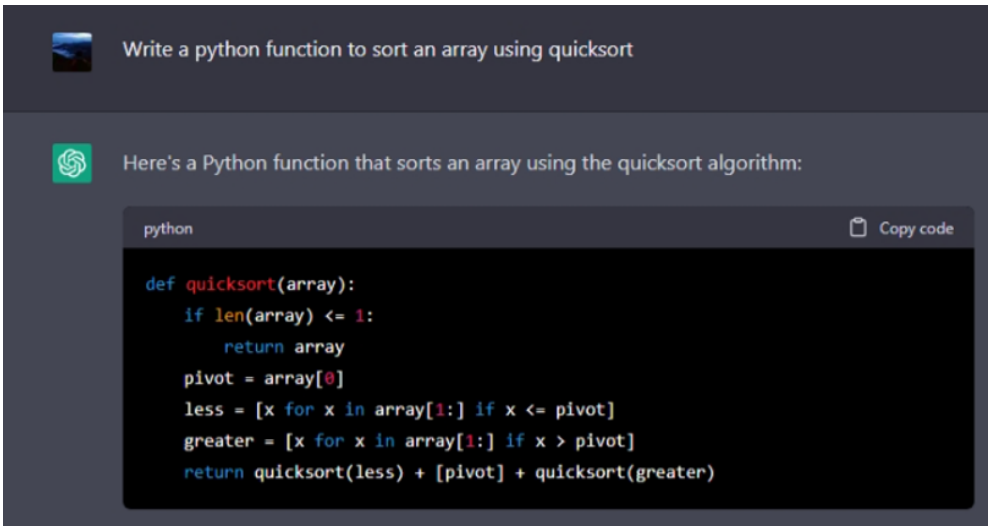
Hình 3.15: Minh họa few-shot learning của GPT-3



Hình 3.16: Minh họa one-shot learning của GPT-3

- ChatGPT rất có ích trong việc đưa ra ý tưởng, sáng tạo nội dung và cung cấp các thông tin liên quan đến kiến thức đại chúng.
- ChatGPT không nên được coi là nguồn thông tin chính xác tuyệt đối. Cần xác minh thông tin bằng nhiều cách khác nhau.
- Trên khía cạnh là một ứng dụng Trí Tuệ Nhân Tạo, ChatGPT nên được sử dụng như một công cụ để giúp con người tăng năng suất, thay vì thay thế họ.

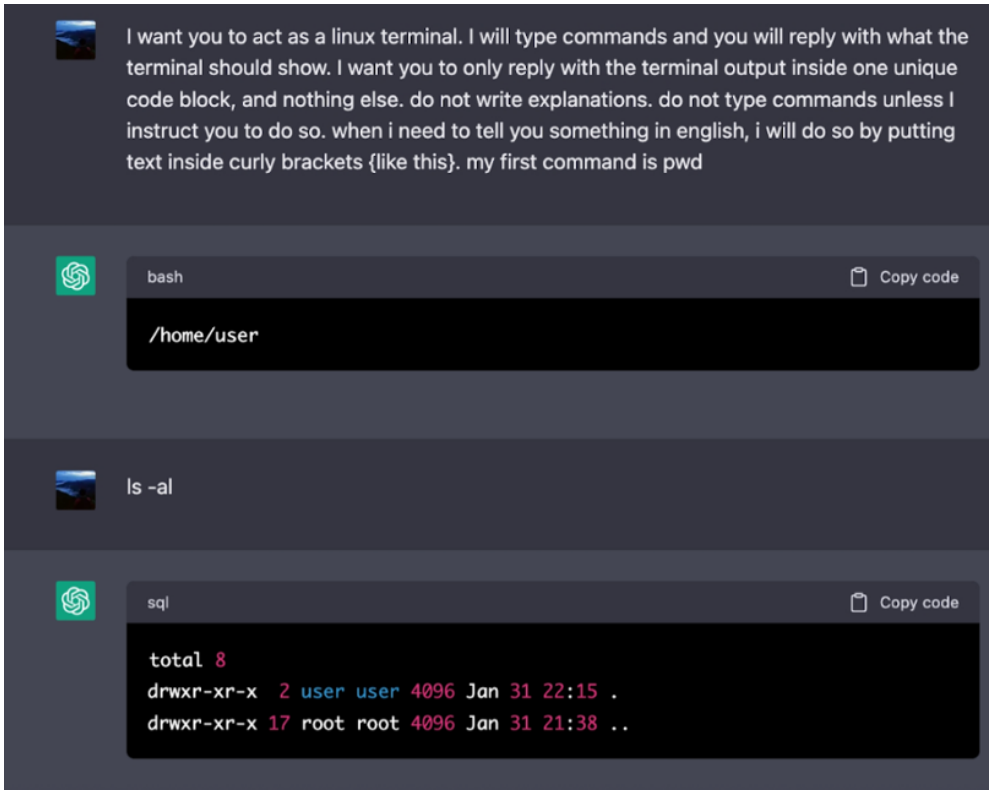
Hình 3.17: Minh họa zero-shot learning của GPT-3



Hình 3.18: ChatGPT tạo lời giải lập trình

3.6 Kết chương

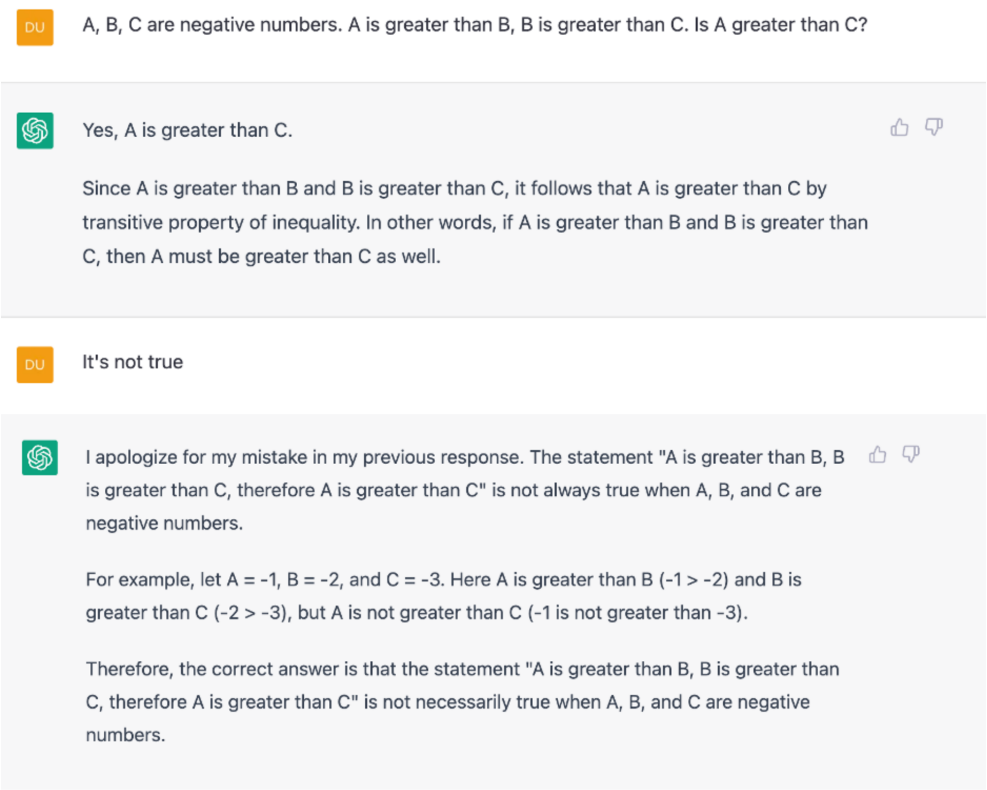
Sau những thành công của các mô hình Seq2Seq dựa trên RNN kết hợp với cơ chế attention, Transformer đã được giới thiệu như một hướng tiếp cận mới cho bài toán này. Điểm đáng chú ý nhất của mô hình này có lẽ là ở cơ chế scaled dot-product attention. Cơ chế này cho phép thông tin có thể được "chú ý" từ cả hai chiều khi tổng hợp véc-tơ biểu diễn cho một từ trong văn bản đồng thời hỗ trợ việc tính toán song song. Như vậy Transformer đã khắc phục được các khuyết điểm của các mô hình



Hình 3.19: ChatGPT giả lập kernel của Linux

Seq2Seq trước đó và đạt được nhiều cải thiện trong các bài toán dịch máy.

Mặc dù được giới thiệu như một mô hình Seq2Seq, Transformer tiếp tục được phát triển và ứng dụng trong việc xây dựng các mô hình ngôn ngữ lớn, tiêu biểu là BERT và GPT. Sử dụng encoder của Transformer, BERT phát triển theo hướng autoencoder, đưa ra dạng biểu diễn cho văn bản. Trong khi đó GPT phát triển theo hướng autoregression sử dụng decoder của Transformer, tiên đoán xác suất của các từ xuất hiện tiếp theo từ một input được cung cấp. Cả hai mô hình này đều rất thành công và thu hút sự chú ý cao. Trong khi BERT được ứng dụng để tạo thành các dạng biểu diễn thông tin cho các *downstream task* khác trong học máy thì GPT được phát triển thành các ứng dụng trợ lý ảo hỗ trợ giao tiếp và cung cấp thông tin cho con người một cách tự nhiên với ứng



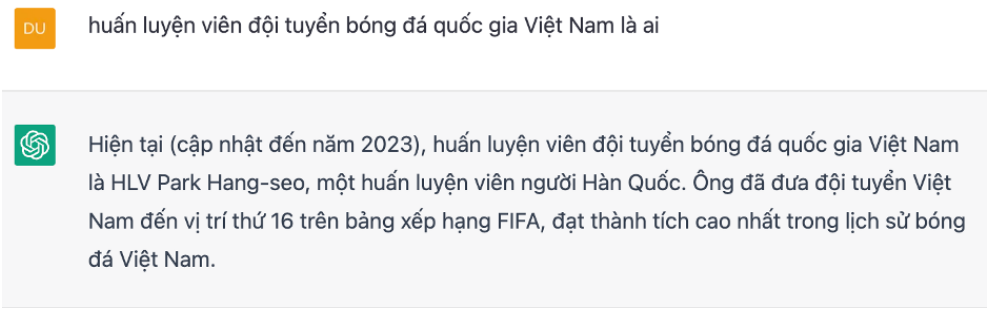
Hình 3.20: Các câu trả lời dựa trên các mẫu xác suất khác nhau của ChatGPT đôi khi không nhất quán

dụng nổi bật là ChatGPT.

3.7 Bài tập

Bài tập 3.7.1. Cho đầu vào của scaled dot-product attention gồm các ma trận:

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



Hình 3.21: ChatGPT trả lời sự kiện không chính xác sau năm 2021

$$\mathbf{W}_Q = \begin{bmatrix} 0.7 & 0.1 & 0.2 \\ 0.4 & 0.9 & 0.7 \\ 0.3 & 0.4 & 0.1 \\ 0.2 & 0.4 & 0.4 \end{bmatrix},$$

$$\mathbf{W}_K = \begin{bmatrix} 0.9 & 0.5 & 0.3 \\ 0.5 & 0.7 & 0.5 \\ 0.8 & 0.3 & 0.1 \\ 0.7 & 0.5 & 0.2 \end{bmatrix},$$

$$\mathbf{W}_V = \begin{bmatrix} 0.8 & 0.2 & 0.2 \\ 0.5 & 0.7 & 0.6 \\ 0.4 & 0.1 & 0.3 \\ 0.8 & 0.4 & 0.3 \end{bmatrix},$$

Tìm giá trị đầu ra của attention.

Bài tập 3.7.2. Cho khối Feed Forward trong mô hình Transformer gồm:

$$W_1 = \begin{bmatrix} 0.8 & 0.3 & 0.2 & 0.1 \\ 0.2 & 0.3 & 0.6 & 0.4 \\ 0.9 & 0.7 & 0.1 & 0.4 \end{bmatrix},$$

$$W_2 = \begin{bmatrix} 0.1 & 0.3 & 0.4 \\ 0.1 & 0.5 & 0.9 \\ 0.4 & 0.8 & 0.3 \\ 0.1 & 0.7 & 0.8 \end{bmatrix},$$

$$b_1 = 0.4$$

$$b_1 = 0.5$$

Với đầu vào là ma trận

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

Tìm đầu ra của khối Feed Forward.

Bảng 3.4: Vocabulary

Index	**Token**
0	[CLS]
1	[SEP]
2	[MASK]
3	tôi
4	đi
5	chạy
6	học

Bài tập 3.7.3. Giả sử bộ vocabulary của BERT gồm các token trong Bảng 3.4. Cho đầu vào của BERT là chuỗi token: ([CLS], tôi, [MASK],

học, [SEP]) và đầu ra của BERT là:

$$output = \begin{bmatrix} 0.30 & 0.25 & 0.05 & 0.10 & 0.20 & 0.03 & 0.07 \\ 0.06 & 0.04 & 0.20 & 0.35 & 0.05 & 0.20 & 0.10 \\ 0.08 & 0.20 & 0.05 & 0.10 & 0.50 & 0.05 & 0.02 \\ 0.15 & 0.05 & 0.03 & 0.07 & 0.20 & 0.10 & 0.40 \\ 0.10 & 0.60 & 0.03 & 0.02 & 0.05 & 0.10 & 0.10 \end{bmatrix}$$

Xác định token [MASK] được BERT dự đoán.

Chương 4

Các mô hình Graph Neural Networks

4.1 Giới thiệu

Graph Neural Networks (GNNs) (Gori et al., 2005) được giới thiệu lần đầu vào năm 2005 và phát triển thêm vào 2009 (Scarselli et al., 2009) là một mô hình học sâu được thực hiện trên đồ thị. Nhờ vào khả năng biểu diễn tốt thông tin đồ thị mà vẫn tiết kiệm được chi phí tính toán mà GNNs đã và đang rất được ưa chuộng trong vài năm trở lại đây. Điển hình là các mô hình cải tiến dựa trên GNNs như *Graph Attention Networks* (GATs) hay *Graph Convolution Networks* (GCNs) đang trở nên phổ biến và đóng góp lời giải cho nhiều bài toán trong nhiều lĩnh vực khác nhau (Zhou et al., 2020).

Đặc điểm mang lại sức mạnh cho GNNs đến từ bản thân các dữ liệu đầu vào đã có cấu trúc, từ đó GNNs cố gắng mô hình hóa mối liên hệ giữa các thành phần khác nhau nhưng có liên kết trong đồ thị bằng cơ chế lan truyền thông tin giữa các lân cận. Nhờ đặc tính như vậy, GNNs dễ dàng tìm ra những đặc trưng tiềm ẩn cục bộ và đặc trưng toàn cục (khi xếp chồng nhiều lớp GNN) của dữ liệu, góp phần làm tăng độ chính xác

của kết quả sau cùng (Gori et al., 2005; Scarselli et al., 2009). Các mô hình cải tiến của GNNs sau này tìm ra nhiều cách khác nhau để hiện thực hóa cơ chế lan truyền thông tin đó. Ngoài ra, nhờ đầu vào ở dạng đồ thị, do đó, GNNs rất phù hợp với các bài toán liên quan đến lý thuyết đồ thị (graph theory), mạng xã hội (social networks), thương mại điện tử (e-commerce), vận chuyển (transportation), hóa học (chemistry),...

4.2 Mô hình GNN truyền thống

4.2.1 Mô hình hóa đồ thị đầu vào

Giả sử chúng ta có một đồ thị $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ với $\mathcal{V} = \{v_i | i = \overline{1, N}\}$ là tập hợp các đỉnh và $\mathcal{E} = \{e_{ij} | i, j \in \overline{1, N}\}$ là tập hợp các cạnh trong nối giữa các đỉnh trong đồ thị. Để mô hình hóa các đỉnh và các cạnh trong một đồ thị, chúng ta có ba cách phổ biến như sau:

- *Ma trận kề* (Adjacency matrix): Ma trận vuông có kích thước $|V| \times |V|$. Phần tử ở vị trí (i, j) trên ma trận sẽ nhận giá trị 1 hoặc 0, thể hiện cho việc có cạnh có hướng nào nối từ j đến i hay không (Công thức 4.1).

$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{nếu có cạnh có hướng nối từ } j \text{ đến } i \\ 0 & \text{ngược lại} \end{cases} \tag{4.1}$$

- *Ma trận liên thuộc* (Incidence matrix): Ma trận chữ nhật có kích thước $|V| \times |E|$. Phần tử ở vị trí (i, j) sẽ nhận giá trị là 0, 1 hoặc -1 tùy thuộc vào cạnh có hướng hoặc vô hướng. Mô tả cụ thể ma

trận liên thuộc có ở (4.2).

$$\mathbf{I}_{ij} = \begin{cases} 1 & \text{nếu có cạnh } j \text{ vô hướng hoặc có hướng nối ra từ } i \\ -1 & \text{nếu có cạnh } j \text{ có hướng nối vào } i \\ 0 & \text{cạnh } j \text{ không nối đến đỉnh } i \end{cases} \tag{4.2}$$

- *Danh sách kề* (Adjacency list): Danh sách kề biểu diễn đồ thị dưới dạng một mảng $|V|$ các danh sách liên kết. Mỗi danh sách liên kết có *head* là đỉnh gốc i và các phần tử sau đó là các đỉnh j sao cho có cạnh có hướng nối từ i đến j.

Do bản chất của các phương pháp học sâu là các phép biến đổi trên ma trận, do đó cách biểu diễn bằng ma trận kề thường được sử dụng như đầu vào cho các mô hình này. Mặt khác, trong thực tế, mỗi đỉnh hoặc mỗi cạnh của đồ thị có thể có các đặc trưng riêng biệt. Ví dụ như cấu trúc của một phân tử bao gồm nhiều loại nguyên tử khác nhau (Cacbon, Oxygen, v.v.), được liên kết bằng nhiều loại liên kết khác nhau (liên kết đơn, liến kết đôi, liên kết Hydrogen, v.v.). Do đó, chúng ta cần phải mô hình hóa các đặc trưng của đỉnh hoặc cạnh để đưa vào mô hình. Ở chương này, chúng tôi sẽ mặc định đồ thị được mô hình hóa theo đỉnh. Trong trường hợp, đồ thị cần mô hình hóa theo cạnh có thể thực hiện phép biến đổi các cạnh thành đỉnh trong đồ thị mới (*line graph*) (Harary & Norman, 1960) rồi áp dụng theo các phương pháp bên dưới hoặc áp dụng các mô hình đồ thị trên cạnh (bạn đọc tự tìm hiểu thêm). Tóm lại, để biểu diễn đồ thị cho các mô hình học sâu sẽ cần ma trận đặc trưng của đỉnh $\mathbf{X}$ (Công thức 4.3) và ma trận kề $\mathbf{A}$ (Công thức 4.1).

$$\mathbf{X} = \{x_i \in \mathbb{R}^d\}, i = \overline{1, |V|} \tag{4.3}$$

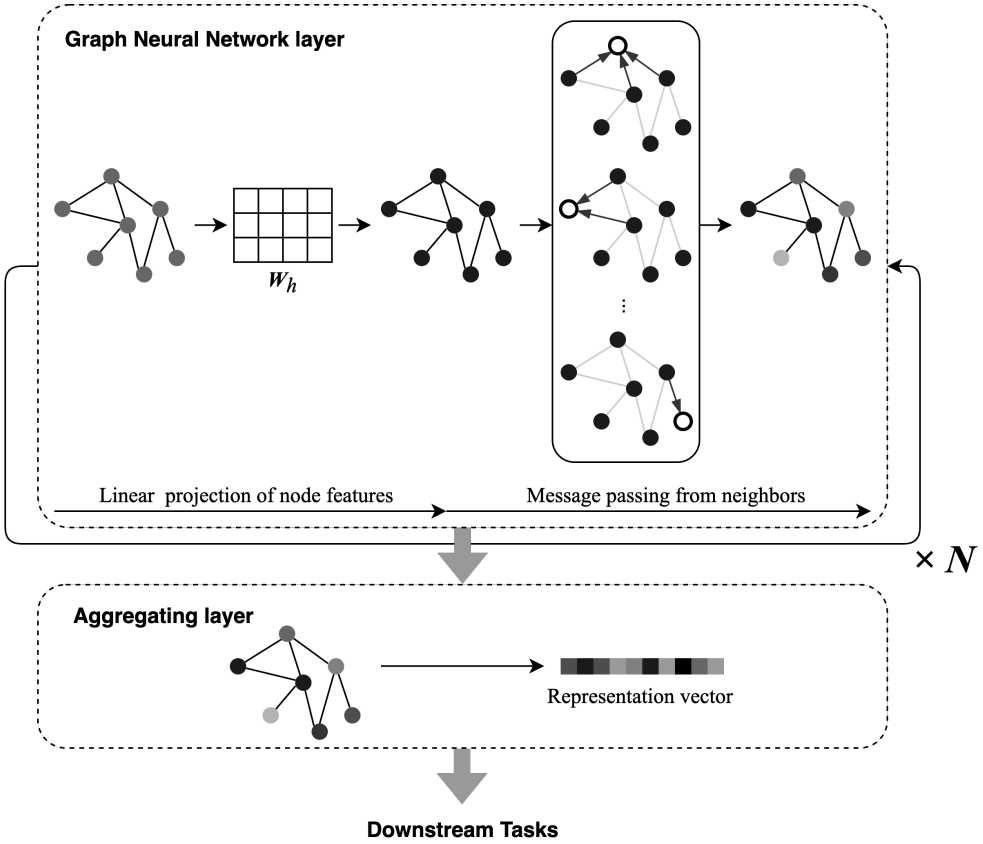
Sau khi có được biển diễn đồ thị phù hợp cho các mẫu dữ liệu, mô hình học sâu để giải quyết bài toán được xây dựng dựa trên các lớp thành phần cơ bản như sau:

- Các lớp *trích xuất đặc trưng đỉnh* (Feature extraction): Các lớp thuộc loại này có chức năng trích xuất các đặc trưng ẩn cho từng đỉnh và phải thỏa tính chất hoán vị tương đương (sẽ giải thích kỹ hơn ở phần sau).
- Các lớp *pooling*: Các lớp này có chức năng tổng quát hóa các đặc trưng mà mô hình đã học được lên một mức trừu tượng hơn. Các lớp này có thể được dùng để *kết hợp* (aggregate) đặc trưng của các đỉnh lân cận vào đỉnh đang xét hoặc để kết hợp đặc trưng của toàn bộ các đỉnh thành một véc-tơ đặc trưng cho các bài toán phân loại đồ thị.
- Các lớp *activation*: Để tối ưu bằng lan truyền ngược, các lớp activation cần được thêm vào ở giữa các lớp hoặc trong công thức tính toán trong một lớp để mô hình khả đạo hàm.
- Các lớp đầu ra: Các lớp này được dùng để học ánh xạ từ véc-tơ đại diện đến đầu ra của mô hình. Lớp đầu ra có thể là các lớp Fully-connected cho các mục đích là phân loại hoặc hồi quy.

Các lớp kể trên được sắp xếp với nhau theo nhiều cách riêng biệt để tạo nên những mô hình GNNs có các đặc tính khác nhau, tùy thuộc vào bài toán đang giải quyết. Tuy nhiên, một bản thiết kế mô hình GNNs chung thường thấy bao gồm nhiều khối các lớp trích xuất đặc trưng và activation được xếp chồng lên nhau, sau đó sẽ đến lớp pooling để kết hợp các đặc trưng lại và cuối cùng là lớp đầu ra tùy thuộc vào bài toán. Hình 4.1 thể hiện kiến trúc tổng quan phổ biến của mạng GNNs.

4.2.2 Trích xuất đặc trưng đỉnh

Quá trình trích xuất đặc trưng hay *mã hóa đỉnh* (node embedding) với mục đích biểu diễn lại đối tượng ở đỉnh vào không gian mã hóa (embedding space) một cách hiệu quả sao cho biểu diễn mới phản ánh những đỉnh "tương tự" nhau sẽ có biểu diễn gần giống nhau. Hình 4.2 mô tả một cách khái quát việc mã hóa đỉnh này, trong đó ENC(u) là hàm mã



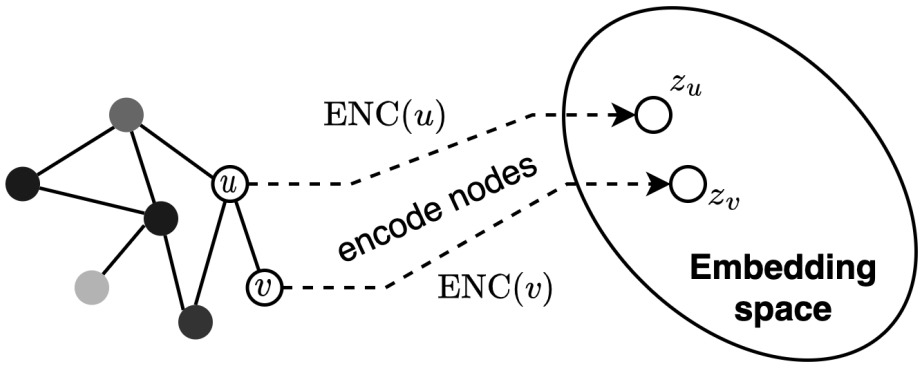
Hình 4.1: Kiến trúc tổng quan của một mô hình GNN

hóa cho đỉnh u. Một cách chính thức để định nghĩa mục đích tối ưu chính là $\text{similarity}(u, v) \approx z_v^\top z_u$. Như vậy, để mã hóa đỉnh chúng ta cần qua ba bước như sau.

1. Định nghĩa một bộ mã hóa (encoder): Hàm $\text{ENC}(u) = z_u$ để ánh xạ từ đỉnh u thành véc-tơ $z_u \in \mathbb{R}^d$.

2. Định nghĩa hàm mục tiêu (cost function): Hàm mục tiêu có thể là hàm $\text{similarity}(u, v)$ chỉ định cách mối quan hệ giữa đỉnh u và v trong không gian véc-tơ ánh xạ với các mối quan hệ trong đồ thị ban đầu. Ngoài ra, chúng ta có thể tối ưu hàm $\text{similarity}(u, v)$ gián tiếp bằng cách định nghĩa một hàm $Loss$ sao cho $\lim_{Loss \to 0} \text{similarity}(u, v) = (z_v^\top z_u)/(||z_u|| \cdot ||z_v||), \forall u, v \in \mathcal{V}$.

3. Tối ưu các tham số trong bộ mã hóa theo hàm mục tiêu sao cho similarity$(u, v) \approx (z_v^\top z_u)/(||z_u|| \cdot ||z_v||)$ hoặc $Loss \to 0$ nếu tối ưu gián tiếp.



Hình 4.2: Khái quát về ý tưởng mã hóa đỉnh

Bộ mã hóa đỉnh

Để thực hiện mã hóa đỉnh, hiện tại, chúng ta có nhiều cách tiếp cận từ có giám sát đến tự giám sát/không giám sát. Điểm khác biệt giữa các cách tiếp cận bao gồm:

- Phương pháp không/tự giám sát sử dụng chủ yếu các thông tin về cấu trúc của đồ thị thay vì đặc trưng đỉnh, nhãn đỉnh hay nhãn đồ thị,... để tạo ra các véc-tơ đặc trưng cho đỉnh. Do đó, phương pháp này phù hợp trong ngữ cảnh chúng ta không có nhiều thông tin của các đỉnh trên đồ thị.

- Các phương pháp có giám sát sẽ học được đặc trưng cho đỉnh phụ thuộc vào bài toán cụ thể cần giải. Và do đó, nếu có đủ nhiều dữ liệu với một bài toán cụ thể (như phân loại đỉnh,...) thì cách học có giám sát sẽ cho hiệu quả tốt hơn trong hầu hết trường hợp. Ngược lại, nếu chưa đủ thông tin về đồ thị hoặc mục đích chúng ta cần có

mô hình trích xuất đặc trưng có thể dùng cho nhiều bài toán khác nhau, học không/tự giám sát nên là lựa chọn ưu tiên hơn.

Trong giới hạn của sách này, chúng tôi trình bày hai cách tiếp cận mã hóa thông dụng nhất là *mã hóa cạn* sử dụng bước đi ngẫu nhiên (tự giám sát) và *mã hóa sâu* dùng các mô hình GNN (có giám sát/tự giám sát).

Mã hóa cạn (Shallow encoding)

Một cách đơn giản nhất để tạo ra bộ mã hóa chính là hiện thực nó như bộ tìm kiếm *vec-tơ nhúng của đỉnh* (embedding vector) trên *ma trận nhúng* (embedding matrix) của đỉnh. Đây chính là ý tưởng tiền thân của Mã hóa sâu dùng các mô hình Graph Neural Networks hiện đại. Với $v \in \mathbb{I}^{|\mathcal{V}|}$ là véc-tơ 1-hot chỉ định vị trí đỉnh và $\mathbf{Z} = \{z_1, z_2, \ldots, z_{|\mathcal{V}|}\} \in \mathbb{R}^{d \times |\mathcal{V}|}$ là ma trận nhúng của đỉnh chúng ta sẽ học được qua quá trình tối ưu với hàm mục tiêu, bộ mã hóa cạn được định nghĩa như (4.4).

$$\mathrm{ENC}(v) = \mathbf{Z}v \tag{4.4}$$

Ưu điểm:

- Đơn giản
- Tốc độ tính toán nhanh

Nhược điểm:

- Cần $O|\mathcal{V}|$ tham số để mã hóa một đồ thị. Trong trường hợp mã hóa một đồ thị lớn hoặc nhiều đồ thị cùng lúc sẽ không khả thi vì chi phí lưu trữ lớn.
- Mang tính "chuyển đổi" (transductive). Bản chất của mã hóa cạn là một bộ tìm kiếm véc-tơ nhúng đã có sẵn cho đúng đỉnh đang xét. Do đó, nếu có một đỉnh mới, chưa từng xuất hiện trong tập dữ liệu huấn luyện, mã hóa cạn sẽ không thể tìm được véc-tơ nhúng đó.

- Không có thông tin đặc trưng của đỉnh. Mã hóa cạn dựa theo sự giống nhau giữa các đỉnh để tạo ra véc-tơ nhúng. Vì vậy trong trường hợp hai đỉnh có thể có cấu trúc đồ thị con tương tự nhau, nhưng có đặc trưng hoàn toàn khác nhau vẫn sẽ có mã hóa gần giống nhau. Ví dụ đơn giản như định danh một người bằng những người họ quen biết, trong trường hợp hai người khác nhau quen biết với cùng tập hợp người (anh em sinh đôi còn nhỏ tuổi) thì véc-tơ biểu diễn cho hai người giống nhau và điều này không hợp lý.

Mã hóa sâu (Deep neural networks based encoding)

Hoán vị bất biến (Permutation invariance) Giả sử chúng ta có một đồ thị $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ với $\mathcal{V} = \{v_i | i = \overline{1, N}\}$ là tập hợp các đỉnh và $\mathcal{E} = \{e_{ij} | i, j \in \overline{1, N}\}$ là tập hợp các cạnh trong nối giữa các đỉnh trong đồ thị và hai ma trận quan trọng để biểu diễn đồ thị cho các mô hình là ma trận đặc trưng đỉnh $\mathbf{X} \in \mathbb{R}^{d \times |\mathcal{V}|}$ và ma trận kề $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$. Như vậy, đồ thị đầu vào cho mô hình học sâu có thể được biểu diễn dưới dạng $(\mathbf{X}, \mathbf{A})$. Với các bài toán cần trích xuất thông tin tổng quát của đồ thị (nói cách khác là ánh xạ từ đồ thị $\mathcal{G}$ thành một véc-tơ $v \in \mathbb{R}^d$), gọi $f_{graph} : (\mathbf{X}, \mathbf{A}) \mapsto \mathbb{R}^d$ là hàm trích xuất đặc trưng, tính hoán vị bất biến của hàm f_{graph} được định nghĩa như (4.5).

$$f_{graph}(\mathbf{X}, \mathbf{A}) = f_{graph}(p(\mathbf{X}), p(\mathbf{A})), \tag{4.5}$$

trong đó, $p(\cdot)$ là một phép hoán vị thứ tự biểu diễn cho các đỉnh thuộc đồ thị $\mathcal{G}$.

Ví dụ, chúng ta có $\mathcal{G}_1$ với tập đỉnh $\mathcal{V}_1 = \{v_1, v_2, v_3\}$ được biểu diễn bằng $(\mathbf{X}_1 = (x_1, x_2, x_3)$, trong đó x_i là biểu diễn của đỉnh v_i. Chúng ta cần thực hiện bài toán phân loại đồ thị (mỗi đồ thị sẽ có nhãn là một lớp nào đó). Xét một hoán vị biểu diễn với $\mathbf{X'}_1 = (x_3, x_1, x_2)$, hàm trích xuất đặc trưng cho bài toán phân loại đồ thị như trên cần thỏa mãn điều kiện

$f_{graph}(\mathbf{X}_1, \mathbf{A}_1) = f_{graph}(\mathbf{X'}_1, \mathbf{A'}_1)$.

Dễ dàng ta thấy, tính hoán vị bất biến này là một tính chất mới ở mã hóa sâu (mã hóa bằng học sâu), khác biệt với cách mã hóa riêng duy nhất cho từng đỉnh ở mã hóa cạn. Nhờ việc thiết kế kiến trúc mô hình đảm bảo tính chất này đã làm tăng *inductive bias*[1] cho các mô hình học sâu trên đồ thị sau này. Ngoài ra, tính hoán vị bất biến còn được thể hiện ở hàm kết hợp đặc trưng của các đỉnh lân cận để hình thành đặc trưng mới cho đỉnh đang xét, sẽ được thảo luận ở các phần tiếp theo.

Hoán vị tương đương (Permutation equivariance) Song song với tính hoán vị bất biến, hoán vị tương đương cũng là một tính chất hết sức quan trọng trong các mô hình học sâu trên đồ thị hiện đại. Tính hoán vị tương đương được thể hiện trong các bài toán phân loại đỉnh hay quá trình trích xuất đặc trưng cho từng đỉnh thuộc đồ thị. Với các cách biểu diễn tương tự như ở hoán vị bất biến, gọi $f_{node} : (\mathbf{X}, \mathbf{A}) \mapsto \hat{\mathbf{X}} \in \mathbb{R}^{d \times |\mathcal{V}|}$ là hàm trích xuất đặc trưng cho từng đỉnh. Tính chất hoán bị tương đương được định nghĩa như (4.6).

$$f_{node}(p(\mathbf{X}), p(\mathbf{A})) = p(f_{node}(\mathbf{X}, \mathbf{A})) \tag{4.6}$$

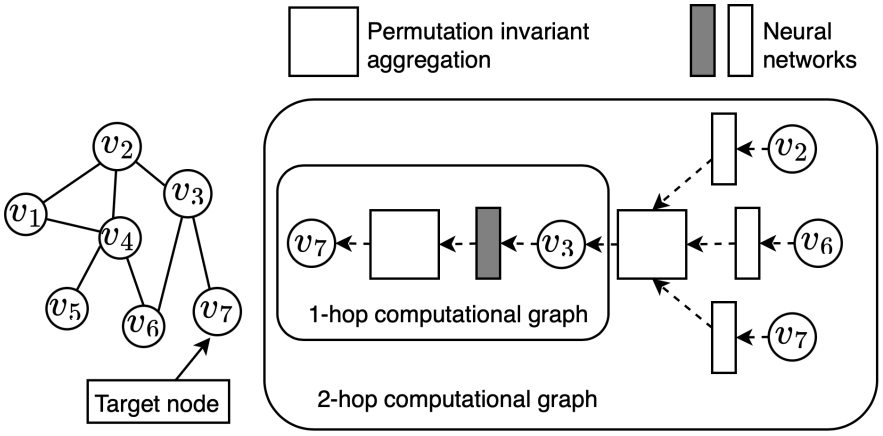
Ví dụ, với phép hoán vị $p(x_i) = x_{(i+1) \mod |\mathcal{V}|}$
, giả sử ta có $\mathbf{X} = \{x_1, x_2, x_3\}$ và f_{node} hoán vị tương đương thì $p(\mathbf{X}) = \{x_2, x_3, x_1\}$ và $\hat{\mathbf{X}} = f_{node}(p(\mathbf{X}), p(\mathbf{A})) = \{\hat{x}_2, \hat{x}_3, \hat{x}_1\}$. Và cũng dễ dàng ta có $p(f_{node}(\mathbf{X}, \mathbf{A})) = p(\{\hat{x}_1, \hat{x}_2, \hat{x}_3\}) = \{\hat{x}_2, \hat{x}_3, \hat{x}_1\}$. Như thế, (4.6) đã được chứng minh.

Đồ thị tính toán (Computational graph) Vì những đặc trưng đã trình bày ở trên cùng với mong muốn tận dụng được thông tin liên kết giữa các đỉnh của đồ thị nên những mô hình học sâu trước đây như mạng học sâu đa tầng, mạng tích chập không thể áp dụng để học trích xuất

[1] *inductive bias* hay còn gọi là thiên kiến quy nạp của một giải thuật hay mô hình đề cập đến những giả định bổ sung mà mô hình tìm được sẽ dùng để dự đoán đầu ra đúng cho các tình huống chưa gặp phải trước đây.

đặc trưng đỉnh trong đồ thị, do vậy chúng ta cần một cách tiếp cận khác hơn. Dựa trên ý tưởng tích chập của mạng nơ-ron tích chập là học từ những điểm lân cận trong ma trận, mã hóa sâu cho đồ thị sẽ học các đặc trưng đỉnh thông qua việc tổng hợp đặc trưng của các đỉnh hàng xóm lân cận. Từ đó, ta có thể xây dựng một đồ thị tính toán có cấu trúc khác nhau ứng với mỗi *đỉnh đích* (target node) như Hình 4.3. Trong đó, đồ thị tính toán bậc 1 (1-hop) và bậc 2 (2-hop) của đỉnh "C" đã được biểu diễn phẳng.

Lưu ý: Từ phần này về sau, chúng ta sẽ gọi tắt mô hình GNN chính là mô hình dùng mã hóa sâu để thuận tiện hơn.



Hình 4.3: Ví dụ minh họa đồ thị tính toán của một đỉnh

Như vậy, với mỗi đỉnh xác định trong đồ thị $\mathcal{G}$ sẽ xác định được một đồ thị tính toán tương ứng với nó. Dễ thấy rằng, điểm đáng chú ý là mô hình GNN cần phải được học và huấn luyện qua nhiều cấu trúc mô hình nơ-ron của đồ thị tính toán một cách đồng thời, điều này khác so với các mô hình truyền thống. Tương tự như các mô hình học sâu khác, với một lần tính toán cho toàn bộ đỉnh (hay còn gọi là một lớp GNN), mô hình chỉ học được những đặc trưng cục bộ của những đồ thị tính toán. Do đó, để mở rộng trường tiếp nhận (receptive field) hay còn có thể hiểu là học được những đặc trưng "toàn cục hơn", cách đơn giản nhất chỉnh là xếp nhiều lớp GNN lại với nhau. Qua mỗi lần cập nhật lại đặc trưng

cho đỉnh đích trong đồ thị tính toán, trường tiếp nhận của đỉnh đích sẽ tăng thêm xấp xỉ 1-hop. Một cách đơn giản hơn, ở tầng thứ l trong mạng GNN, véc-tơ đặc trưng của một đỉnh sẽ chứa các thông tin đặc trưng từ các đồ thị tính toán mà đỉnh chính cách điểm đang xét khoảng l hop.

Sau khi định nghĩa các khái niệm về cách tạo ra đồ thị tính toán, chúng ta sẽ tìm hiểu về các phép biến đổi trong mạng GNN dựa trên khái niệm chính là tổng hợp các đỉnh lân cận. Hiện nay có nhiều biến thể về GNN sẽ được đề cập ở Mục 4.3 và sự khác biệt chính giữa các biến thể là cách tiếp cận khác nhau về cách tổng hợp của các đỉnh lân cận. Trong phần này, tác giả chỉ nêu một cách khái quát nhất quá trình tính toán và biến đổi trong mạng GNN. Vì thứ tự của các đỉnh trong đồ thị là tùy ý nên hàm tổng hợp phải có tính chất hoán vị bất biến. Một cách tổng quát, quá trình tính toán của một tầng GNN được biểu diễn qua công thức (4.7).

$$m_u^{(l)} = \mathrm{MSG}^{(l)}(h_u^{(l-1)}) \tag{4.7a}$$

$$h_u^{(l)} = \mathrm{AGG}^{(l)}(\{m_v^{(l)}, v \in \mathcal{N}(u)\}, m_u^{(l)}) \tag{4.7b}$$

Trong (4.7a), $h_u^{(l-1)} \in \mathbb{R}^d$ là véc-tơ đặc trưng của đỉnh u ở tầng thứ $l-1$, $\mathrm{MSG}^{(l)}$ và $m_u^{(l)}$ lần lượt là hàm tạo thông tin và véc-tơ chứa thông tin của đỉnh u để truyền đi ở tầng l. Trong (4.7b), $\mathrm{AGG}^{(l)}, h_u^{(l)}$ lần lượt là hàm tổng hợp các véc-tơ thông tin từ các đỉnh lân cận và thông tin của chính đỉnh u; $\mathcal{N}(u)$ là tập hợp các đỉnh lân cận của u theo một phép lấy mẫu nhất định. Hàm $\mathrm{MSG}(\cdot)$ có thể được hiện thực bằng mô hình mạng nơ-ron đa tầng (Multi-layer perceptrons) kết hợp với các hàm kích hoạt phi tuyến (Non-linear actication functions) để học được những đặc trưng ẩn của riêng từng đỉnh. Trong khi đó, hàm $\mathrm{AGG}(\cdot)$ chính là một hàm kết hợp thỏa mã tính chất hoán vị bất biến như đã nêu ở trên. Hình 4.4 biểu diễn một ví dụ về việc tính đặc trưng đỉnh đích cho hai đồ thị tính toán thuộc cùng một đồ thị.

Với cách làm này, dễ dàng ta thấy được sau khi tính toán với công thức (4.7), thứ tự biểu diễn của các đỉnh không thay đổi, tức là thỏa mãn tính chất hoán vị tương đương như đã đề cập ở trên. Để đơn giản hóa vấn đề, chúng ta có thể hiểu công thức (4.7) chính là hàm trích xuất đặc trưng đỉnh $f_{node}^{(l)}$ ở lớp GNN thứ l. Giả sử chúng ta xây dựng mô hình GNN với L lớp, lúc này đầu ra của lớp GNN thứ L chính là đầu ra của hàm $f_{node}^{(L)}$ và bằng $\hat{\mathbf{X}} = \mathbf{H}^{(L)} = \{h_u^{(L)}, u \in \mathcal{V}\}$.

Đến lúc này, nếu mục đích xây dựng mô hình GNN để thực hiện các công việc ở mức độ đỉnh (ví dụ như phân loại đỉnh...) thì chúng ta có thể đến bước tiếp theo chính là tính toán hàm mục tiêu và thực hiện tối ưu. Ngược lại, nếu các công việc của chúng ta ở mức độ đồ thị (ví dụ phân loại đồ thị có vòng/không vòng...), khi đó, chúng ta cần một hàm kết hợp để tính toán ra véc-tơ đặc trưng cho đồ thị từ tất cả véc-tơ đặc trưng của các đỉnh thuộc đồ thị đó. Gọi $f_{graph}^{(L)}$ là hàm trích xuất đặc trưng cho toàn bộ đồ thị với L lớp GNN. Định nghĩa hình thức cho nó được trình bày ở (4.8).

$$f_{graph}^{(L)} = \mathrm{AGG}_{graph}(f_{node}^{(L)}), \tag{4.8}$$

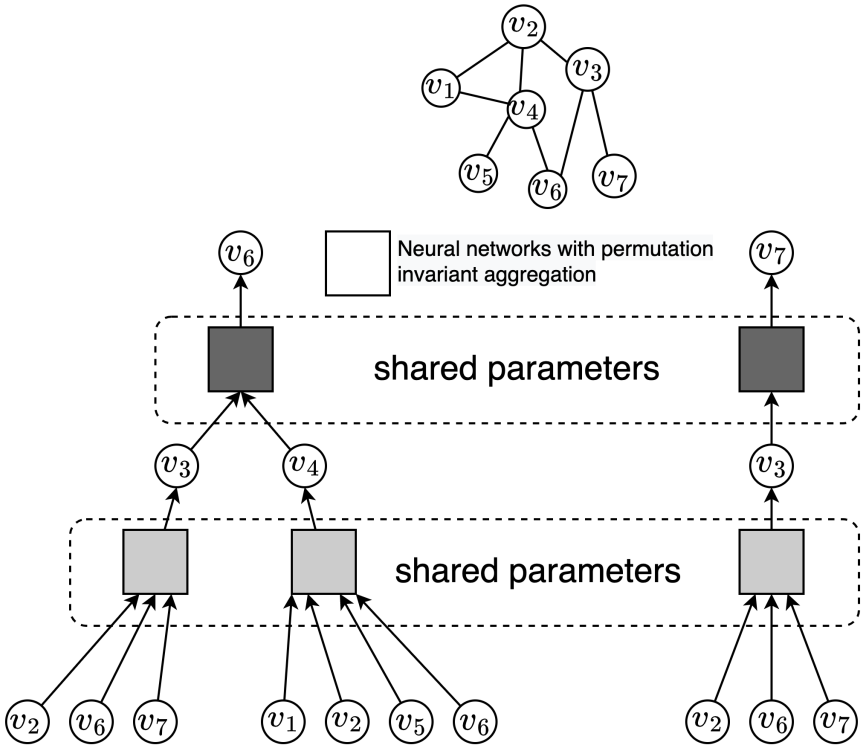
trong đó $\mathrm{AGG}_{graph}(\cdot)$ là hàm kết hợp tất cả các đặc trưng của đỉnh để tạo thành đặc trưng cho toàn bộ đồ thị, và đương nhiên, hàm này cũng phải thỏa mãn điều kiện hoán vị bất biến.

Ưu điểm:

- Đặc trưng được trích xuất phù hợp riêng cho bài toán cần giải, do đó có thể đạt được hiệu quả tốt hơn.
- Có thể xử lý hiệu quả với những đồ thị lớn và phức tạp, bao gồm nhiều đặc trưng đỉnh, đặc trưng cạnh...

Nhược điểm:

- Quá trình xây dựng mô hình cầu kỳ và phức tạp.



Hình 4.4: Ví dụ về việc tính toán đặc trưng cho các đồ thị tính toán ở một tầng GNN

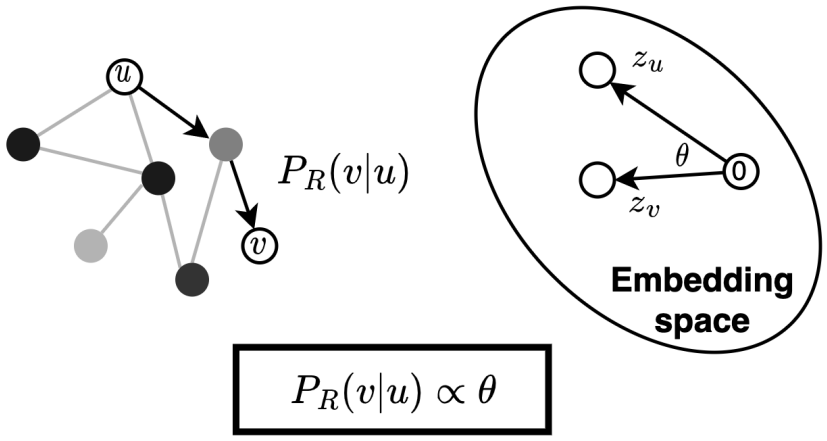
- Đôi lúc yêu cầu phần cứng tốt (GPU, TPU...) để có thể thực thi được.

4.2.3 Hàm mục tiêu và tối ưu

Mã hóa cạn

Để học được những đặc trưng của đồ thị trong điều kiện không giám sát/tự giám sát, chúng ta sẽ cần tối ưu các đặc trưng của đỉnh sao cho similarity$(u, v) \approx (z_v^\top z_u)/(||z_v^\top|| \cdot ||z_u||) = \cos(\theta)$ $(\theta = (\vec{z_u}, \vec{z_v}))$. Ở phần này, chúng tôi trình bày cách sử dụng *Bước đi ngẫu nhiên* (Random walk) (Pearson, 1905) trong hàm tính độ tương tự cho hai đỉnh u, v bất kỳ.

Đầu tiên, chúng ta tìm hiểu về Bước đi ngẫu nhiên, thế nào là Bước đi ngẫu nhiên. Với một đồ thị $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, chọn một đỉnh $v_1 \in \mathcal{V}$ làm đỉnh bắt đầu. Sau đó, chọn ngẫu nhiên một đỉnh $v_2 \in \mathcal{N}(v_2)$ là di chuyển sang nó. Quá trình chọn đỉnh và di chuyển này được lặp lại nhiều lần. Chuỗi các đỉnh ngẫu nhiên đã đi qua trong một quá trình gọi là một Bước đi ngẫu nhiên trên đồ thị. Gọi $P_R(v|u)$ là xác suất đi đến đỉnh v với một Bước đi ngẫu nhiên bắt đầu từ u sử dụng chiến lược Bước đi ngẫu nhiên R. Dễ dàng nhận ra rằng, nếu đường đi từ đỉnh u và v càng xa theo chiến lược R thì khả năng hai đỉnh này sẽ rất khác nhau và ngược lại. Như vậy, chọn similarity$(u, v) \propto P_R(v|u)$ là phù hợp để đánh giá độ giống nhau giữa hai đỉnh u và v bất kỳ. Hình 4.5 thể hiện một góc nhìn trực quan về mối liên quan giữa xác suất của Bước đi ngẫu nhiên và độ giống nhau giữa hai đỉnh u và v.



Hình 4.5: Ví dụ trực quan về Bước đi ngẫu nhiên và sự giống nhau giữa hai đỉnh bất kỳ

Với nền tảng như trên, quá trình tối ưu của mã hóa cạn sử dụng *Gradient Descent* được thực thi như sau:

1. Thực thi quá trình Bước đi ngẫu nhiên với số bước nhỏ và cố định sử dụng chiến lược R.

2. Với mỗi đỉnh bắt đầu $u \in \mathcal{V}$, lưu lại tập hợp đỉnh lân cận đã đi qua $N_R(u)$. Lưu ý rằng, một đỉnh $v \in \mathcal{V}$ có thể xuất hiện nhiều lần trong $N_R(u)$ bởi vì quá trình di chuyển ngẫu nhiên có thể quay lại đỉnh cũ nếu tồn tại đường đi. Và $N_R(u)$ không chứa đỉnh u vì chúng ta muốn tối ưu góc giữa các véc-tơ biểu diễn những đỉnh lân cận u với véc-tơ biểu diễn của u. Trong khi đó véc-tơ của u đã giống với chính nó, vì thế chúng ta không đưa vào tập $N_R(u)$.

3. Tối ưu hóa hàm mất mát $\mathcal{L}$ được định nghĩa ở (4.9).

$$\mathcal{L} = \sum_{u \in \mathcal{V}} \sum_{v \in N_R(u)} -\log(P(v|z_u)) \tag{4.9}$$

   Lưu ý rằng, ở công thức (4.9), tiền điều kiện của chúng ta là z_u thay vì u. Lý do chính là chúng ta sử dụng z_u như là véc-tơ chứa thông tin để đưa vào tính toán, do đó, tiền điều kiện thực tế phải là z_u.

   Bằng cách hệ số hóa bằng hàm Softmax, công thức (4.9) được viết lại như (4.10).

$$\mathcal{L} = \sum_{u \in \mathcal{V}} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(z_u^\top z_v)}{\sum_{n \in \mathcal{V} \setminus \{u\}} \exp(z_u^\top z_n)}\right) \tag{4.10}$$

Sau khi chúng ta có được hàm mục tiêu, quy trình tối ưu hóa hàm mục tiêu được thực hiện như sau:

1. Bước 1: Khởi tạo $z_u \in \mathbb{R}^d$ ngẫu nhiên cho toàn bộ đỉnh $u \in \mathcal{V}$.
2. Bước 2: Với mỗi đỉnh u, tính đạo hàm $\frac{\partial \mathcal{L}}{\partial z_u}$.
3. Bước 3: Với mỗi đỉnh u, cập nhật lại véc-tơ đặc trưng $z_u \leftarrow z_u - \eta \frac{\partial \mathcal{L}}{\partial z_u}$, trong đó η là hệ số tốc độ học.
4. Bước 4: Lặp lại Bước 2 và 3 cho đến khi $\mathcal{L} \to 0$.

Tuy nhiên, với quá trình tối ưu trên, có thể xảy ra trường hợp giá trị

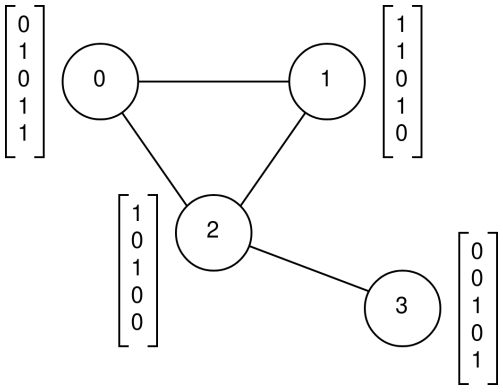
hàm tối ưu không đến được cực tiểu (người đọc tự tìm hiểu thêm), do đó, thay vì tối ưu cho toàn bộ đỉnh cùng lúc, chúng ta sẽ thực hiện tối ưu cho từng đỉnh ở mỗi lần lặp. Quá trình này gọi là *Stochastic Gradient Descent*, cụ thể như sau:

1. Bước 1: Khởi tạo $z_u \in \mathbb{R}^d$ ngẫu nhiên cho toàn bộ đỉnh $u \in \mathcal{V}$.
2. Bước 2: Chọn ngẫu nhiên một đỉnh $u \in \mathcal{V}$, tính đạo hàm $\frac{\partial \mathcal{L}^{(u)}}{\partial z_u}$ với

$$\mathcal{L}^{(u)} = \sum_{v \in N_R(u)} -\log(P(v|z_u)).$$

3. Bước 3: Với mỗi đỉnh $v \in \mathcal{V}$, cập nhật lại véc-tơ đặc trưng $z_v \leftarrow z_v - \eta \frac{\partial \mathcal{L}^{(u)}}{\partial z_v}$, trong đó η là hệ số tốc độ học.
4. Bước 4: Lặp lại Bước 2 và 3 cho đến khi $\mathcal{L} \to 0$.

Ví dụ 4.1: Với đồ thị và các véc-tơ z_u được cho như Hình 4.6. Hãy tính đặc trưng của các đỉnh dùng Stochastic Gradient Descent với hai đỉnh bắt đầu và bước đi ngẫu nhiên được chọn lần lượt là: $0 \to [2 \to 3]$ và $1 \to [2]$ và hệ số tốc độ học $\eta = 0.9$.



Hình 4.6: Đồ thị ví dụ mẫu

Lời giải: Ta có

$$\mathbf{Z} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned} \exp(z_0^\top z_1) &= e^2 \\ \exp(z_0^\top z_2) &= e^0 = 1 \\ \exp(z_0^\top z_3) &= e^1 = e \end{aligned}$$

Xét đường đi $0 \to [2 \to 3]$, ta có:

$$\begin{aligned} \mathcal{L}^{(0)} &= -\log(P(2|z_0)) - \log(P(3|z_0)) \\ &= -\log\left(\frac{\exp(z_0^\top z_2)}{\sum_{n\in\mathcal{V}} \exp(z_0^\top z_n)}\right) - \log\left(\frac{\exp(z_0^\top z_3)}{\sum_{n\in\mathcal{V}} \exp(z_0^\top z_n)}\right) \\ &= -\log\big(\exp(z_0^\top z_2)\big) - \log\big(\exp(z_0^\top z_3)\big) + 2\log\big(\exp(z_0^\top z_1) \\ &+ \exp(z_0^\top z_2) + \exp(z_0^\top z_3)\big) \\ &= -z_0^\top z_2 - z_0^\top z_3 + 2\log\big(\exp(z_0^\top z_1) + \exp(z_0^\top z_2) + \exp(z_0^\top z_3)\big) \end{aligned}$$

$$\begin{aligned} \frac{\partial \mathcal{L}^{(0)}}{z_0} &= -z_2 - z_3 + 2\frac{z_1 \exp(z_0^\top z_1) + z_2 \exp(z_0^\top z_2) + z_3 \exp(z_0^\top z_3)}{\exp(z_0^\top z_1) + \exp(z_0^\top z_2) + \exp(z_0^\top z_3)} \\ &= -z_2 - z_3 + 2\frac{e^2 z_1 + z_2 + e z_3}{1 + e + e^2} \\ &= [0.5105, 1.3305, -1.3305, 1.3305, -0.5105]^\top \end{aligned}$$

$$
\begin{aligned}
\frac{\partial \mathcal{L}^{(0)}}{z_1} &= 2\frac{z_0 \exp(z_0^\top z_1)}{\exp(z_0^\top z_1) + \exp(z_0^\top z_2) + \exp(z_0^\top z_3)} \\
&= 2\frac{e^2 z_0}{1 + e + e^2} \\
&= [0.0000, 1.3305, 0.0000, 1.3305, 1.3305]^\top
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial \mathcal{L}^{(0)}}{z_2} &= -z_0 + 2\frac{z_0 \exp(z_0^\top z_2)}{\exp(z_0^\top z_1) + \exp(z_0^\top z_2) + \exp(z_0^\top z_3)} \\
&= -z_0 + 2\frac{z_0}{1 + e + e^2} \\
&= [0.0000, -0.8199, 0.0000, -0.8199, -0.8199]^\top
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial \mathcal{L}^{(0)}}{z_3} &= -z_0 + 2\frac{z_0 \exp(z_0^\top z_3)}{\exp(z_0^\top z_1) + \exp(z_0^\top z_2) + \exp(z_0^\top z_3)} \\
&= -z_0 + 2\frac{e z_0}{1 + e + e^2} \\
&= [0.0000, -0.5105, 0.0000, -0.5105, -0.5105]^\top
\end{aligned}
$$

$$
\begin{aligned}
z_0 &= z_0 - \eta \frac{\partial \mathcal{L}^{(0)}}{z_0} \\
&= [0, 1, 0, 1, 1]^\top - 0.9 * [0.5105, 1.3305, -1.3305, 1.3305, -0.5105]^\top \\
&= [-0.4595, -0.1974, 1.1974, -0.1974, 1.4595]^\top
\end{aligned}
$$

$$
\begin{aligned}
z_1 &= z_1 - \eta \frac{\partial \mathcal{L}^{(0)}}{z_1} \\
&= [1, 1, 0, 1, 0]^\top - 0.9 * [0.0000, 1.3305, 0.0000, 1.3305, 1.3305]^\top \\
&= [1.0000, -0.1974, 0.0000, -0.1974, -1.1974]^\top
\end{aligned}
$$

$$
\begin{aligned}
z_2 &= z_2 - \eta \frac{\partial \mathcal{L}^{(0)}}{z_2} \\
&= [1, 0, 1, 0, 0]^\top - 0.9 * [0.0000, -0.8199, 0.0000, -0.8199, -0.8199]^\top \\
&= [1.0000, 0.7379, 1.0000, 0.7379, 0.7379]^\top
\end{aligned}
$$

$$
\begin{aligned}
z_3 &= z_3 - \eta \frac{\partial \mathcal{L}^{(0)}}{z_3} \\
&= [0, 0, 1, 0, 1]^\top - 0.9 * [0.0000, -0.5105, 0.0000, -0.5105, -0.5105]^\top \\
&= [0.0000, 0.4595, 1.0000, 0.4595, 1.4595]^\top
\end{aligned}
$$

Nhận xét: Dễ dàng nhận ra được, véc-tơ z_3 và z_2 đã gần hơn với véc-tơ z_0, trong đó, mức độ thay đổi khoảng cách của đỉnh đích (đỉnh ứng với z_3) cao hơn. Ngược lại, khoảng cách giữa (z_1, z_0), (z_1, z_2) và (z_1, z_3) bị đẩy ra xa hơn do đỉnh 1 không nằm trong đường đi đang xét.

$$
\begin{aligned}
\exp(z_1^\top z_0) &= \exp(-2.1292) \\
\exp(z_1^\top z_2) &= \exp(-0.1750) \\
\exp(z_1^\top z_3) &= \exp(-1.9291)
\end{aligned}
$$

Xét đường đi $1 \to [2]$, ta có:

$$\begin{aligned}
\mathcal{L}^{(1)} &= -\log(P(2|z_1)) = -\log\left(\frac{\exp(z_1^\top z_2)}{\sum_{n\in\mathcal{V}}\exp(z_1^\top z_n)}\right) \\
&= -\log\big(\exp(z_1^\top z_2)\big) + \log\big(\exp(z_1^\top z_0) + \exp(z_1^\top z_2) + \exp(z_1^\top z_3)\big) \\
&= -z_1^\top z_2 + \log\big(\exp(z_1^\top z_0) + \exp(z_1^\top z_2) + \exp(z_1^\top z_3)\big)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial\mathcal{L}^{(1)}}{z_0} &= \frac{z_1\exp(z_1^\top z_0)}{\exp(z_1^\top z_0) + \exp(z_1^\top z_2) + \exp(z_1^\top z_3)} \\
&= [0.1078, -0.0213, 0.0000, -0.0213, -0.1290]
\end{aligned}$$

$$\begin{aligned}
\frac{\partial\mathcal{L}^{(1)}}{z_1} &= -z_2 + \frac{z_0\exp(z_1^\top z_0) + z_2\exp(z_1^\top z_2) + z_3\exp(z_1^\top z_3)}{\exp(z_1^\top z_0) + \exp(z_1^\top z_2) + \exp(z_1^\top z_3)} \\
&= [-0.2889, -0.1375, 0.0213, -0.1375, 0.1727]
\end{aligned}$$

$$\begin{aligned}
\frac{\partial\mathcal{L}^{(1)}}{z_2} &= -z_1 + \frac{z_1\exp(z_1^\top z_2)}{\exp(z_1^\top z_0) + \exp(z_1^\top z_2) + \exp(z_1^\top z_3)} \\
&= [-0.2394, 0.0473, 0.0000, 0.0473, 0.2867]
\end{aligned}$$

$$\begin{aligned}
\frac{\partial\mathcal{L}^{(1)}}{z_3} &= \frac{z_1\exp(z_1^\top z_3)}{\exp(z_1^\top z_0) + \exp(z_1^\top z_2) + \exp(z_1^\top z_3)} \\
&= [0.1316, -0.0260, 0.0000, -0.0260, -0.1576]
\end{aligned}$$

$$
\begin{aligned}
z_0 &= z_0 - \eta \frac{\partial \mathcal{L}^{(1)}}{z_0} \\
&= [-0.4595, -0.1974, 1.1974, -0.1974, 1.4595]^\top \\
&- 0.9 * [0.1078, -0.0213, 0.0000, -0.0213, -0.1290]^\top \\
&= [-0.5565, -0.1783, 1.1974, -0.1783, 1.5756]^\top
\end{aligned}
$$

$$
\begin{aligned}
z_1 &= z_1 - \eta \frac{\partial \mathcal{L}^{(1)}}{z_1} \\
&= [1.0000, -0.1974, 0.0000, -0.1974, -1.1974]^\top \\
&- 0.9 * [-0.2889, -0.1375, 0.0213, -0.1375, 0.1727]^\top \\
&= [1.2600, -0.0737, -0.0191, -0.0737, -1.3529]^\top
\end{aligned}
$$

$$
\begin{aligned}
z_2 &= z_2 - \eta \frac{\partial \mathcal{L}^{(1)}}{z_2} \\
&= [1.0000, 0.7379, 1.0000, 0.7379, 0.7379]^\top \\
&- 0.9 * [-0.2394, 0.0473, 0.0000, 0.0473, 0.2867]^\top \\
&= [1.2155, 0.6954, 1.0000, 0.6954, 0.4799]^\top
\end{aligned}
$$

$$
\begin{aligned}
z_3 &= z_3 - \eta \frac{\partial \mathcal{L}^{(1)}}{z_3} \\
&= [0.0000, 0.4595, 1.0000, 0.4595, 1.4595]^\top \\
&- 0.9 * [0.1316, -0.0260, 0.0000, -0.0260, -0.1576]^\top \\
&= [-0.1185, 0.4829, 1.0000, 0.4829, 1.6014]^\top
\end{aligned}
$$

Nhận xét: Chúng ta thấy được rằng, ở đường đi đầu tiên, khoảng cách giữa z_1 và z_2 được tăng lên, do từ đỉnh 2 không đi vào đỉnh 1. Tuy nhiên, khoảng cách giữa 1 và 2 đã giảm xuống khi qua đường đi thứ hai. Điều này chứng tỏ những lập luận ban đầu của chúng ta về tính hợp lý của bước đi ngẫu nhiên và sự giống nhau giữa các đỉnh là chính xác.

Đọc thêm: Với cách hệ số hóa hàm xác suất có điều kiện $P(v|z_u)$ bằng hàm Softmax, chúng ta đã có thể sử dụng các quá trình tối ưu như Gradient Descent. Tuy nhiên, việc tính toán chuẩn hóa ở mẫu số hàm Softmax (Công thức (4.10)) trở nên kém hiệu quả vì độ phức tạp lên đến $O(|\mathcal{V}|^2)$. Giải pháp cho vấn đề này chính là, chúng ta có thể tính xấp xỉ hàm $\mathcal{L}$ bằng phương pháp Negative Sampling (Goldberg & Levy, 2014; Mikolov et al., 2013). Một cách ngắn gọn, chúng ta sẽ có công thức (4.11) thay cho (4.10) như bên dưới.

$$\mathcal{L} \approx \sum_{u \in \mathcal{V}} \sum_{v \in N_R(u)} -\left(\log(\sigma(z_u^\top z_v))) - \sum_{i=1}^{k} \log(z_u^\top z_{n_i}) \right), n_i \sim P_{\mathcal{V}}, \quad (4.11)$$

trong đó σ là hàm sigmoid, k là một siêu tham số chỉ số lượng đỉnh được chọn ngẫu nhiên và $P_{\mathcal{V}}$ là một phân bố xác suất trên toàn bộ đỉnh. Tham số k càng cao đồng nghĩa với việc hàm mục tiêu sẽ thiên vị những đỉnh ngoài đường đi, nhưng cũng đồng thời tăng tính chính xác cho phép xấp xỉ. Lưu ý, tập k đỉnh n_i được chọn, nên là những đỉnh không nằm trên những Bước đi ngẫu nhiên đã có giữa u và v, thay vì trên toàn bộ đỉnh (dễ hiện thực, nhưng độ chính xác khi xấp xỉ kém hơn).

Ngoài ra, bạn đọc cũng có thể đọc thêm mô hình *node2vec* (Grover & Leskovec, 2016) sử dụng *Bước đi ngẫu nhiên có thiên vị* (Biased Random Walk) để áp dụng cho phương pháp mã hóa cạn này.

Mã hóa sâu

Đối bài toán học có giám sát, hàm mục tiêu của mô hình được xác định bằng việc tối ưu hóa hàm mất mát cho nhiệm vụ phía sau (downstream

task) sau khi đã thu được véc-tơ nhúng đặc trưng của mỗi đỉnh. Ví dụ với bài toán phân loại đỉnh, chúng ta có thể xây dựng hàm mục tiêu là hàm *Cross Entropy* sau khi cho véc-tơ đặc trưng đỉnh qua các lớp đầu ra (Multilayer Perceptron, ...). Tóm lại, gọi *Loss* là hàm mục tiêu cần tối ưu, công thức (4.12) định nghĩa một cách tổng quát cho hàm mục tiêu này.

$$Loss = \mathcal{L}\left(\phi\left(f(\mathbf{X}, \mathbf{A})\right), y\right), \tag{4.12}$$

trong đó, $\mathcal{L}$ là hàm tính độ chênh lệch giữa đầu ra dự đoán bởi mô hình và đầu ra thực tế; y là đầu ra thực tế; $f(\cdot)$ là hàm trích xuất đặc trưng phù hợp và ϕ là hàm đầu ra của mô hình.

4.3 Các biến thể của GNNs

4.3.1 Mô hình Graph Convolution Networks

Như đã đề cập ở phần trước, các biến thể của GNN khác nhau chủ yếu về cách tạo tin và tổng hợp tin từ các đỉnh lân cận. Trong *Graph Convolution Network* (GCN) (Kipf & Welling, 2017), mô hình giới thiệu cách tiếp cận cơ bản nhất để hiện thực các ý tưởng ban đầu của GNN. Trong trường hợp không miêu tả cụ thể, mô hình GCN sẽ giả sử có cung lặp tại tất cả các đỉnh thuộc đồ thị. Mô hình GCN chi tiết hóa công thức (4.7) bằng công thức (4.13) như bên dưới.

$$m_u^{(l)} = \mathbf{W}^{(l)} h_u^{(l-1)} \tag{4.13a}$$

$$h_u^{(l)} = \sigma\left(\frac{1}{|\mathcal{N}(u)| + 1} \sum_{v \in \mathcal{N}(u) \cup \{u\}} m_v^{(l)}\right) \tag{4.13b}$$

Công thức (4.13a) là hiện thực hóa hàm $\mathrm{MSG}^{(l)}(\cdot)$ để tạo ra thông tin

truyền đi cho mỗi đỉnh tại lớp thứ l. Trong đó, ma trận $\mathbf{W}^{(l)} \in \mathbb{R}^{d' \times d}$ là ma trận trọng số mô hình. Sau đó, véc-tơ đặc trưng mới cho mỗi đỉnh được tổng hợp lại bằng cách lấy trung bình của tổng tất cả các đỉnh liền kề và chính nó. Cuối cùng, hàm kích hoạt phi tuyến σ được áp dụng để giới hạn lại hoặc điều chỉnh thông tin được tổng hợp (xem (4.13b)). Bằng cách này, với các bài toán trên đồ thị có những đỉnh đồng nhất được giải quyết hiệu quả (Ví dụ bài toán tìm mối quan hệ giữa hai người trong đồ thị quan hệ). Tuy nhiên, việc áp dụng nhiều tầng GCN sẽ đem lại bất lợi, bởi vì khi đó, đặc trưng các đỉnh sẽ gần như tương tự nhau. Bạn đọc hãy chứng minh ở bài tập bên dưới.

Ví dụ 4.2: Với đồ thị và các véc-tơ đặc trưng ban đầu của các đỉnh được cho như Hình 4.6. Ma trận $\mathbf{W}^{(l)}$) được cho như sau:

$$\mathbf{W}^{(1)} = \begin{bmatrix} 0.30 & 0.25 & 0.05 & 0.10 & 0.20 \\ 0.06 & 0.04 & 0.20 & 0.35 & 0.05 \\ 0.08 & 0.20 & 0.05 & 0.10 & 0.50 \end{bmatrix} \quad \mathbf{W}^{(2)} = \begin{bmatrix} 0.25 & 0.10 & 0.75 \\ 0.60 & 0.02 & 0.20 \\ 0.08 & 0.20 & 0.05 \\ 0.80 & 0.22 & 0.45 \\ 0.40 & 0.92 & 0.12 \end{bmatrix}$$

Hãy tính đặc trưng của tất cả các đỉnh khi qua lớp GCN thứ nhất và thứ hai sử dụng công thức đã được mô tả ở (4.13). Hàm σ được chọn là hàm ReLU.

Lời giải:

$$\mathbf{X} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$$
\begin{aligned}
\mathbf{M}^{(1)} = \mathbf{W}^{(1)}\mathbf{X} &= \begin{bmatrix} 0.30 & 0.25 & 0.05 & 0.10 & 0.20 \\ 0.06 & 0.04 & 0.20 & 0.35 & 0.05 \\ 0.08 & 0.20 & 0.05 & 0.10 & 0.50 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 0.55 & 0.65 & 0.35 & 0.25 \\ 0.44 & 0.45 & 0.26 & 0.25 \\ 0.80 & 0.38 & 0.13 & 0.55 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{H}^{(1)} &= \mathrm{ReLU}\left((\mathbf{M}^{(1)}\mathbf{A}) / \sum_i \mathbf{A}_{ij} \right) \\
&= \begin{bmatrix} 0.55 & 0.65 & 0.35 & 0.25 \\ 0.44 & 0.45 & 0.26 & 0.25 \\ 0.80 & 0.38 & 0.13 & 0.55 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} / \sum_i \left(\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \right) \\
&= \begin{bmatrix} 1.55 & 1.55 & 1.80 & 0.60 \\ 1.15 & 1.15 & 1.40 & 0.51 \\ 1.31 & 1.31 & 1.86 & 0.68 \end{bmatrix} / \begin{bmatrix} 3 \\ 3 \\ 4 \\ 2 \end{bmatrix} \\
&= \begin{bmatrix} 0.5167 & 0.5167 & 0.4500 & 0.3000 \\ 0.3833 & 0.3833 & 0.3500 & 0.2550 \\ 0.4367 & 0.4367 & 0.4650 & 0.3400 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{M}^{(2)} = \mathbf{W}^{(2)}\mathbf{H}^{(1)} &= \begin{bmatrix} 0.25 & 0.10 & 0.75 \\ 0.60 & 0.02 & 0.20 \\ 0.08 & 0.20 & 0.05 \\ 0.80 & 0.22 & 0.45 \\ 0.40 & 0.92 & 0.12 \end{bmatrix} \begin{bmatrix} 0.5167 & 0.5167 & 0.4500 & 0.3000 \\ 0.3833 & 0.3833 & 0.3500 & 0.2550 \\ 0.4367 & 0.4367 & 0.4650 & 0.3400 \end{bmatrix} \\
&= \begin{bmatrix} 0.4950 & 0.4950 & 0.4963 & 0.3555 \\ 0.4050 & 0.4050 & 0.3700 & 0.2531 \\ 0.1398 & 0.1398 & 0.1293 & 0.0920 \\ 0.6942 & 0.6942 & 0.6463 & 0.4491 \\ 0.6117 & 0.6117 & 0.5578 & 0.3954 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{H}^{(2)} &= \mathrm{ReLU}\left((\mathbf{M}^{(2)}\mathbf{A}) / \sum_i \mathbf{A}_{ij} \right) \\
&= \begin{bmatrix} 0.4950 & 0.4950 & 0.4963 & 0.3555 \\ 0.4050 & 0.4050 & 0.3700 & 0.2531 \\ 0.1398 & 0.1398 & 0.1293 & 0.0920 \\ 0.6942 & 0.6942 & 0.6463 & 0.4491 \\ 0.6117 & 0.6117 & 0.5578 & 0.3954 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} / \begin{bmatrix} 3 \\ 3 \\ 4 \\ 2 \end{bmatrix} \\
&= \begin{bmatrix} 1.4863 & 1.4863 & 1.8418 & 0.8518 \\ 1.1800 & 1.1800 & 1.4331 & 0.6231 \\ 0.4089 & 0.4089 & 0.5009 & 0.2213 \\ 2.0346 & 2.0346 & 2.4837 & 1.0954 \\ 1.7813 & 1.7813 & 2.1767 & 0.9532 \end{bmatrix} / \begin{bmatrix} 3 \\ 3 \\ 4 \\ 2 \end{bmatrix} \\
&= \begin{bmatrix} 0.4954 & 0.4954 & 0.4604 & 0.4259 \\ 0.3933 & 0.3933 & 0.3583 & 0.3116 \\ 0.1363 & 0.1363 & 0.1252 & 0.1106 \\ 0.6782 & 0.6782 & 0.6209 & 0.5477 \\ 0.5938 & 0.5938 & 0.5442 & 0.4766 \end{bmatrix}
\end{aligned}
$$

Nhận xét: Qua ví dụ trên, chúng ta thấy được, những đỉnh có cấu trúc đồ thị tính toán k-hop giống nhau, sẽ có véc-tơ biểu diễn giống nhau ở lớp thứ k (đỉnh 0 và đỉnh 1). Ngoài ra, khoảng cách các véc-tơ biểu diễn các đỉnh ở lớp thứ hai đã ngắn hơn ở lớp thứ nhất.

4.3.2 Mô hình GraphSAGE

Với biến thể *GraphSAGE* (Hamilton et al., 2017), chúng ta có cách xây dựng thông tin truyền đi ở mỗi đỉnh tương tự như GCN. Tuy nhiên, ở bước kết hợp đặc trưng các đỉnh lân cận, thay vì phải tổng hợp một lúc tất cả đỉnh lân cận cùng với đỉnh đang xét, GraphSAGE kết hợp các véc-tơ đặc trưng của các đỉnh lân cận thành một véc-tơ biểu diễn đặc trưng cho toàn bộ lân cận, sau đó mới thực hiện kết hợp với đỉnh đang xét. Do đó, GraphSAGE có khả năng nhận ra những mối liên hệ tiềm ẩn giữa các đỉnh lân cận và mô hình hóa sự ảnh hưởng của nó lên đặc trưng của đỉnh đang xét. Tên gọi GraphSAGE bắt nguồn từ *SAmple and aggregAtE*, có nghĩa là *Lấy mẫu và Kết hợp*. Các phép tính toán trong GraphSAGE được thể hiện ở công thức (4.14).

$$m_u^{(l)} = \mathbf{W}_m^{(l)} h_u^{(l-1)} \tag{4.14a}$$

$$\begin{cases} h_{\mathcal{N}(u)}^{(l)} = \mathrm{AGG}\left(\{m_v^{(l)}, \forall v \in \mathcal{N}(u)\}\right) \\ h_u^{(l)} = \sigma\left(\mathbf{W}_h^{(l)} \cdot \mathrm{CONCAT}(m_u^{(l)}, h_{\mathcal{N}(u)}^{(l)})\right) \end{cases} \tag{4.14b}$$

Trong công thức (4.14a), $\mathbf{W}_m^{(l)} \in \mathbb{R}^{d' \times d}$ mà ma trận trọng số mô hình sẽ học được. Các véc-tơ đặc trưng của các đỉnh sau khi nhân với $\mathbf{W}_m^{(l)}$ sẽ tạo ra thông tin truyền đi cho các đỉnh đó. Công thức (4.14b) tiếp tục được áp dụng ngay sau đó để kết hợp lại các véc-tơ đại diện của các đỉnh lân cận nhằm tạo ra véc-tơ đại diện chung cho toàn bộ lân cận của đỉnh $u \in \mathcal{V}$. Hàm $\mathrm{AGG}(\cdot)$ ở đây có thể là bất kỳ hàm số nào thỏa mãn tính chất hoán vị bất biến như sum, min, max, avg,... Cuối cùng, véc-tơ thông tin của đỉnh u đang xét được nối lại với véc-tơ thông tin đại diện của lân

cận u; chúng được ánh xạ ngược lại không gian đặc trưng của đỉnh bằng ma trận $\mathbf{W}_h^{(l)} \in \mathbb{R}^{d'' \times 2d'}$. Kết quả sau đó được kích hoạt bởi một hàm phi tuyến σ để điều chỉnh, giới hạn lại lượng thông tin.

Ví dụ 4.3: Với đồ thị và các véc-tơ đặc trưng ban đầu của các đỉnh được cho như Hình 4.6. Các ma trận $\mathbf{W}_m^{(l)}$ và $\mathbf{W}_h^{(l)}$ được cho như sau:

$$\mathbf{W}_m^{(1)} = \begin{bmatrix} 0.30 & 0.25 & 0.05 & 0.10 & 0.20 \\ 0.06 & 0.04 & 0.20 & 0.35 & 0.05 \\ 0.08 & 0.20 & 0.05 & 0.10 & 0.50 \end{bmatrix}$$

$$\mathbf{W}_h^{(1)} = \begin{bmatrix} 0.20 & 0.20 & 0.75 & 0.90 & 0.10 & 0.60 \\ 0.65 & 0.70 & 0.25 & 0.85 & 0.15 & 0.25 \\ 0.10 & 0.50 & 0.35 & 0.30 & 0.55 & 0.80 \end{bmatrix}$$

$$\mathbf{W}_m^{(2)} = \begin{bmatrix} 0.35 & 0.95 & 0.35 \\ 0.05 & 0.85 & 0.50 \\ 0.05 & 0.60 & 0.05 \end{bmatrix} \quad \mathbf{W}_h^{(2)} = \begin{bmatrix} 0.90 & 0.20 & 0.45 & 0.05 & 0.15 & 0.70 \\ 0.60 & 0.70 & 0.60 & 0.80 & 0.15 & 0.25 \\ 0.10 & 0.50 & 0.75 & 0.20 & 0.50 & 0.85 \end{bmatrix}$$

Hãy tính đặc trưng của tất cả các đỉnh khi qua lớp GraphSAGE thứ nhất và thứ hai sử dụng công thức đã được mô tả ở (4.14). Hàm AGG$(\cdot)$ là hàm tính trung bình (mean) và σ được chọn là hàm ReLU.

Lời giải:

$$\mathbf{X} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$
\mathbf{M}^{(1)} = \mathbf{W}_m^{(1)}\mathbf{X} = \begin{bmatrix} 0.30 & 0.25 & 0.05 & 0.10 & 0.20 \\ 0.06 & 0.04 & 0.20 & 0.35 & 0.05 \\ 0.08 & 0.20 & 0.05 & 0.10 & 0.50 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}
$$

$$
= \begin{bmatrix} 0.55 & 0.65 & 0.35 & 0.25 \\ 0.44 & 0.45 & 0.26 & 0.25 \\ 0.80 & 0.38 & 0.13 & 0.55 \end{bmatrix}
$$

$$
\mathbf{H}_{\mathcal{N}}^{(1)} = (\mathbf{M}^{(1)}\mathbf{A}) / \sum_i \mathbf{A}_{ij}
$$

$$
= \begin{bmatrix} 0.55 & 0.65 & 0.35 & 0.25 \\ 0.44 & 0.45 & 0.26 & 0.25 \\ 0.80 & 0.38 & 0.13 & 0.55 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} / \begin{bmatrix} 2 \\ 2 \\ 3 \\ 1 \end{bmatrix}
$$

$$
= \begin{bmatrix} 1.00 & 0.90 & 1.45 & 0.35 \\ 0.71 & 0.70 & 1.14 & 0.26 \\ 0.51 & 0.93 & 1.73 & 0.13 \end{bmatrix} / \begin{bmatrix} 2 \\ 2 \\ 3 \\ 1 \end{bmatrix}
$$

$$
= \begin{bmatrix} 0.5000 & 0.4500 & 1.4833 & 0.3500 \\ 0.3550 & 0.3500 & 1.3800 & 0.2600 \\ 0.2550 & 0.4650 & 1.5767 & 0.1300 \end{bmatrix}
$$

$$
\begin{aligned}
\mathbf{H}^{(1)} &= \mathrm{ReLU}\left(\mathbf{W}_h^{(1)} \cdot \mathrm{CONCAT}_i(\mathbf{M}^{(l)}\mathbf{H}_{\mathcal{N}}^{(1)})\right) \\
&= \begin{bmatrix} 0.20 & 0.20 & 0.75 & 0.90 & 0.10 & 0.60 \\ 0.65 & 0.70 & 0.25 & 0.85 & 0.15 & 0.25 \\ 0.10 & 0.50 & 0.35 & 0.30 & 0.55 & 0.80 \end{bmatrix} \begin{bmatrix} 0.55 & 0.65 & 0.35 & 0.25 \\ 0.44 & 0.45 & 0.26 & 0.25 \\ 0.80 & 0.38 & 0.13 & 0.55 \\ 0.5000 & 0.4500 & 1.4833 & 0.3500 \\ 0.3550 & 0.3500 & 1.3800 & 0.2600 \\ 0.2550 & 0.4650 & 1.5767 & 0.1300 \end{bmatrix} \\
&= \begin{bmatrix} 1.4365 & 1.2240 & 1.0385 & 0.9315 \\ 1.4075 & 1.3837 & 1.0540 & 0.8440 \\ 1.1042 & 1.1225 & 1.0258 & 0.6945 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{M}^{(2)} = \mathbf{W}_m^{(2)}\mathbf{H}^{(1)} &= \begin{bmatrix} 0.35 & 0.95 & 0.35 \\ 0.05 & 0.85 & 0.50 \\ 0.05 & 0.60 & 0.05 \end{bmatrix} \begin{bmatrix} 1.4365 & 1.2240 & 1.0385 & 0.9315 \\ 1.4075 & 1.3837 & 1.0540 & 0.8440 \\ 1.1042 & 1.1225 & 1.0258 & 0.6945 \end{bmatrix} \\
&= \begin{bmatrix} 2.2264 & 2.1358 & 1.7238 & 1.3709 \\ 1.8203 & 1.7986 & 1.4607 & 1.1112 \\ 0.9715 & 0.9476 & 0.7356 & 0.5877 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{H}_{\mathcal{N}}^{(2)} &= (\mathbf{M}^{(2)}\mathbf{A}) / \sum_i \mathbf{A}_{ij} \\
&= \begin{bmatrix} 2.2264 & 2.1358 & 1.7238 & 1.3709 \\ 1.8203 & 1.7986 & 1.4607 & 1.1112 \\ 0.9715 & 0.9476 & 0.7356 & 0.5877 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} / \begin{bmatrix} 2 \\ 2 \\ 3 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 3.8597 & 3.9502 & 5.7331 & 1.7238 \\ 3.2594 & 3.2811 & 4.7302 & 1.4607 \\ 1.6832 & 1.7072 & 2.5068 & 0.7356 \end{bmatrix} / \begin{bmatrix} 2 \\ 2 \\ 3 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 1.9298 & 1.9751 & 1.9110 & 1.7238 \\ 1.6297 & 1.6405 & 1.5767 & 1.4607 \\ 0.8416 & 0.8536 & 0.8356 & 0.7356 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{H}^{(2)} &= \mathrm{ReLU}\left(\mathbf{W}_h^{(2)} \cdot \mathrm{CONCAT}_i(\mathbf{M}^{(2)}\mathbf{H}_{\mathcal{N}}^{(2)})\right) \\
&= \begin{bmatrix} 0.90 & 0.20 & 0.45 & 0.05 & 0.15 & 0.70 \\ 0.60 & 0.70 & 0.60 & 0.80 & 0.15 & 0.25 \\ 0.10 & 0.50 & 0.75 & 0.20 & 0.50 & 0.85 \end{bmatrix} \begin{bmatrix} 2.2264 & 2.1358 & 1.7238 & 1.3709 \\ 1.8203 & 1.7986 & 1.4607 & 1.1112 \\ 0.9715 & 0.9476 & 0.7356 & 0.5877 \\ 1.9298 & 1.9751 & 1.9110 & 1.7238 \\ 1.6297 & 1.6405 & 1.5767 & 1.4607 \\ 0.8416 & 0.8536 & 0.8356 & 0.7356 \end{bmatrix} \\
&= \begin{bmatrix} 3.7351 & 3.6507 & 3.0916 & 2.5408 \\ 5.1917 & 5.1486 & 4.4724 & 3.7351 \\ 3.7776 & 3.7644 & 3.3353 & 2.8339 \end{bmatrix}
\end{aligned}
$$

Nhận xét: Với mô hình GraphSAGE chúng ta thấy được rằng các đỉnh có đồ thị tính toán giống nhau chưa chắc đã có véc-tơ biểu diễn hoàn toàn giống nhau. Điều này làm cho GraphSAGE khác biệt với mô hình

GCN ban đầu. GraphSAGE đạt được điều này bởi vì có thêm một ma trận trọng số dành riêng cho việc tổng hợp thông tin từ đỉnh đích và các đỉnh lân cận. Nhờ đặc tính này, GraphSAGE có hiệu quả hơn GCN trong các bài toán cần đánh giá mức độ đóng góp của các đỉnh lân cận tách biệt với đỉnh đích.

4.3.3 Mô hình Graph Attention Networks

Trong vài năm trở lại đây, *cơ chế attention* (attention mechanism) (Vaswani et al., 2017) trở nên rất nổi bật vì những đóng góp quan trọng của nó trong việc đánh giá tầm quan trọng của một đối tượng với đối tượng khác. Điều này mô phỏng lại quá trình nhận thức tự nhiên của con người và đã giúp ích cho các mô hình học sâu hiện đại đánh giá đúng hơn mối liên hệ giữa các đối tượng đang xem xét. *Graph Attention Network* (GAT) (Veličković et al., 2018) sử dụng cơ chế attention để đánh giá độ quan trọng của mỗi đỉnh lân cận với đỉnh đang xem xét, từ đó tổng hợp thông tin theo mức độ quan trọng đó. Đỉnh lân cận càng quan trọng thì thông tin từ đỉnh đó sẽ ảnh hưởng càng nhiều vào thông tin của đỉnh đang xét. Công thức tổng quát của GAT được mô tả ở (4.15) như bên dưới.

$$m_u^{(l)} = \mathbf{W}_m^{(l)} h_u^{(l-1)} \tag{4.15a}$$

$$h_u^{(l)} = \sigma\left(\sum_{v \in \mathcal{N}(u) \cup \{u\}} \alpha_{uv} m_v^{(l)}\right) \tag{4.15b}$$

Trong công thức (4.15a), $\mathbf{W}_m^{(l)} \in \mathbb{R}^{d' \times d}$ là ma trận trọng số mô hình sẽ học được. Còn ở (4.15b), σ là hàm kích hoạt phi tuyến và α_{uv} thể hiện mức độ quan trọng của đỉnh v lên đỉnh u. α_{uv} thông thường được gọi là *hệ số chú ý* (attention coefficient) và được định nghĩa như (4.16).

$$e_{uv}^{(l)} = \mathrm{F}(m_u^{(l)}, m_v^{(l)}), \tag{4.16a}$$

$$\alpha_{uv}^{(l)} = \frac{\exp(e_{uv}^{(l)})}{\sum_{k \in \mathcal{N}(u) \cup \{u\}} \exp(e_{uk}^{(l)})}, \tag{4.16b}$$

trong đó, $\mathrm{F}(\cdot)$ là một hàm số trả về mức độ liên quan giữa hai véc-tơ đầu vào. Một số hàm $\mathrm{F}(\cdot)$ thông dụng được liệt kê như bên dưới.

- Dot product attention (Vaswani et al., 2017): $\mathrm{F}(x, y) = x^\top y$
- Luong attention (Luong et al., 2015): $\mathrm{F}(x, y) = x^\top \mathbf{W} y$
- Bahdanau attention (Bahdanau et al., 2015b): $\mathrm{F}(x, y) = v^\top \tanh(\mathbf{W}_2 x + \mathbf{W}_2 y)$

Như vậy, chúng ta có thể thấy rằng, GAT có thể mô hình hóa mối tương quan mạnh yếu giữa các đỉnh trong đồ thị, từ đó quyết định lượng thông tin mà đỉnh lân cận có thể truyền vào đỉnh đang xem xét. Có nhiều cách để hiện thực hàm tính độ quan trọng giữa các đỉnh, từ đó làm cho GAT có khả năng khái quát hóa cao và có thể áp dụng cho nhiều bài toán ở nhiều lĩnh vực khác nhau.

Ví dụ 4.4: Với đồ thị và các véc-tơ đặc trưng ban đầu của các đỉnh được cho như Hình 4.6. Ma trận $\mathbf{W}^{(l)}$) được cho như sau:

$$\mathbf{W}^{(1)} = \begin{bmatrix} 0.30 & 0.25 & 0.05 & 0.10 & 0.20 \\ 0.06 & 0.04 & 0.20 & 0.35 & 0.05 \\ 0.08 & 0.20 & 0.05 & 0.10 & 0.50 \end{bmatrix} \quad \mathbf{W}^{(2)} = \begin{bmatrix} 0.25 & 0.10 & 0.75 \\ 0.60 & 0.02 & 0.20 \\ 0.08 & 0.20 & 0.05 \\ 0.80 & 0.22 & 0.45 \\ 0.40 & 0.92 & 0.12 \end{bmatrix}$$

Hãy tính đặc trưng của tất cả các đỉnh khi qua lớp GAT thứ nhất và thứ hai sử dụng công thức đã được mô tả ở (4.15). Hàm σ được chọn là hàm

ReLU và hàm tính attention được chọn là *Dot Product*.

Lời giải:

$$\mathbf{X} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\mathbf{M}^{(1)} = \mathbf{W}^{(1)}\mathbf{X} = \begin{bmatrix} 0.30 & 0.25 & 0.05 & 0.10 & 0.20 \\ 0.06 & 0.04 & 0.20 & 0.35 & 0.05 \\ 0.08 & 0.20 & 0.05 & 0.10 & 0.50 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.55 & 0.65 & 0.35 & 0.25 \\ 0.44 & 0.45 & 0.26 & 0.25 \\ 0.80 & 0.38 & 0.13 & 0.55 \end{bmatrix}$$

$$\mathbf{E}^{(1)} = (\mathbf{M}^{(1)})^\top \mathbf{M}^{(1)} = \begin{bmatrix} 0.55 & 0.44 & 0.80 \\ 0.65 & 0.45 & 0.38 \\ 0.35 & 0.26 & 0.13 \\ 0.25 & 0.25 & 0.55 \end{bmatrix} \begin{bmatrix} 0.55 & 0.65 & 0.35 & 0.25 \\ 0.44 & 0.45 & 0.26 & 0.25 \\ 0.80 & 0.38 & 0.13 & 0.55 \end{bmatrix}$$

$$= \begin{bmatrix} 1.1361 & 0.8595 & 0.4109 & 0.6875 \\ 0.8595 & 0.7694 & 0.3939 & 0.4840 \\ 0.4109 & 0.3939 & 0.2070 & 0.2240 \\ 0.6875 & 0.4840 & 0.2240 & 0.4275 \end{bmatrix}$$

$$
\begin{aligned}
\mathcal{A}^{(1)} &= \text{softmax}_j(\mathbf{E}^{(1)} \otimes \mathbf{A}) \\
&= \text{softmax}_j \left(\begin{bmatrix} 1.1361 & 0.8595 & 0.4109 & 0.6875 \\ 0.8595 & 0.7694 & 0.3939 & 0.4840 \\ 0.4109 & 0.3939 & 0.2070 & 0.2240 \\ 0.6875 & 0.4840 & 0.2240 & 0.4275 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \right) \\
&= \text{softmax}_j \left(\begin{bmatrix} 1.1361 & 0.8595 & 0.4109 & 0.0000 \\ 0.8595 & 0.7694 & 0.3939 & 0.0000 \\ 0.4109 & 0.3939 & 0.2070 & 0.2240 \\ 0.0000 & 0.0000 & 0.2240 & 0.4275 \end{bmatrix} \right) \\
&= \begin{bmatrix} 0.4459 & 0.3382 & 0.2159 & 0.0000 \\ 0.3935 & 0.3596 & 0.2470 & 0.0000 \\ 0.2756 & 0.2710 & 0.2248 & 0.2286 \\ 0.0000 & 0.0000 & 0.4493 & 0.5507 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{H}^{(1)} &= (\mathcal{A}^{(1)}(\mathbf{M}^{(1)})^\top)^\top \\
&= \left(\begin{bmatrix} 0.4459 & 0.3382 & 0.2159 & 0.0000 \\ 0.3935 & 0.3596 & 0.2470 & 0.0000 \\ 0.2756 & 0.2710 & 0.2248 & 0.2286 \\ 0.0000 & 0.0000 & 0.4493 & 0.5507 \end{bmatrix} \begin{bmatrix} 0.55 & 0.44 & 0.80 \\ 0.65 & 0.45 & 0.38 \\ 0.35 & 0.26 & 0.13 \\ 0.25 & 0.25 & 0.55 \end{bmatrix} \right)^\top \\
&= \begin{bmatrix} 0.5406 & 0.5366 & 0.4636 & 0.2949 \\ 0.4045 & 0.3991 & 0.3588 & 0.2545 \\ 0.5133 & 0.4835 & 0.4784 & 0.3613 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{M}^{(2)} &= \mathbf{W}^{(2)}\mathbf{H}^{(1)} \\
&= \begin{bmatrix} 0.25 & 0.10 & 0.75 \\ 0.60 & 0.02 & 0.20 \\ 0.08 & 0.20 & 0.05 \\ 0.80 & 0.22 & 0.45 \\ 0.40 & 0.92 & 0.12 \end{bmatrix} \begin{bmatrix} 0.5406 & 0.5366 & 0.4636 & 0.2949 \\ 0.4045 & 0.3991 & 0.3588 & 0.2545 \\ 0.5133 & 0.4835 & 0.4784 & 0.3613 \end{bmatrix} \\
&= \begin{bmatrix} 0.5606 & 0.5367 & 0.5106 & 0.3702 \\ 0.4351 & 0.4266 & 0.3810 & 0.2543 \\ 0.1498 & 0.1469 & 0.1328 & 0.0926 \\ 0.7525 & 0.7346 & 0.6651 & 0.4545 \\ 0.6500 & 0.6398 & 0.5729 & 0.3955 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{E}^{(2)} &= (\mathbf{M}^{(2)})^\top \mathbf{M}^{(2)} \\
&= \begin{bmatrix} 0.5606 & 0.5367 & 0.5106 & 0.3702 \\ 0.4351 & 0.4266 & 0.3810 & 0.2543 \\ 0.1498 & 0.1469 & 0.1328 & 0.0926 \\ 0.7525 & 0.7346 & 0.6651 & 0.4545 \\ 0.6500 & 0.6398 & 0.5729 & 0.3955 \end{bmatrix}^\top \begin{bmatrix} 0.5606 & 0.5367 & 0.5106 & 0.3702 \\ 0.4351 & 0.4266 & 0.3810 & 0.2543 \\ 0.1498 & 0.1469 & 0.1328 & 0.0926 \\ 0.7525 & 0.7346 & 0.6651 & 0.4545 \\ 0.6500 & 0.6398 & 0.5729 & 0.3955 \end{bmatrix} \\
&= \begin{bmatrix} 1.5148 & 1.4772 & 1.3448 & 0.9311 \\ 1.4772 & 1.4407 & 1.3112 & 0.9077 \\ 1.3448 & 1.3112 & 1.1941 & 0.8270 \\ 0.9311 & 0.9077 & 0.8270 & 0.5732 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{A}^{(2)} &= \text{softmax}_j(\mathbf{E}^{(2)} \otimes \mathbf{A}) \\
&= \text{softmax}_j \left(\begin{bmatrix} 1.5148 & 1.4772 & 1.3448 & 0.9311 \\ 1.4772 & 1.4407 & 1.3112 & 0.9077 \\ 1.3448 & 1.3112 & 1.1941 & 0.8270 \\ 0.9311 & 0.9077 & 0.8270 & 0.5732 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \right) \\
&= \text{softmax}_j \left(\begin{bmatrix} 1.5148 & 1.4772 & 1.3448 & 0.0000 \\ 1.4772 & 1.4407 & 1.3112 & 0.0000 \\ 1.3448 & 1.3112 & 1.1941 & 0.8270 \\ 0.0000 & 0.0000 & 0.8270 & 0.5732 \end{bmatrix} \right) \\
&= \begin{bmatrix} 0.3563 & 0.3431 & 0.3006 & 0.0000 \\ 0.3557 & 0.3430 & 0.3013 & 0.0000 \\ 0.2921 & 0.2825 & 0.2513 & 0.1741 \\ 0.0000 & 0.0000 & 0.5631 & 0.4369 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{H}^{(2)} &= (\mathcal{A}^{(1)}(\mathbf{M}^{(2)})^\top)^\top \\
&= \left(\begin{bmatrix} 0.3563 & 0.3431 & 0.3006 & 0.0000 \\ 0.3557 & 0.3430 & 0.3013 & 0.0000 \\ 0.2921 & 0.2825 & 0.2513 & 0.1741 \\ 0.0000 & 0.0000 & 0.5631 & 0.4369 \end{bmatrix} \begin{bmatrix} 0.5606 & 0.5367 & 0.5106 & 0.3702 \\ 0.4351 & 0.4266 & 0.3810 & 0.2543 \\ 0.1498 & 0.1469 & 0.1328 & 0.0926 \\ 0.7525 & 0.7346 & 0.6651 & 0.4545 \\ 0.6500 & 0.6398 & 0.5729 & 0.3955 \end{bmatrix}^\top \right)^\top \\
&= \begin{bmatrix} 0.5374 & 0.5373 & 0.5081 & 0.4493 \\ 0.4159 & 0.4159 & 0.3876 & 0.3256 \\ 0.1437 & 0.1437 & 0.1348 & 0.1152 \\ 0.7201 & 0.7200 & 0.6736 & 0.5731 \\ 0.6233 & 0.6233 & 0.5834 & 0.4954 \end{bmatrix}
\end{aligned}
$$

Nhận xét: Khi áp dụng cơ chế attention, trong bán kính 1 hop, những

đỉnh nào có đặc trưng càng gần (hỗ trợ) với đặc trưng của đỉnh trung tâm sẽ có hệ số attention cao, từ đó ảnh hưởng của nó lên đỉnh trung tâm được tăng cao. Nhờ đặc trưng như vậy mà Graph Attention Networks có khả năng phân biệt những đỉnh có cấu trúc đồ thị tính toán giống nhau. Tuy nhiên, trong trường hợp đồ thị tính toán hoàn toàn giống nhau (cấu trúc và đặc trưng của đỉnh), đặc trưng của hai đỉnh vẫn tuân theo quy luật ban đầu như mô hình Graph Convolutional Neural Networks truyền thống.

4.3.4 Mô hình Gated Graph Neural Networks

Tận dụng ý tưởng của Graph Convolutinal Networks là học trên đồ thị với nhiều lớp mô hình và ý tưởng của *Gated Recurrent Neural Networks* (GRNN) (Y. Li et al., 2016) về kiểm soát lượng thông tin mô hình sẽ học được sau mỗi bước, *Gated Graph Neural Networks* (GGNN) sử dụng một cổng để kiểm soát lượng thông tin biểu diễn của mỗi đỉnh qua từng lớp. Công thức của Gated Graph Neural Networks được mô tả như ở (4.17). Trong đó, GRU$(\cdot)$ là hàm đại diện cho mô hình GRU (xem lại phần 1.2). Các trọng số của mô hình GRU trong một lớp GGNN có thể được chia sẻ khi nhiều lớp GGNN được xếp chồng lên nhau tạo thành một cơ chế quản lý thông tin giữa các lớp GGNN trong toàn bộ mô hình.

$$m_u^{(l)} = \mathbf{W}^{(l)} \sum_{v \in \mathcal{N}(u)} h_v^{(l-1)} \tag{4.17a}$$

$$h_u^{(l)} = \mathrm{GRU}(h_u^{(l-1)}, m_u^{(l)}) \tag{4.17b}$$

4.3.5 Các mô hình đồ thị khác

Những năm trở lại đây, các mô hình dựa trên đồ thị được phát triển rất mạnh mẽ và được ứng dụng sâu vào các vấn đề trong thực tế, đặc biệt là lĩnh vực hóa dược (Reiser et al., 2022). Các mô hình mới đều dựa trên nền tảng là ý tưởng chung của GNN phát triển lên, kết hợp với các

ràng buộc và kỹ thuật cụ thể để thỏa mãn tính chất riêng cho từng loại bài toán. Cụ thể hơn, các mô hình sử dụng một cơ chế có tên *Mesage Passing Neural Networks* (MPNN) (Gilmer et al., 2017), cơ chế này cho phép tạo ra những "message" đặc trưng cho một cặp đỉnh cụ thể (ví dụ (u, v)) dựa trên đặc trưng của hai đỉnh (z_u, z_v)và đặc trưng của cạnh nối hai đỉnh đó (e_{uv}). Bạn đọc có thểm tham khảo các bài báo liên quan ở một số nghiên cứu sau (Gilmer et al., 2017; Han et al., 2022; Jo et al., 2022).

Ngoài ra, một nhánh khá mới nữa chính là áp dụng mô hình đồ thị vào các bài toán có tọa độ như mô phỏng chuyển động vật lý của hạt cơ bản, mô phỏng sự di chuyển của dân cư, tính toán tọa độ của các hợp chất hóa học khi tương tác với nhau... Lĩnh vực này thường được nhắc đến dưới tên gọi *Geometric Deep Learning* (Bronstein et al., 2021). Điểm đặc biệt của các bài toán dạng này chính là phải mô hình hóa các thông tin về tọa độ vào đồ thị và thực hiện trích xuất đặc trưng trên những thông tin tọa độ đó. Hơn thế nữa, các bài toán dạng này cần tính toán và cập nhật lại tọa độ mới của các đối tượng. Điều này trở thành một thách thức khi có các ràng buộc về vật lý như tốc độ chuyển động của hạt, trọng lực của trái đất, từ trường, cấu trúc cố định của phân tử... Ví dụ, trong không gian ba chiều, chúng ta muốn tìm được vị trí tương đối khi tương tác của hai hợp chất và giữ nguyên cấu trúc ban đầu, mô hình đồ thị xây dựng phải thỏa mãn tính chất xoay bất biến và tịnh tiến bất biến. Bạn đọc có thể tìm hiểu các chủ đề thú vị của lĩnh vực này tại (Sanchez-Gonzalez et al., 2020; Shlomi et al., 2020; Stärk et al., 2022).

4.4 Các dạng bài toán áp dụng GNNs điển hình

4.4.1 Các mô hình dự đoán toàn đồ thị

Trong thực tế, có nhiều bài toán cần phân loại đồ thị hay dự đoán một con số là tính chất hay đặc trưng của đối tượng được mô hình hóa bằng

đồ thị đó. Những bài toán tiêu biểu trong các lĩnh vực có thể kể đến như:

- Hình ảnh: phân loại hình ảnh dựa trên các thành phần liên thông, phân tích đặc trưng vật thể dựa vào các vật thể tương đương,...
- Âm thanh: chuyển đổi lời nói thành văn bản,...
- Văn bản: phân loại câu/đoạn văn dựa vào cấu trúc, dịch máy bảo toàn cấu trúc câu...
- Các lĩnh vực khác: mô phỏng chuyển động vật lý của các nguyên tử, dự đoán tính chất hóa học của các hợp chất,...

Một cách khái quát, các bài toán trên khi áp dụng các mô hình GNNs để giải quyết thường đi qua ba bước chính:

1. Mã hóa đồ thị đầu vào;
2. Kết hợp véc-tơ đặc trưng của các đỉnh để tạo thành véc-tơ đại diện cho toàn bộ đồ thị;
3. Sử dụng các bộ classifier hay regressor phù hợp để tính toán đầu ra của mô hình.

Gọi GNN : $(\mathbf{X}, \mathbf{A}) \mapsto \hat{\mathbf{X}} \in \mathbb{R}^{d \times |\mathcal{V}|}$ là hàm số đại diện cho một lớp GNN dùng để trích xuất thông tin ẩn từ các đỉnh. Kết hợp cùng công thức (4.8), chúng ta có một khung mô hình hoàn chỉnh dành cho các bài toán dự đoán trên toàn đồ thị như được mô tả ở (4.18).

$$\hat{\mathbf{X}} = \mathrm{GNN}(\mathbf{X}, \mathbf{A}) \tag{4.18a}$$

$$h_{graph} = \mathrm{AGG}_{graph}(\hat{\mathbf{X}}) \tag{4.18b}$$

$$\hat{y}_{graph} = \phi(h_{graph}) \tag{4.18c}$$

Trong (4.18c), $\phi(\cdot)$ là hàm đầu ra, hàm này được thiết kế để phù hợp với đầu ra của bài toán. Ví dụ, với bài toán phân loại đồ thị trên nhiều lớp, $\phi(\cdot) = \mathrm{softmax}(\mathrm{MLP}(\cdot))$.

Ví dụ 4.5: Với đồ thị gốc được cho như Hình 4.6 và ma trận $\hat{\mathbf{X}}$ như bên dưới. Hàm AGG_{graph} là $\text{Mean}(\cdot)$, ϕ là một linear transform với $\mathbf{W}_y$ và b_y. Hãy xác định lớp đầu ra của đồ thị đã cho.

$$\hat{\mathbf{X}} = \begin{bmatrix} 0.50 & 0.50 & 0.45 & 0.45 \\ 0.40 & 0.40 & 0.35 & 0.30 \\ 0.15 & 0.15 & 0.15 & 0.10 \\ 0.70 & 0.70 & 0.60 & 0.55 \\ 0.60 & 0.60 & 0.55 & 0.50 \end{bmatrix}$$

$$\mathbf{W}_y = \begin{bmatrix} 0.2 & 0.5 & 0.2 & 0.6 & 0.1 \\ 0.4 & 0.1 & 0.6 & 0.8 & 0.3 \\ 0.5 & 0.6 & 0.8 & 0.9 & 0.4 \end{bmatrix} \; b_y = \begin{bmatrix} 0.7 \\ 0.4 \\ 0.3 \end{bmatrix}$$

Lời giải:

$$h_{graph} = \text{Mean}(\hat{\mathbf{X}}) = \text{Mean}\left(\begin{bmatrix} 0.50 & 0.50 & 0.45 & 0.45 \\ 0.40 & 0.40 & 0.35 & 0.30 \\ 0.15 & 0.15 & 0.15 & 0.10 \\ 0.70 & 0.70 & 0.60 & 0.55 \\ 0.60 & 0.60 & 0.55 & 0.50 \end{bmatrix} \right) = \begin{bmatrix} 0.4750 \\ 0.3625 \\ 0.1375 \\ 0.6375 \\ 0.5625 \end{bmatrix}$$

$$\mathbf{W}_y h_{graph} + b_y = \begin{bmatrix} 0.2 & 0.5 & 0.2 & 0.6 & 0.1 \\ 0.4 & 0.1 & 0.6 & 0.8 & 0.3 \\ 0.5 & 0.6 & 0.8 & 0.9 & 0.4 \end{bmatrix} \begin{bmatrix} 0.4750 \\ 0.3625 \\ 0.1375 \\ 0.6375 \\ 0.5625 \end{bmatrix} + \begin{bmatrix} 0.7 \\ 0.4 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 1.4425 \\ 1.3875 \\ 1.6638 \end{bmatrix}$$

$$
\begin{aligned}
\hat{y}_{graph} &= \text{argmax}_y(\text{softmax}(\mathbf{W}_y h_{graph} + b_y)) \\
&= \text{argmax}_y \left(\text{softmax} \left(\begin{bmatrix} 1.4425 \\ 1.3875 \\ 1.6638 \end{bmatrix} \right) \right) \\
&= \text{argmax}_y \left(\begin{bmatrix} 0.3131 \\ 0.2963 \\ 0.3906 \end{bmatrix} \right) = 2
\end{aligned}
$$

4.4.2 Các mô hình dự đoán trên đỉnh

Bên cạnh các bài toán dự đoán trên toàn đồ thị, thực tế chúng ta cũng có rất nhiều bài toán được mô hình hóa bằng một đồ thị rất lớn, và trên đó, các mô hình được áp dụng để dự đoán phân lớp hoặc đặc trưng/tính chất cho từng đỉnh. Các bài toán có thể kể đến như phân loại người mua hàng tiềm năng trên đồ thị quan hệ giữa người dùng và những sản phẩm họ đã mua; Tìm ra tài khoản giả mạo trên đồ thị thể hiện mối liên kết giữa các tài khoản trên mạng xã hội; Dự đoán tính chất của một protein dựa vào những protein tương tự;... Đối với dạng bài toán này, khung mô hình cũng tương tự với bài toán dự đoán trên toàn đồ thị, điểm khác biệt duy nhất là bài toán này không cần phải tính toán véc-tơ đại diện cho đồ thị. Công thức (4.19) thể hiện khung mô hình cho bài toán này.

$$\hat{\mathbf{X}} = \text{GNN}(\mathbf{X}, \mathbf{A}) \tag{4.19a}$$

$$\hat{y}_{node} = \phi(\hat{\mathbf{X}}) \tag{4.19b}$$

Ví dụ 4.6: Với đồ thị gốc được cho như Hình 4.6 và ma trận $\hat{\mathbf{X}}$ như bên dưới. Hàm ϕ là một linear transform với $\mathbf{W}_y$ và b_y. Hãy xác định lớp đầu ra của từng đỉnh trong đồ thị đã cho.

$$\hat{\mathbf{X}} = \begin{bmatrix} 0.50 & 0.50 & 0.45 & 0.45 \\ 0.40 & 0.40 & 0.35 & 0.30 \\ 0.15 & 0.15 & 0.15 & 0.10 \\ 0.70 & 0.70 & 0.60 & 0.55 \\ 0.60 & 0.60 & 0.55 & 0.50 \end{bmatrix}$$

$$\mathbf{W}_y = \begin{bmatrix} 0.2 & 0.5 & 0.2 & 0.6 & 0.1 \\ 0.4 & 0.1 & 0.6 & 0.8 & 0.3 \\ 0.5 & 0.6 & 0.8 & 0.9 & 0.4 \end{bmatrix} b_y = \begin{bmatrix} 0.4 \\ 0.7 \\ 0.3 \end{bmatrix}$$

Lời giải:

$$\begin{aligned} \mathbf{W}_y\hat{\mathbf{X}} + b_y &= \begin{bmatrix} 0.2 & 0.5 & 0.2 & 0.6 & 0.1 \\ 0.4 & 0.1 & 0.6 & 0.8 & 0.3 \\ 0.5 & 0.6 & 0.8 & 0.9 & 0.4 \end{bmatrix} \begin{bmatrix} 0.50 & 0.50 & 0.45 & 0.45 \\ 0.40 & 0.40 & 0.35 & 0.30 \\ 0.15 & 0.15 & 0.15 & 0.10 \\ 0.70 & 0.70 & 0.60 & 0.55 \\ 0.60 & 0.60 & 0.55 & 0.50 \end{bmatrix} + \begin{bmatrix} 0.4 \\ 0.7 \\ 0.3 \end{bmatrix} \\ &= \begin{bmatrix} 1.21 & 1.21 & 1.11 & 1.04 \\ 1.77 & 1.77 & 1.65 & 1.56 \\ 1.78 & 1.78 & 1.62 & 1.48 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} \hat{y}_{node} &= \text{argmax}_y(\text{softmax}(\mathbf{W}_y\hat{\mathbf{X}} + b_y)) \\ &= \text{argmax}_y \left(\text{softmax} \left(\begin{bmatrix} 1.21 & 1.21 & 1.11 & 1.04 \\ 1.77 & 1.77 & 1.65 & 1.56 \\ 1.78 & 1.78 & 1.62 & 1.48 \end{bmatrix} \right) \right) \\ &= \text{argmax}_y \left(\begin{bmatrix} 0.2213 & 0.2213 & 0.2287 & 0.2361 \\ 0.3874 & 0.3874 & 0.3924 & 0.3972 \\ 0.3913 & 0.3913 & 0.3789 & 0.3667 \end{bmatrix} \right) = [2, 2, 1, 1] \end{aligned}$$

4.4.3 Các mô hình dự đoán trên cạnh

Ngoài hai bài toán phổ biến trên, chúng ta có một dạng bài toán khác cũng quan trọng không kém, đó chính là dự đoán cạnh và những tính chất của cạnh nối giữa hai đỉnh trong cùng một đồ thị. Bài toán dạng này thường được ứng dụng trong các sàn thương mại điện tử để dự đoán khả năng mua hàng sắp tới của người dùng; trong lĩnh vực khoa học cơ bản, có thể dùng để dự đoán liên kết giữa các nguyên tử, phân tử với nhau hay dự đoán mối liên hệ giữa các chất hóa học với nhau;... Không những thế, bài toán này còn là tiền đề cho nhiều bài toán phía sau có thể thực thi được. Ví dụ như bài toán gom cụm người dùng dựa vào mối liên hệ hay gom cụm các chất hóa học giống nhau... Bài toán này ngoài việc cần thiết các mô hình trích xuất đặc trưng đỉnh tốt, nó còn yêu cầu chúng ta thiết kế chiến lược huấn luyện/kiểm thử phù hợp (Chia tập dataset theo cạnh nhưng vẫn đảm bảo một số tính chất của đồ thị như liên thông...). Bạn đọc có thể đọc thêm tại (Zhang & Chen, 2018). Khung mô hình cho bài toán này không có quá nhiều khác biệt ngoại trừ việc độ phức tạp sẽ tăng trong trường hợp đồ thị ban đầu thưa (sparse graphs). Công thức 4.20 mô tả khung mô hình để giải quyết bài toán này.

$$\hat{\mathbf{X}} = \mathrm{GNN}(\mathbf{X}, \mathbf{A}) \tag{4.20a}$$

$$\hat{y}_{edge} = \{\phi(\hat{x}_u, \hat{x}_v), (u, v) \notin \mathcal{E}\} \tag{4.20b}$$

Trong (4.20b), $\phi(\cdot)$ là hàm tính toán đầu ra cho cạnh giữa hai đỉnh.

Ví dụ 4.7: Với đồ thị gốc được cho như Hình 4.6 và ma trận $\hat{\mathbf{X}}$ như bên dưới. Hàm ϕ là một linear transform được định nghĩa là $\phi(x, y) = \sigma(\mathbf{W}_y \cdot (x + y) + b_y)$ với $\mathbf{W}_y$ và b_y được cho như bên dưới. Hãy xác định có cạnh tồn tại giữa các đỉnh chưa có cạnh hay không với xác suất tối thiểu là 0.5.

$$\hat{\mathbf{X}} = \begin{bmatrix} 0.50 & 0.50 & 0.45 & 0.45 \\ 0.40 & 0.40 & 0.35 & 0.30 \\ 0.15 & 0.15 & 0.15 & 0.10 \\ 0.70 & 0.70 & 0.60 & 0.55 \\ 0.60 & 0.60 & 0.55 & 0.50 \end{bmatrix}$$

$$\mathbf{W}_y = \begin{bmatrix} 0.2 & 0.5 & 0.2 & 0.6 & 0.1 \end{bmatrix} b_y = \begin{bmatrix} 0.1 \end{bmatrix}$$

Lời giải:

$$\begin{aligned} \hat{y}_{0,3} &= \sigma(\mathbf{W}_y \cdot (\hat{x}_0 + \hat{x}_3) + b_y) \\ &= \sigma\left(\begin{bmatrix} 0.2 & 0.5 & 0.2 & 0.6 & 0.1 \end{bmatrix} \cdot \left(\begin{bmatrix} 0.50 \\ 0.40 \\ 0.15 \\ 0.70 \\ 0.60 \end{bmatrix} + \begin{bmatrix} 0.45 \\ 0.30 \\ 0.10 \\ 0.55 \\ 0.50 \end{bmatrix} \right) + \begin{bmatrix} 0.1 \end{bmatrix} \right) \\ &= \sigma(1.46) = 0.81 \end{aligned}$$

$\implies$ Tồn tại cạnh giữa đỉnh 0 và đỉnh 3.

$$\begin{aligned} \hat{y}_{1,3} &= \sigma(\mathbf{W}_y \cdot (\hat{x}_1 + \hat{x}_3) + b_y) \\ &= \sigma\left(\begin{bmatrix} 0.2 & 0.5 & 0.2 & 0.6 & 0.1 \end{bmatrix} \cdot \left(\begin{bmatrix} 0.50 \\ 0.40 \\ 0.15 \\ 0.70 \\ 0.60 \end{bmatrix} + \begin{bmatrix} 0.45 \\ 0.30 \\ 0.10 \\ 0.55 \\ 0.50 \end{bmatrix} \right) + \begin{bmatrix} 0.1 \end{bmatrix} \right) \\ &= \sigma(1.46) = 0.81 \end{aligned}$$

$\implies$ Tồn tại cạnh giữa đỉnh 1 và đỉnh 3.

4.5 Kết chương

Đồ thị (graph) là một dạng dữ liệu thường gặp trong thực tế. Tuy nhiên các mô hình học máy kinh điển như CNN hay RNN chưa xử lý dạng dữ liệu này vì các mô hình này khó biểu diễn được *topology*, tức mối quan hệ sắp xếp giữa đỉnh và cạnh, của đồ thị.

GNN được giới thiệu để giải quyết vấn đề trên. Ý tưởng chính của GNN vẫn là cố gắng đưa ra các dạng học biểu diễn của đồ thị thông qua chiến lược aggregation để từ đó giải quyết các bài toán downstream trên loại dữ liệu này. Các mô hình GNN đã kế thừa và phát triển các kỹ thuật tiên tiến được giới thiệu từ các mô hình học sâu khác như cơ chế tích chập, cơ chế gating và attention như các chương trước đã có đề cập. Tuy nhiên, khi vận dụng vào GNN, các kỹ thuật này đã được điều chỉnh để phù hợp với quan hệ trên đồ thị, đặc biệt là với các khái niệm Random Walk hay Multihop đặc thù của dạng dữ liệu này. Trong khi dữ liệu chuỗi đã thu được nhiều kết quả thực tế ấn tượng, dữ liệu đồ thị vẫn đang mở ra nhiều cơ hội và thách thức cho các nghiên cứu và ứng dụng trong lĩnh vực này.

4.6 Bài tập

Bài tập 4.6.1. Hãy tìm thêm 5 hàm kết hợp ngoài những hàm cơ bản (sum, mean, min, max) thỏa mãn tính chất hoán vị bất biến.

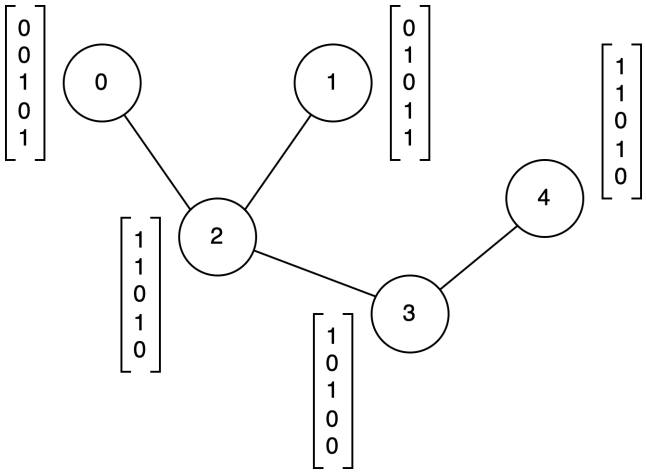
Bài tập 4.6.2. Biểu diễn đồ thị đầu vào bằng các ma trận $(\mathbf{X}, \mathbf{A})$ rất thuận tiện cho việc tính toán. Tuy nhiên, cách biểu diễn này vẫn có nhược điểm khi thao tác với những đồ thị lớn. Hãy đề xuất tối thiểu 2 cách để giải quyết vấn đề này.

Bài tập 4.6.3. Hãy chứng minh/giải thích công thức được dùng trong mã hóa cạnh như bên dưới đây.

$$P(v|z_u) = \frac{\exp(z_u^\top z_v)}{\sum_{n \in \mathcal{V} \setminus \{u\}} \exp(z_u^\top z_n)}$$

Bài tập 4.6.4. Trong công thức (4.7b), hàm AGG(·) phải sử dụng $m_u^{(l)}$ để tạo ra véc-tơ đặc trưng cho đỉnh u ở tầng l, $h_u^{(l)}$. Hãy nêu tối thiểu hai bài toán áp dụng được hiệu quả cơ chế này và tối thiểu hai bài toán không cần dùng đến $m_u^{(l)}$ nhưng vẫn đạt được kết quả tốt. Hãy giải thích.

Bài tập 4.6.5. Cho đồ thị và các véc-tơ z_u được cho như Hình 4.7. Hãy tính đặc trưng của các đỉnh dùng Stochastic Gradient Descent với 3 đỉnh bắt đầu và bước đi ngẫu nhiên được chọn lần lượt là: $0 \to [2 \to 1]$, $1 \to [2 \to 1]$ và $2 \to [3 \to 4]$ và hệ số học $\eta = 0.8$.



Hình 4.7: Đồ thị mẫu dành cho các bài tập 4.5.5 và 4.5.9

Bài tập 4.6.6. Hãy chứng minh tại sao những đỉnh có cấu trúc đồ thị tính toán k-hop giống nhau sẽ có véc-tơ biểu diễn giống nhau ở lớp thứ k.

Bài tập 4.6.7. Hãy chứng minh khi áp dụng GCN nhiều lớp sẽ dẫn đến việc các véc-tơ biểu diễn các đỉnh sẽ càng lúc càng gần nhau và khó phân biệt.

Bài tập 4.6.8. Hãy chứng minh hàm AGG(·) ở công thức (4.15) thỏa mãn tính chất hoán vị bất biến.

Bài tập 4.6.9. Cho đồ thị và các véc-tơ đặc trưng ban đầu của đỉnh như Hình 4.7. Hãy tính các véc-tơ đặc trưng đỉnh sau khi qua L lớp R.

- R là GCN với AGG = ReLU(Mean($\cdot$)), $L = 1, 2, 3$ và

$$\mathbf{W}^{(1)} = \begin{bmatrix} 0.20 & 0.60 & 0.50 & 0.30 & 0.10 \\ 0.10 & 0.30 & 0.40 & 0.40 & 0.70 \\ 0.70 & 0.05 & 0.05 & 0.10 & 0.10 \end{bmatrix} \quad \mathbf{W}^{(2)} = \begin{bmatrix} 0.20 & 0.30 & 0.80 \\ 0.60 & 0.50 & 0.20 \\ 0.80 & 0.10 & 0.10 \end{bmatrix}$$

$$\mathbf{W}^{(3)} = \begin{bmatrix} 0.80 & 0.70 & 0.20 \\ 0.40 & 0.50 & 0.80 \\ 0.20 & 0.90 & 0.90 \end{bmatrix}$$

- R là GraphSAGE với AGG = ReLU(Mean($\cdot$)) và $L = 1, 2$
  Ma trận $\mathbf{W}_m^{(1)}$ và $\mathbf{W}_m^{(2)}$ được sử dụng lại là ma trận $\mathbf{W}^{(1)}$ và $\mathbf{W}^{(2)}$ như ở phần GCN bên trên.

$$\mathbf{W}_h^{(1)} = \begin{bmatrix} 0.10 & 0.20 & 0.10 & 0.20 & 0.30 & 0.70 \\ 0.20 & 0.50 & 0.10 & 0.40 & 0.90 & 0.50 \\ 0.50 & 0.90 & 0.40 & 0.10 & 0.10 & 0.30 \end{bmatrix}$$

$$\mathbf{W}_h^{(2)} = \begin{bmatrix} 0.20 & 0.30 & 0.10 & 0.10 & 0.30 & 0.40 \\ 0.10 & 0.60 & 0.20 & 0.50 & 0.60 & 0.80 \\ 0.60 & 0.70 & 0.30 & 0.30 & 0.40 & 0.20 \end{bmatrix}$$

- R là GAT với hàm tính attention là *Luong attention* với $\mathbf{W}_\alpha$ là ma trận trọng số và $L = 1, 2$
  Ma trận $\mathbf{W}^{(1)}$ và $\mathbf{W}^{(2)}$ được sử dụng lại tương tự như ở phần GCN bên trên.

$$\mathbf{W}_\alpha^{(1)} = \begin{bmatrix} 0.10 & 0.40 & 0.50 \\ 0.60 & 0.20 & 0.20 \\ 0.30 & 0.40 & 0.30 \end{bmatrix} \quad \mathbf{W}_\alpha^{(2)} = \begin{bmatrix} 0.20 & 0.40 & 0.30 \\ 0.50 & 0.10 & 0.20 \\ 0.20 & 0.30 & 0.40 \end{bmatrix}$$

Bài tập 4.6.10. Với đầu ra là đặc trưng ở lớp cuối cùng của ba mô hình

GCN, GraphSAGE và GAT như đã tính toán ở bài **4.5.9**, hãy tính toán và so sánh đầu ra khi sử dụng các biến thể GNN khác nhau.

1. Phân loại toàn bộ đồ thị bằng một tầng linear transformation với $\mathbf{W}_y$ và b_y như bên dưới.

$$\mathbf{W}_y = \begin{bmatrix} 0.80 & 0.30 & 0.10 \\ 0.50 & 0.70 & 0.20 \\ 0.30 & 0.40 & 0.80 \end{bmatrix} \; b_y = \begin{bmatrix} 0.30 \\ 0.40 \\ 0.20 \end{bmatrix}$$

2. Phân loại từng đỉnh trong đồ thị bằng một tầng linear transformation với $\mathbf{W}_y$ và b_y như bên trên.

Bài tập 4.6.11. (Bonus) Hãy chứng minh việc lặp lại k lần phép tính gated ouput ở Gated Graph Neural Networks khi áp dụng vào Graph Attention Networks sẽ cho kết quả xấp xỉ như khi chúng ta thực hiện k-hop attention trên Graph Attention Networks.

Chương 5

Các mô hình Generative Adversarial Networks

5.1 Giới thiệu

Theo phương thức học, các thuật toán học máy thường được chia làm 2 nhóm chính là supervised learning (học có giám sát) và unsupervised learning (học không có giám sát). Supervised learning bao gồm các thuật toán dự đoán đầu ra của một dữ liệu mới dựa trên cặp (dữ liệu, đầu ra) đã biết trrước từ tập dữ liệu. Loại giải thuật này được ứng dụng trong các bài toán phân loại và hồi quy. Ngược lại, unsupervised learning là một nhóm thuật toán hay phương pháp kỹ thuật cho phép máy tự học hỏi và tìm ra một mô hình hay cấu trúc nào đó ẩn trong một bộ dữ liệu không được gắn nhãn trước. Điều này đồng nghĩa với việc chúng ta chỉ có bộ dữ liệu đầu vào và hoàn toàn không biết đầu ra của nó là gì.

Trong supervised learning, có 2 khó khăn thường gặp khi mở rộng quy mô bài toán đó là việc quản lý hoặc chỉnh sửa thủ công dữ liệu được gắn nhãn mới rất tốn kém và mất nhiều thời gian; và việc thu thập dữ liệu tự động cũng gặp nhiều khó khăn và phức tạp. Để giải quyết được vấn đề này, việc *làm giàu dữ liệu* (data augmentation) là phương pháp phổ biến

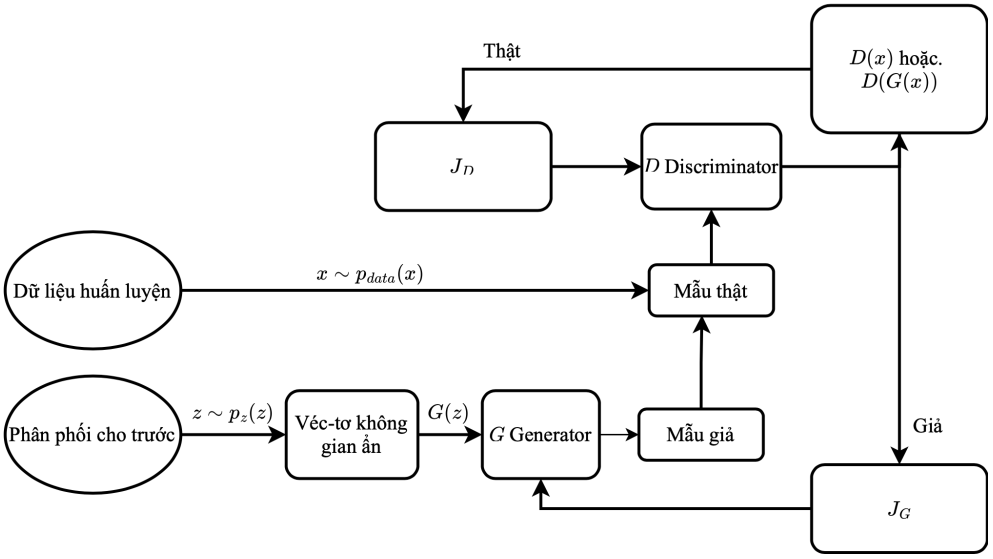
được sử dụng nhiều nhất. Áp dụng phương pháp này giúp làm giảm sai số tổng quát và tăng khả năng "học hỏi" của mô hình. Việc tăng dữ liệu được thực hiện bằng cách tự động tạo ra các mẫu mới và được chấp nhận từ tập dữ liệu huấn luyện ban đầu. Đối với lĩnh vực xử lý ảnh, phương pháp này được thực hiện thông qua việc áp dụng các toán tử biến đổi ảnh chẳng hạn như xoay, cắt xén, thu phóng và các phép biến đổi đơn giản khác trên ảnh. Tuy nhiên, phương pháp này cũng có điểm yếu là chỉ có thể thu thập dữ liệu với thông tin vô cùng hạn chế và bị giới hạn. Chính vì vậy, các nhà nghiên cứu đã đề xuất ra một phương pháp tăng dữ liệu *tiên tiến nhất* (state-of-the-art) đó là tạo ra các mẫu chất lượng cao thông qua các *mô hình sinh* (generative models).

Thông thường, các mô hình sinh hoạt động dựa trên chuỗi Markov, *ước tính hợp lý tối đa* (Maximum Likelihood Estimation, hay MLE) và suy luận xấp xỉ. *Máy Boltzmann hạn chế* (Restricted Boltzmann Machine) (Rumelhart and McClelland, 1987) và các mô hình đã phát triển của nó như *mạng niềm tin sâu* (Deep Belief Network) (Hinton et al., 2006) và *Máy Boltzmann sâu* (Deep Boltzmann Machine) (Salakhutdinov and Hinton, 2009) cũng đều dựa trên MLE, tuy nhiên các mô hình này vẫn chưa cho thấy khả năng khái quát hóa tốt khi áp dụng với số lượng lớn dữ liệu thực tế. Do đó, mạng GAN (hay Generative Adversarial Network) đã được đề xuất như một ý tưởng cho việc học không giám sát (Goodfellow et al., 2014) bởi Ian Goodfellow vào năm 2014. Trong đó, Yann LeCun, giám đốc nghiên cứu của IBC tại Facebook, đã giới thiệu việc *huấn luyện đối nghịch* (adversarial learning) trong mô hình này là ý tưởng thú vị nhất trong mười năm qua về học máy.

5.2 Kiến trúc mạng GAN

Như đã giới thiệu ở trên, mô hình GAN là một mạng nơ-ron tuân theo phương thức học không có giám sát. Mô hình này có thể tạo ra các mẫu rất giống với phân phối dữ liệu thực thông qua việc tự động học các quy tắc và cấu trúc tiềm ẩn của các mẫu dữ liệu hiện có.

Kiến trúc của mạng GAN cơ bản có cách tiếp cận giống với lý thuyết trò chơi Minimax. GAN thường chứa hai mạng nơ-ron để huấn luyện và cạnh tranh lẫn nhau gọi là *generator* và *discriminator*. Lý do có từ "đối nghịch" trong tên của GAN là bởi vì hai mạng này có xung đột về hàm mục tiêu trong suốt quá trình huấn luyện. Hai mạng này có thể được ví như lần lượt kẻ giả mạo và cảnh sát. Trong đó, kẻ giả mạo (generator) sẽ cố gắng làm giả và tạo ra một đối tượng tương tự như thật bằng cách học các thủ thuật mới nhất để đánh lừa cảnh sát (discriminator). Ngược lại, cảnh sát phải liên tục cập nhật thông tin để phát hiện ra đâu là đối tượng thực đâu là đối tượng được tạo ra bởi kẻ giả mạo. Từ đó, hai mạng này cứ liên tục cập nhật kiến thức lẫn nhau để đạt được mục tiêu riêng của mình. Quá trình huấn luyện này tiếp tục cho đến khi discriminator không thể phân biệt được đâu là thật đâu là giả. Hay nói cách khác, generator khi đó đã có thể tạo ra các mẫu dữ liệu có cấu trúc gần tương đương với tập dữ liệu thực tế, và có thể được sử dụng như một phương pháp làm giàu dữ liệu.



Hình 5.1: Kiến trúc mạng Vanilla GAN - mô hình mạng GAN đầu tiên

Kiến trúc tổng quát của mạng GAN ban đầu (hay còn gọi là Vanilla GAN) được minh họa trong Hình 5.1. x và $G(z)$ lần lượt là mẫu thật

trong dữ liệu huấn luyện và mẫu giả được khối generator G tạo nên từ phân phối cho trước z. Sau đó, khối discriminator D sẽ có nhiệm vụ đánh giá xác suất dữ liệu đầu vào là thật hoặc giả và lan truyền ngược để cập nhật lại trọng số cho G và D.

Cụ thể hơn, đầu tiên, generator sẽ nhận véc-tơ nhiễu $z \sim p_z(z)$ (một véc-tơ ngẫu nhiên có phân phối đồng nhất hoặc phân phối Gaussian cho trước) có độ dài cố định làm đầu vào. Sau đó, generator tổng hợp nên dữ liệu mới $G(z)$ từ các phân phối chuẩn của mẫu thật $x \sim p_{data}(x)$. Có thể nói, việc generator tạo ra một hình ảnh không yêu cầu đầu vào ban đầu của một hình ảnh cụ thể mà hoàn toàn là một véc-tơ các giá trị ngẫu nhiên. Sau khi huấn luyện, các điểm của véc-tơ nhiều chiều này được khớp với các điểm trong miền bài toán, tạo thành một biểu diễn nén của phân phối dữ liệu. Không gian véc-tơ này được gọi là *không gian tiềm ẩn* (latent space) hoặc không gian véc-tơ bao gồm các *biến tiềm ẩn* (latent variables). Các biến tiềm ẩn bao gồm các biến quan trọng nhưng không thể quan sát được trực tiếp cho một miền của bài toán. Các mô hình học máy sẽ học từ những không gian tiềm ẩn đó để tạo ra một loạt các dữ liệu mới với các đặc điểm gần tương tự như các mẫu thật.

Discriminator được huấn luyện để tối đa hóa khả năng gán nhãn chính xác cho dữ liệu thật và giả. Do đó, nó sẽ hoạt động như một bộ phân loại nhị phân và phân biệt các mẫu $G(z)$ giả với các mẫu x thật. Hay nói cách khác, nếu đầu vào được tạo thành từ dữ liệu thật, discriminator sẽ phân loại nó là dữ liệu thật và trả về một giá trị số gần bằng 1. Ngược lại, nếu đầu vào bao gồm dữ liệu được tạo bởi genertor, discriminator sẽ phân loại nó là dữ liệu giả và trả về giá trị số gần bằng 0.

Mô hình mạng nơ-ron sử dụng làm generator và discriminator rất đa dạng, có thể là những *mạng nơ-ron tích chập* (convolutional neural networks), những *mạng nơ-ron hồi quy* (recurrent neural networks) hay cũng có thể là những autoencoder, miễn là chúng có những tham số có thể học được. Do đó, discriminator yêu cầu *hàm mất mát* (loss function) J_D và generator yêu cầu hàm mất mát J_G để cập nhật lại tham số trong mạng

(Hình 5). Trong đó, generator chỉ cập nhật các thông số của nó thông qua các tín hiệu lan truyền ngược của đầu ra giả, còn discriminator sẽ nhận được nhiều thông tin hơn và cập nhật lại thông số bằng cách sử dụng cả đầu ra giả và đầu ra thật.

Việc tối ưu hóa hàm mất mát của mạng GAN giống với chiến lược trong trò chơi Minimax hai người chơi, theo đó mỗi người chơi sẽ cố gắng tìm cách giảm mất mát của bản thân mình hoặc tăng chi phí của người chơi kia để giành được chiến thắng. Trong GAN, generator và discriminator là đại diện cho hai người chơi trong trò chơi trên. Mô hình generator tạo ra dữ liệu giả tốt hơn sẽ làm cho discriminator khó phân biệt hơn và ngược lại, khi discriminator phân biệt tốt hơn thì generator cần phải tạo ra ảnh giống thật hơn để qua mặt discriminator. Từ đó, chúng sẽ lần lượt cập nhật lại trọng số trong mạng của mình. Cụ thể:

- Nếu $X = x \sim p_{data}(x)$, ta cần huấn luyện discriminator sao cho tối đa hóa được xác suất gán đúng nhãn cho các mẫu có sẵn trong tập huấn luyện. Hàm mục tiêu khi đó là:

$$D(x) \to 1 \iff \max_D E_{x \sim p_{data}(x)}[log(D(x))] \tag{5.1}$$

- Nếu $X = G(z)$:
  - Ta cần huấn luyện discriminator để không bị đánh lừa bởi các mẫu tạo ra bởi generator. Khi đó, hàm mục tiêu là:

$$D(x) \to 0 \iff \max_D E_{z \sim p_z(z)}[log(1 - D(G(z)))] \tag{5.2}$$

  - Đồng thời, ta cũng huấn luyện generator có khả năng tạo ra các mẫu càng gần giống với dữ liệu thật càng tốt. Khi đó, hàm mục tiêu là:

$$D(x) \to 1 \iff \min_G E_{x \sim p_{data}(x)}[log(D(x))] \tag{5.3}$$

Tổng hợp lại, ta có hàm mất mát của mạng GAN được định nghĩa theo công thức tổng quát sau:

$$C(G) = \min_{G} \max_{D} E_{x \sim p_{data}(x)}[log(D(x))] + E_{z \sim p_z(z)}[log(1 - D(G(z)))] \tag{5.4}$$

Trong đó, E là ký hiệu của kỳ vọng toán học, p_{data} đại diện cho phân phối của dữ liệu và p_z là đại diện cho phân phối của dữ liệu nhiễu ngẫu nhiên.

Ví dụ 5.1 Tại một thời điểm trong quá trình huấn luyện một mạng GAN đơn giản, gọi r và z lần lượt là véc-tơ mẫu thật và véc-tơ không gian tiềm ẩn được sinh ra một cách ngẫu nhiên làm đầu vào huấn luyện cho khối generator. Giả sử đầu ra của khối discriminator với mẫu thật là $D(r) = \begin{bmatrix} 0.8 & 0.75 & 0.65 & 0.7 \end{bmatrix}$ và đầu ra của mẫu giả là $D(G(z)) = \begin{bmatrix} 0.5 & 0.45 & 0.6 & 0.7 \end{bmatrix}$.

Khi đó, giá trị của hàm mất mát của mạng GAN tính trên:

- Discriminator:

$$\begin{aligned}
L_D &= L_{discriminate\ real}(pt\ 5.1) + L_{discriminate\ fake}(pt\ 5.2) \\
&= Cross_entropy(\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 0.8 & 0.75 & 0.65 & 0.7 \end{bmatrix}) \\
&\quad + Cross_entropy(\begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0.5 & 0.45 & 0.6 & 0.7 \end{bmatrix}) \\
&= -\frac{1}{4}(\log(0.8) + \log(0.75) + \log(0.65) + \log(0.7)) \\
&\quad - \frac{1}{4}(\log(1 - 0.5) + \log(1 - 0.45) + \log(1 - 0.6) + \log(1 - 0.7)) \\
&= 0.32457072 + 0.8528117 = 1.1773824
\end{aligned}$$

- Generator:

$$
\begin{aligned}
L_G &= L_{generate\ fake}(pt\ 5.3) \\
&= Cross_entropy(\begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}, \begin{bmatrix}0.5 & 0.45 & 0.6 & 0.7\end{bmatrix}) \\
&= -\frac{1}{4}(\log(0.5) + \log(0.45) + \log(0.6) + \log(0.7)) \\
&= 0.5897887
\end{aligned}
$$

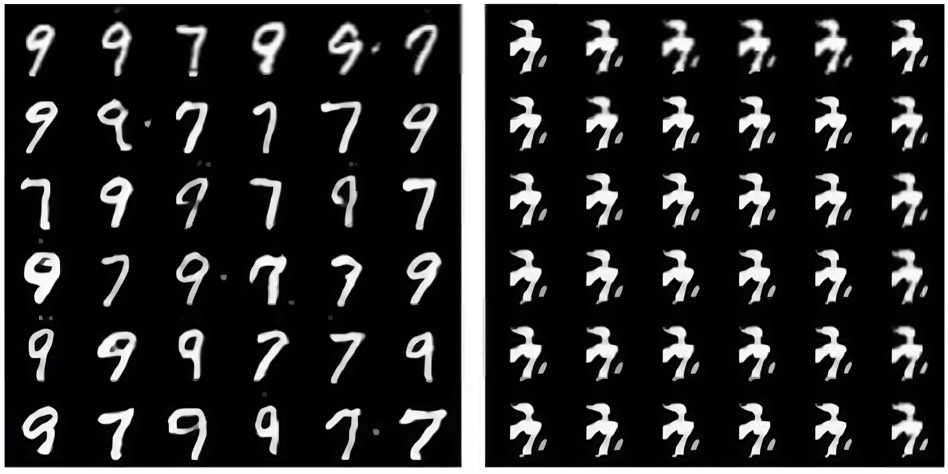
5.3 Hạn chế của mạng GAN

Việc huấn luyện các mô hình mạng GAN rất phức tạp, đòi hỏi phải có sự cân bằng giữa hiệu suất của cả generator và discriminator. Có hai vấn đề thường gặp phải trong quá trình huấn luyện như sau:

- **Hiện tượng Mode Collapse** Đây là tình huống xảy ra khi generator chỉ có thể tổng hợp một tập con nhỏ của phân phối dữ liệu hoàn chỉnh bởi vì quá trình huấn luyện không cho phép tổng hợp phong phú các biến thể của phân phối gốc. Ví dụ minh họa cho tình huống này được thể hiện ở Hình 5.2. Hình bên trái là tập hợp dữ liệu lấy từ tập MNIST, bao gồm các hình ảnh của các chữ số viết tay từ 0 đến 9. Tuy nhiên vì không thể nắm bắt được đầy đủ sự đa dạng của tập hợp gốc nên dữ liệu dùng để huấn luyện đã bị "collapse" khi chỉ còn lại hai loại chữ số. Điều này dẫn đến kết quả ở hình bên phải là sau quá trình huấn luyện, khối generator chỉ có thể tạo ra một đầu ra duy nhất với bất ký nhiễu đầu vào nào và "tình cờ" discriminator cho rằng đây là dữ liệu thật nhất. Khi đó, generator sẽ cứ tạo đi tạo lại đầu ra đó mà chẳng có cập nhật lại để cải thiện gì.

- **Hiện tượng Vanishing Gradient** Vấn đề này bắt nguồn khi discriminator hoặc generator trở nên quá áp đảo, gây ra sự mất cân bằng không thể đảo ngược và dẫn đến bế tắc trong quá trình huấn luyện. Một ví dụ là khi discriminator có thể dễ dàng phân biệt

được mẫu sinh ra và mẫu thật (tức là loss của D gần bằng không), thì sự trì trệ trong việc đi ngược lại đạo hàm (gradient descent) ở generator lại làm loss của G không thể hội tụ được và ngăn cản quá trình học và cải thiện hiệu suất của bộ này. Hiện tượng này có thể được khắc phục bằng cách sử dụng các hàm chi phí thích hợp cho phép thu được tỷ lệ học phù hợp ứng với mỗi epoch để hai mạng đối nghịch có thể cải thiện được hiệu suất của nó kịp thời trước khi xảy ra bế tắc.



Hình 5.2: Minh họa cho hiện tượng Mode Collapse khi huấn luyện mạng GAN

5.4 Các cột mốc quan trọng

5.4.1 Mô hình GAN gốc

Nghiên cứu gốc của Ian Goodfellow (Goodfellow et al., 2014) có thể xem như là tài liệu bắt buộc phải đọc đối với bất kỳ ai mong muốn tìm hiểu về GANs. Bài báo này định nghĩa framework của GAN và thảo luận về hàm mất mát **non-saturating**. Ngoài ra, bài báo này cũng đưa ra kết quả chứng minh hiệu quả của GAN bằng thực nghiệm trên các bộ dữ liệu hình ảnh phổ biến như MNIST, TFD và CIFAR-10.

Các mệnh đề và định lý quan trọng được phát biểu trong bài báo này gồm có:

1. **Mệnh đề 1:** Với hàm G của generator cố định, bài toán tối ưu hóa hàm D của discriminator sẽ có nghiệm là:

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \tag{5.5}$$

   Nhắc lại một chút ở trên, mục tiêu huấn luyện của discriminator D, với bất kỳ generator G cho trước, là tối đa hóa tổng $V(D, G)$ ở Công thức 5.4. Công thức này có thể được viết lại tương đương với biểu thức tích phân như sau:

$$\begin{aligned} V(D,G) &= \int_x p_{data}(x)\log(D(x))\,dx + \int_z p_z(z)\log(1 - D(G(z)))\,dz \\ &= \int_x \left(p_{data}(x)\log(D(x)) + p_G(x)\log(1 - D(x))\right)\,dx \end{aligned} \tag{5.6}$$

   Nếu đặt $p_{data}(x)$, $p_G(x)$ và $D(x)$ lần lượt là a, b và u, ta có thể thấy với mọi $(a, b) \in \mathbb{R}\backslash\{(0, 0)\}$ cực trị của hàm số $f(u) = a\log(u) + b\log(1 - u)$ xảy ra tại điểm:

$$f'(u) = 0 \Leftrightarrow \frac{a}{u} - \frac{b}{1-u} = 0 \Leftrightarrow u = \frac{a}{a+b}$$

   Sau đó, thay u vừa tìm được vào đạo hàm bậc 2 của f, ta có:

$$f''(u) = -\frac{a}{(a+b)^2} - \frac{b}{(1 - \frac{a}{a+b})^2} < 0$$

   Từ đó chứng minh rằng $\frac{a}{a+b}$ là cực đại của hàm f. Do vậy, $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$ là điểm cực đại của hàm mục tiêu 5.4 nêu trên.

2. **Định lý 1:** Global minimum của hàm $C(G)$ trong Công thức 5.4 đạt được khi và chỉ khi $p_g = p_{data}$. Khi đó giá trị của $C(G)$ là

$-\log(4)$.

3. **Mệnh đề 2:** Nếu G và D đủ năng lực cập nhật, và ở mỗi bước của thuật toán GAN, discriminator được tối ưu với G cho trước, và p_g được cập nhật để cải thiện $C(G)$ trong Công thức 5.4 thì $p_g \rightarrow p_{data}$.

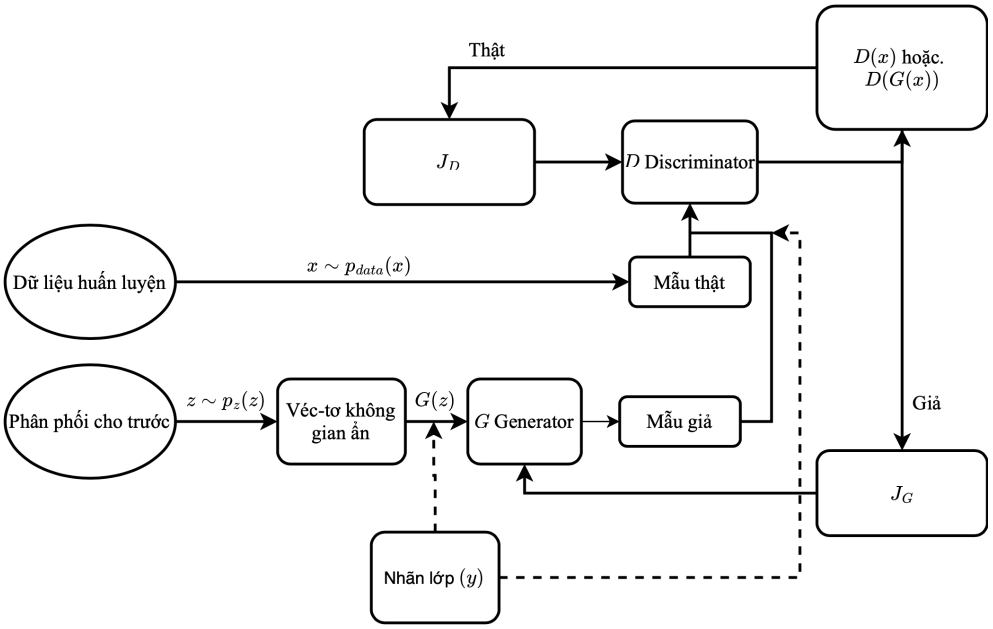
5.4.2 Mô hình Conditional GANs

Mô hình *Conditional Generative Adversarial Networks* (cGAN) (Mirza and Osindero, 2014) là một phần mở rộng của framework Vanilla GAN ban đầu được phát triển bởi Mirza và Osindero vào năm 2014. Lý do là bởi nếu tập huấn luyện chứa nhiều lớp dữ liệu, thì việc sinh ra các mẫu giả trong Vinilla GAN phụ thuộc vào véc-tơ không gian tiềm ẩn đến từ các giá trị ngẫu nhiên, điều này nằm ngoài sự kiểm soát của người dùng trong việc sinh ra các lớp dữ liệu giả. Chính vì vậy, cGAn được phát triển nhằm kết hợp thêm điều kiện sử dụng những thông tin trước đó để mô hình có khả năng tạo ra lớp dữ liệu riêng biệt. Thông qua việc thêm nhãn y làm đại diện cho lớp dữ liệu vào input của cả khối generator và discriminator vào kiến trúc của mô hình ban đầu, người dùng có thể kiểm soát được lớp dữ liệu tạo ra bởi generator cũng như thực hiện phân biệt chuẩn xác hơn cho một lớp cụ thể bằng discriminator. Hàm mất mát của mô hình này, về cơ bản cũng giống với hàm mất mát của mô hình Vanilla GAN (xem Công thức 5.4), nhưng được bổ sung các nhãn lớp y để được công thức như sau:

$$\min_G \max_D E_{x\sim p_{data}(x)}[log(D(\mathbf{x}|\mathbf{y}))] + E_{z\sim p_z(z)}[log(1 - D(G(\mathbf{z}|\mathbf{y})))] \quad (5.7)$$

Hình 5.3 thể hiện kiến trúc của một mô hình cGAN đơn giản. So sánh với kiến trúc mô hình Vanila GAN ở Hình 5.1, có thể thấy rằng cGAn chỉ khác biệt ở điểm là mô hình này sử dụng thêm nhãn lớp y ở đầu vào của cả khối generator và discriminator. Đối với khối generator, y được kết hợp với véc-tơ nhiễu đầu vào $p_z(z)$ trong một biểu diễn ẩn chung. Còn trong khối discriminator, cả hai loại đầu vào (bao gồm x và $G(z)$)

đều được kết hợp với nhãn y tương ứng để sinh ra phân phối xác suất mới có điều kiện là $D(x|y)$ hoặc $D(G(z|y))$.



Hình 5.3: Kiến trúc mạng cGAN

5.4.3 Mô hình DCGANs

Mô hình *Deep Convolutional Generative Adversarial Networks* (DCGAN) (Radford et al., 2015) được thiết kế bởi Radford và các cộng sự vào năm 2015 và được định hướng sử dụng trong các bài toán liên quan đến hình ảnh và thị giác máy tính. Lớp mạng GAN này được phát triển tập trung vào việc cải thiện tính ổn định trong quá trình huấn luyện so với mô hình mạng GAN ban đầu (vốn sử dụng perceptron đa tầng ở cả generator và discriminator). Thay vào đó, generator và discriminator trong DCGAN được lấy cảm hứng từ kiến trúc mạng tích chập (Convolutional Neural Network, hay còn được viết tắt là CNN).

Cụ thể, discriminator sẽ có kiến trúc tương tự như một mạng tích chập truyền thống vì bản chất, nhiệm vụ phân biệt của discriminator không

khác gì nhiệm vụ phân loại nhị phân cổ điển mà CNN thường sử dụng. Trong khi đó, đối với generator cần phải sử dụng một mạng tích chập đặc biệt gọi là *mạng tích chập chuyển vị* (transposed convolutional networks), hay một số nghiên cứu dùng cách gọi khác là *mạng giải chập* (deconvolutional networks). Loại mạng này tương tự như một mạng nơ-ron tích chập, tuy nhiên lại hoạt động theo một quy trình hoàn toàn ngược lại. Các mạng tích chập chuyển vị sẽ cố gắng tìm lại các đặc trưng hoặc tín hiệu đã mất mà trước đây có thể bị xem là không quan trọng đối với nhiệm vụ của mạng nơ-ron tích chập. Quá trình này giúp mô hình tạo ra hình ảnh từ các đặc trưng cấp cao do véc-tơ nhiễu ngẫu nhiên cung cấp. Ngoài ra vì mục đích của nó là mở rộng các bản đồ đặc trưng (upsampling) chứ không phải giảm chúng như trong CNN nên các tầng gộp cũng sẽ không được sử dụng trong kiến trúc mạng này.

Có thể thấy trong hình trên, khối generator trong kiến trúc mạng DCGAN sẽ xử lý véc-tơ nhiễu đầu vào ngẫu nhiên qua một loạt các lớp tích chập chuyển vị, từ đó dần tạo ra các bản đồ đặc trưng mới có kích thước lớn hơn và cuối cùng tổng hợp thành đầu ra $G(z)$ là hình ảnh với 3 kênh màu RGB và độ phân giải là 64×64 pixel.

Sau đây là ví dụ chi tiết cho 2 kiểu định nghĩa về cách tính phép toán tích chập chuyển vị được dùng trong các tầng ở khối generator.

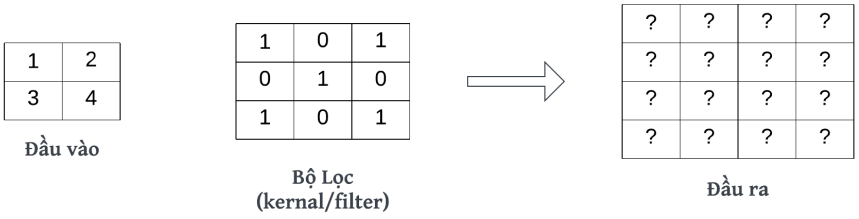
Cho dữ liệu đầu vào là ma trận hình vuông kích thước 2×2 và một bộ lọc 3×3 có giá trị lần lượt như Hình 5.4.

- **Kiểu tích chập chuyển vị 1** được định nghĩa bằng cách nhân mỗi phần tử trên dữ liệu đầu vào với bộ lọc. Ma trận kết quả có kích thước bằng với kích thước của bộ lọc và sẽ được viết vào đầu ra ở vị trí tương ứng. Nếu các phần tử ở đầu ra có vị trí bị đè lắp lên nhau thì ta sẽ thực hiện phép cộng để cộng dồn các giá trị vào. Chi tiết từng bước của cách tính được mô tả lần lượt trong các Hình 5.5, 5.6, 5.7, 5.8 và 5.9.
- **Kiểu tích chập chuyển vị 2** đầu tiên sẽ thêm các giá trị 0 vào

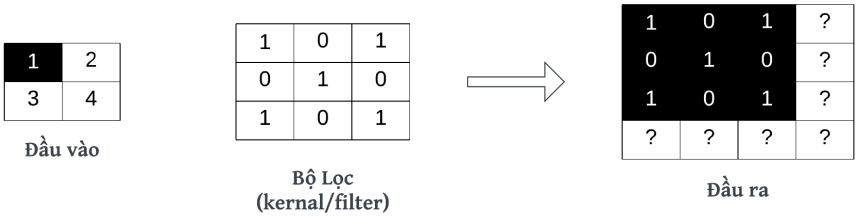
phần biên của ma trận dữ liệu đầu vào để mở rộng nó thành một bản đồ đặc trưng có kích thước lớn hơn. Sau đó thực hiện phép tích chập tiêu chuẩn giữa bản đồ đặc trưng mới và bộ lọc để tạo ra ma trận đầu ra có kích thước mong muốn. Trong ví dụ ở Hình 5.4, ta cần tạo ra ma trận đầu ra có kích thước là 4×4. Do đó, ta cần tính giá trị của độ đệm p (padding) để mở rộng đầu vào theo công thức của phép tích chập là:

$$o = \frac{n + 2 * p - f + s}{s} \Rightarrow p = \frac{o * s - s + f - n}{2} = 2$$

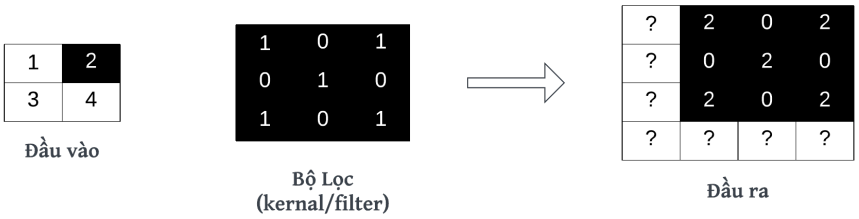
Trong đó: s là độ trượt (stride) được xét bằng 1, còn n, o và f lần lượt là kích thước của ba ma trận đầu vào, đầu ra và bộ lọc. Công thức này được suy ra bằng cách tính ngược lại công thức tính kích thước đầu ra của phép tích chập tiêu chuẩn. Chính vì vậy nên, ta sẽ thêm 2 lớp giá trị bằng 0 vào biên của ma trận đầu ra ban đầu để được ma trận mới như Hình 5.10. Cuối cùng, kết quả đầu ra của phép tích chập chuyển vị kiểu 2 được tính như trong Hình 5.11.



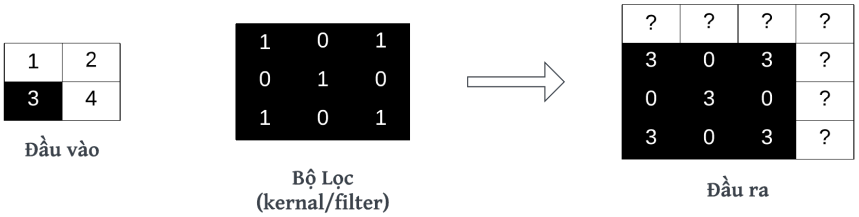
Hình 5.4: Ví dụ cho cách tính tích chập chuyển vị



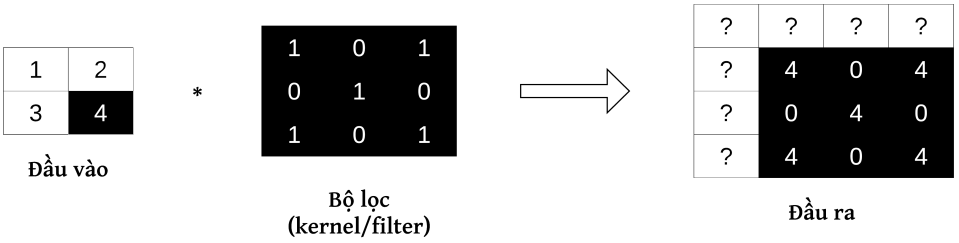
Hình 5.5: Tích chập chuyển vị kiểu 1 bước 1



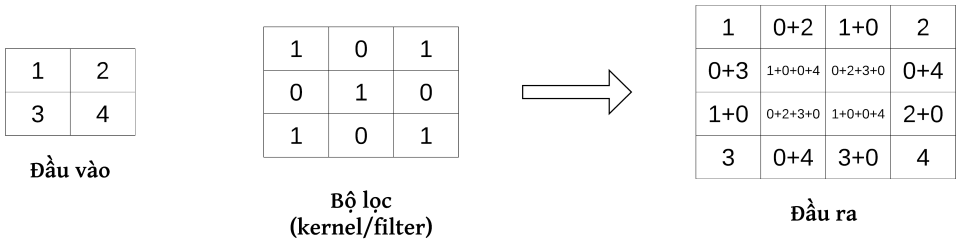
Hình 5.6: Tích chập chuyển vị kiểu 1 bước 2



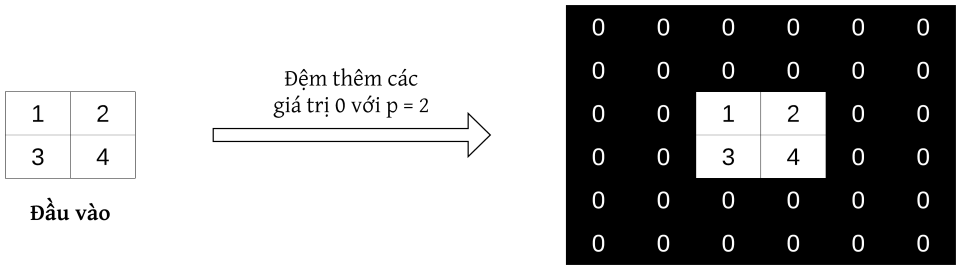
Hình 5.7: Tích chập chuyển vị kiểu 1 bước 3



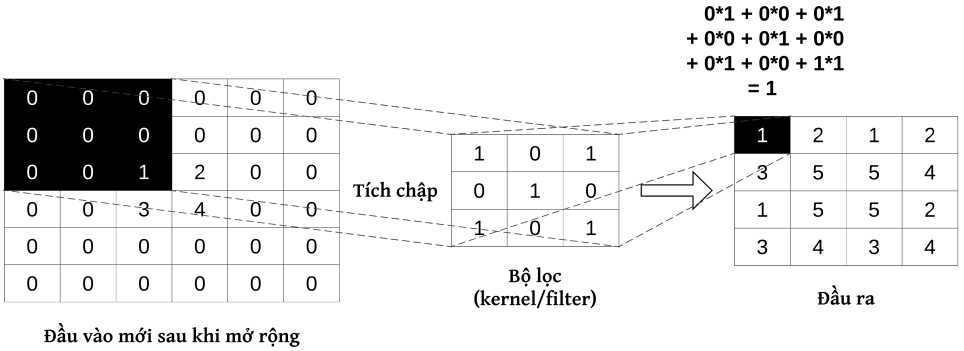
Hình 5.8: Tích chập chuyển vị kiểu 1 bước 4



Hình 5.9: Tích chập chuyển vị kiểu 1 bước 5



Hình 5.10: Mở rộng ma trận đầu vào trong tích chập chuyển vị kiểu 2



Hình 5.11: Kết quả đầu ra của phép tích chập chuyển vị kiểu 2

Ngoài ra, thông thường trong các mạng CNN truyền thống, ta thấy hàm ReLU là hàm kích hoạt sẽ được sử dụng trong mọi tầng tích chập và hàm sigmoid hoặc softmax sẽ được sử dụng sau khi qua tầng kết nối đầy đủ để thực hiện chức năng phân loại. Điều này có chút khác biệt với các tầng kích hoạt ở DCGAN. Trong DCGAN, các tầng của generator cũng

sử dụng hàm kích hoạt ReLU, tuy nhiên ở tầng cuối cùng, tác giả ưu tiên sử dụng hàm tanh vì các dữ liệu hình ảnh được sử dụng cho quá trình huấn luyện nếu được chuẩn hóa trong phạm vi (-1,1) sẽ cho kết quả tốt hơn khi dùng hàm sigmoid để chuẩn hóa về khoảng (0,1) hoặc để nguyên ảnh. Còn đối với discriminator, hàm kích hoạt LeakyReLU sẽ được sử dụng trong tất cả các tầng ngoại trừ tầng cuối cùng sử dụng hàm sigmoid để thu được xác suất nhị phân và thực hiện phân loại giữa ảnh giả và ảnh thật.

Kể từ khi được công bố chính thức, DCGAN đã được thử nghiệm và trở thành một trong những mô hình cơ sở để sinh hình ảnh tiên tiến nhất. Do khả năng xử lý các đặc trưng hình ảnh vượt trội và độ ổn định trong quá trình huấn luyện, nghiên cứu này giúp giảm thiểu các vấn đề thường gặp khi huấn luyện Vanilla GAN như mode collapse và vanishing gradient.

5.5 Kết chương

Mặc dù có kiến trúc bao gồm hai khối xử lý tương tự như mô hình Encoder-Decoder, nhưng mô hình GAN đưa ra một cách tiếp cận sáng tạo và thú vị khi đưa ra *cơ chế học đối kháng* (adversarial learning) giữa hai khối kiến trúc này. Ý tưởng này xuất phát từ giải thuật Minimax nổi tiếng của AI cổ điển, nhưng đã được vận dụng dựa trên cơ chế huấn luyện của mạng nơ-ron để phát triển thành một mô hình sinh phục vụ cho nhiều mục đích khác nhau, từ việc làm giàu dữ liệu huấn luyện cho đến sinh ra các bản nhạc mới theo một phong cách được xác định trước. Đặc biệt, GAN có thể hoạt động theo cơ chế không giám sát, nên mô hình này rất hữu ích trong những tình huống mà việc thu thập và gán nhãn dữ liệu đòi hỏi chi phí cao. Kiến trúc tổng quát của GAN không phụ thuộc vào một kiến trúc mạng đặc biệt nên có thể áp dụng cho các dạng dữ liệu ở các lĩnh vực khác nhau.

Tuy nhiên, cũng như các mô hình sinh khác, GAN phụ thuộc nội dung và chất lượng của dữ liệu được dùng để huấn luyện. Do nội dung do GAN

tạo sinh không kiểm soát được với cơ chế không giám sát nên việc sử dụng dữ liệu này trong các ứng dụng có tính nhạy cảm về độ chính xác của nội dung. Các mô hình sinh nói chung, bao gồm GAN nói riêng, cũng đòi hỏi một dữ liệu huấn luyện lớn, hạ tầng tính toán mạnh và thời gian huấn luyện dài để đạt được hiệu suất sử dụng cần thiết.

5.6 Bài tập

Bài tập 5.6.1. Cho công thức của hai hàm kích hoạt ReLU và LeakyReLU lần lượt là:

- **ReLU**:

$$\varphi(\psi) = max\{\psi, 0\}$$

- **LeakyRelU**:

$$\varphi(\psi) = \begin{cases} \alpha \times (\psi) & \text{if } \psi < 0 \\ r \cdot f_{n-1} & \text{if } \psi \geq 0 \end{cases}$$

Dựa vào 2 công thức trên và hạn chế của mạng GAN đã trình bày ở mục 5.3, giải thích tại sao trong khối discriminator, tác giả thường ưu tiên sử dụng hàm kích hoạt LeakyRELU thay thế cho RELU.

Bài tập 5.6.2. Tính tổng giá trị của hàm mất mát trên generator và discriminator tại một thời điểm trong quá trình huấn luyện một mạng GAN đơn giản, biết r và z lần lượt là véc-tơ mẫu thật và véc-tơ không gian tiềm ẩn được sinh ra một cách ngẫu nhiên làm đầu vào huấn luyện cho khối generator. Khi đó, đầu ra của khối discriminator với mẫu giả là $D(G(z)) = \begin{bmatrix} 0.5 & 0.5 & 0.4 & 0.45 & 0.6 \end{bmatrix}$ và mẫu thật là $D(r) = \begin{bmatrix} 0.8 & 0.85 & 0.7 & 0.7 & 0.75 \end{bmatrix}$.

Bài tập 5.6.3. Tính kết quả đầu ra của bước tính tích chập chuyển vị (theo 2 cách) trong khối generator của một mô hình mạng DCGAN, biết

một số dữ kiện đầu vào như sau:

$$Input = \begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix}, Filter = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Phần II

Ứng dụng các mô hình học sâu vào một số bài toán thực tế

Chương 6

Ứng dụng của học sâu trong bài toán xử lý tiếng nói

6.1 Giới thiệu

6.1.1 Lịch sử phát triển

Nhận dạng giọng nói tự động (Automatic Speech Recognition - ASR) là quá trình giải mã giọng nói dựa trên máy tính để chuyển đổi giọng nói thành văn bản tương ứng. Một hệ thống nhận dạng giọng nói điển hình nhận đầu vào là âm thanh từ loa hoặc micro, sau đó phân tích nó bằng cách sử dụng các mẫu, mô hình hoặc thuật toán để tạo văn bản đầu ra ứng với nội dung được nói.

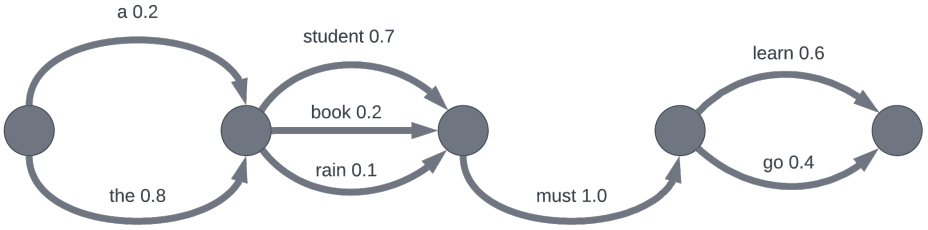
Trên thực tế, xử lý giọng nói đã xuất hiện trước máy tính nhiều thập kỷ. Thiết bị đầu tiên có khả năng nhận dạng giọng nói là một món đồ chơi từ những năm 1920, tên là "Radio Rex"(David and Selfridge, 1962), chú chó Rex sẽ xuất hiện khi ta gọi "Rex"(Hình 6.1). Sau đó, vào năm 1952, một hệ thống ASR được phát triển bởi Davis, Biddulph và Balashek (1952), có thể nhận dạng các số đếm từ 0 đến 9. Năm 1962, Shoebox được IBM giới thiệu với khả năng nhận biết và phân biệt được 16 từ.

Hình 6.1: Radio Rex

Các hệ thống ASR sơ khai này hoạt động dựa trên so sánh mẫu đầu vào của người nói với các mẫu âm thanh được cài sẵn. Phương pháp này chỉ hoạt động tốt ở mức độ nhận dạng các từ có khác biệt nhiều về mặt ngữ âm, nhưng lại kém hiệu quả khi cần nhận dạng nhiều từ vựng hơn. Do đó, hiệu suất của các hệ thống ASR rất thấp vì chúng chỉ có thể nhận dạng được các âm vị cơ bản của giọng nói được phát âm rõ ràng bởi một người nói duy nhất (Rabiner and Juang, 1993).

Thập niên 1970 đánh dấu nhiều bước phát triển quan trọng: tập trung vào việc nhận dạng giọng nói liên tục, phát triển các công cụ nhận dạng giọng nói với bộ từ vựng lớn và các thử nghiệm để tạo ra các hệ thống nhận dạng không phụ thuộc vào người nói. Trong đó, phải kể đến "Harpy", một hệ thống có thể nhận dạng hơn 1000 từ - vốn từ vựng trung bình của một đứa trẻ. Các hệ thống này đã có tác động sâu sắc đến nghiên cứu và phát triển nhận dạng giọng nói khi chứng minh lợi ích của các mô hình thống kê theo hướng dữ liệu qua các phương pháp tiếp cận dựa trên mẫu và chuyển hướng nghiên cứu ASR sang các phương pháp mô hình thống kê như *mô hình Markov ẩn* (Hidden Markov Model - HMM)(Baum and Petrie, 1966).

Các mô hình Markov ẩn biểu thị các đơn vị ngôn ngữ (âm vị hoặc từ) dưới dạng một chuỗi các trạng thái với xác suất chuyển đổi giữa các trạng thái. Để chuyển từ trạng thái này sang trạng thái khác, mô hình sẽ sử dụng xác suất chuyển đổi cao nhất. Điểm mạnh của HMM là nó có thể mô tả xác suất và phân tích hiệu quả các biến đổi của phổ tín hiệu giọng nói theo thời gian, giúp nhận dạng và giải mã hiệu quả đầu vào giọng nói liên tục. Tuy nhiên, HMM đòi hỏi nhiều thời gian đào tạo, dung lượng bộ nhớ lớn và sức mạnh tính toán khổng lồ để lưu trữ tham số mô hình và đánh giá các khả năng xảy ra (xem Hình 6.2).



Hình 6.2: Mô hình Markov ẩn 4 trạng thái

Mặc dù HMM đã trở thành trọng tâm trong việc nghiên cứu ASR vào những năm 1980, giai đoạn này cũng đánh dấu sự phát triển trở lại của các mô hình mạng nơ-ron nhân tạo (ANN). Việc sử dụng kết hợp giữa HMM và ANN trong những năm 1980 đã dẫn đến những nỗ lực đáng kể trong việc xây dựng các hệ thống nhận dạng giọng nói liên tục có bộ từ vựng lớn.

Từ những năm 2000 đến nay, lĩnh vực ASR đã có nhiều bước tiến mạnh mẽ, bao gồm sự phát triển của các thuật toán và kỹ thuật mô hình mới, những tiến bộ trong nhận dạng giọng nói trong môi trường có tiếng ồn và việc tích hợp nhận dạng giọng nói vào công nghệ di động.

6.1.2 Tổng quan về nhận dạng giọng nói

Nhận dạng giọng nói là làm cho máy hiểu, nhận biết được ngữ nghĩa của lời nói. Đây là quá trình biến đổi tín hiệu âm thanh thu được qua micro, qua các thiết bị thu thanh khác... thành một chuỗi các từ, sau đó được nhận dạng để sử dụng trong các ứng dụng điều khiển thiết bị, nhập dữ liệu hoặc soạn thảo văn bản bằng lời... hoặc đưa đến một quá trình xử lý ngôn ngữ ở mức cao hơn.

Trên thực tế, quá trình nhận dạng giọng nói của người là một quá trình phức tạp. Hiện nay, các nhà nghiên cứu đã cố gắng tìm hiểu, phân tích và mô phỏng quá trình nhận dạng giọng nói của người dưới dạng các chương trình máy tính. Tuy nhiên đây là vấn đề rất rộng, có liên quan tới nhiều ngành nghiên cứu như sinh học, hoá học, vật lý ... Do vậy, hiện nay vẫn chưa có một giải pháp thật sự hoàn hảo để giải quyết bài toán này (Jurafsky and Martin, 2020).

Nhận dạng giọng nói là quá trình phức tạp bao gồm nhiều khâu biến đổi. Tín hiệu mà người phát ra là tín hiệu tương tự, qua quá trình lấy mẫu, lượng tử hoá và mã hoá để thu được các mẫu tín hiệu dạng số (tín hiệu mà máy tính có thể hiểu và xử lý được). Các mẫu tín hiệu này được đưa vào để trích chọn đặc trưng. Những đặc trưng này sẽ là đầu vào cho quá trình nhận dạng. Sau khi nhận dạng tín hiệu người dùng phát âm, hệ thống sẽ đưa ra kết quả nhận dạng. Tuỳ thuộc vào mô hình ứng dụng mà hệ thống sẽ có các dạng đầu ra khác nhau.

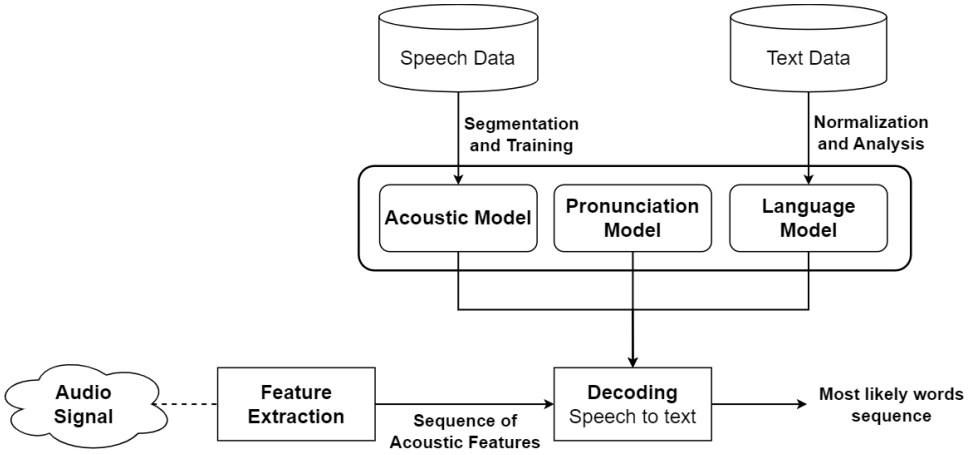
Hệ thống nhận dạng giọng nói phân hóa trên ba khía cạnh chính: tính phụ thuộc người nói, tính liên tục của giọng nói và kích thước từ vựng. Dựa trên dữ liệu huấn luyện, hệ thống ASR có thể phụ thuộc vào người nói (khi hệ thống phải được đào tạo cho từng người nói riêng lẻ); không phụ thuộc vào người nói (khi dữ liệu chứa nhiều giọng của những người nói khác nhau nên hệ thống có thể nhận dạng chính xác bất kỳ giọng nói mới nào); và thích ứng (khi hệ thống có thể dần dần thích ứng với một người dùng cụ thể thông qua đào tạo). Từ khía cạnh của tính liên tục của lời nói, có các hệ thống nhận dạng từ rời rạc, xác định các từ được

phát ra một cách độc lập; hệ thống nhận dạng giọng nói liên tục, có khả năng nhận dạng toàn bộ câu mà không có khoảng dừng giữa các từ; và hệ thống xác định từ khóa từ một giọng nói liên tục. Cuối cùng, các hệ thống ASR có thể được thiết dựa trên kích thước từ vựng mà chúng có thể nhận dạng hoặc không giới hạn về từ vựng.

6.2 Kiến trúc cơ bản của hệ thống nhận dạng giọng nói

Nhìn chung, tất cả các hệ thống ASR đều có kiến trúc giống nhau và có thể được chỉnh sửa hoặc bổ sung cho phù hợp với bài toán cụ thể. Một hệ thống ASR thường bao gồm năm thành phần chính (Hình 6.3):

- Bộ trích chọn đặc trưng (Feature extraction)
- Mô hình âm học (Acoustic model)
- Mô hình ngôn ngữ (Language model)
- Mô hình phát âm (Pronunciation model)
- Bộ giải mã (Decoder)



Hình 6.3: Kiến trúc cơ bản của hệ thống nhận dạng giọng nói

6.2.1 Bộ trích chọn đặc trưng

Trong nhận dạng giọng nói, mục tiêu chính của bước trích chọn đặc trưng là tính toán các véc-tơ đặc trưng nhằm cung cấp một biểu diễn nhỏ gọn hơn của tín hiệu đầu vào. Một thuật toán trích chọn đặc trưng tốt phải có thể trích xuất các đặc trưng trong thời gian thực trong khi vẫn đảm bảo giữ lại tối đa thông tin. Đồng thời, các đặc trưng được trích xuất phải cho phép hệ thống phân biệt được các âm gần giống nhau mà không cần quá nhiều dữ liệu huấn luyện, và phải biểu diễn được các thông tin bất biến giữa nhiều người nói và môi trường nói. Một số phương pháp trích chọn đặc trưng được sử dụng phổ biến bao gồm *Linear Discriminant Analysis* (LDA), *Linear Predictive Coding* (LPC), *Cepstral*, *Filter-Bank*, *Cepstral Mean Subtraction* (CMS) và *Mel-Frequency Cepstrum Coefficients* (MFCC) (Anusuya and Katti, 2011). Đối với các bài toán nhận dạng giọng nói trong môi trường có tiếng ồn, nhiều kỹ thuật trích chọn đặc trưng mạnh mẽ và phức tạp hơn như *Zero Crossing Peak Amplitude* (ZCPA), *Average Localized Synchrony Detection* (ALSD), *Perceptual Minimum Variance Distortionless Response* (PMVDR), *Power-Normalized Cepstral Coefficients* (PNCC), *Invariant Integration Features* (IIF), *Sparse Auditory Reproducing Kernel* (SPARK) và *Gabor filter bank* được sử dụng (J. Li et al., 2014).

Mặc dù có nhiều kỹ thuật trích chọn đặc trưng được sử dụng, nhưng phổ biến nhất vẫn là các đặc trưng MFCC (Davis and Mermelstein, 1980). Dưới đây ta sẽ mô tả ngắn gọn về phương pháp MFCC, bao gồm 6 bước như sau:

1. Framing: Do tín hiệu tiếng nói có thể coi là ổn định trong một khoảng thời gian ngắn cỡ khoảng vài chục ms, nên khi tiến hành các phép phân tích biến đổi trên tín hiệu tiếng nói người ta thường chia tín hiệu thành các đoạn cỡ 10-30 ms, công việc đó được gọi là phân khung tín hiệu (Framing).

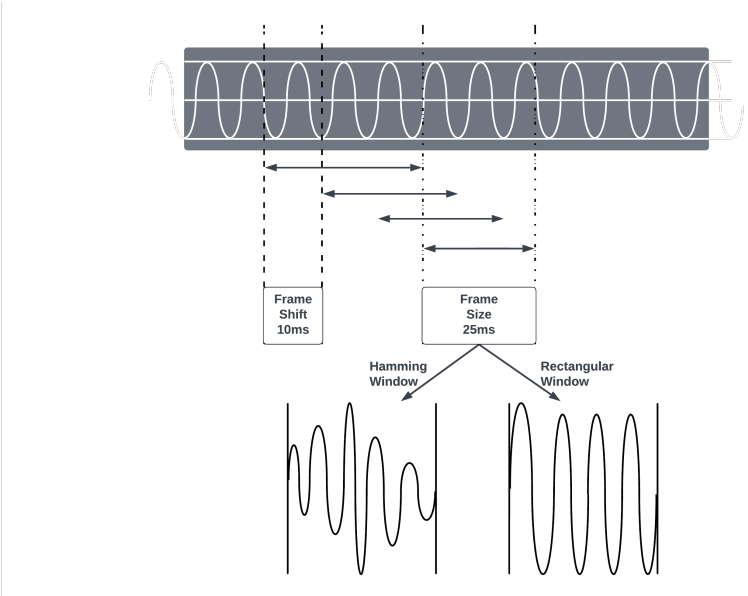
2. Windowing: Quá trình framing sẽ làm rời rạc hóa tín hiệu âm thanh, gây ra sự ngắt quãng ở 2 đầu mỗi đoạn được cắt. Do đó, Hamming

window được sử dụng để làm mượt phổ tín hiệu và làm giảm tính ngắt quãng bằng cách làm giảm các giá trị tín hiệu về 0 tại hai đầu:

$$y_t(n) = x_t(n) * w(n), 0 \leq n \leq N - 1 \tag{6.1}$$

$$w(n) = 0.54 - 0.46cos(\frac{2\pi n}{N}) \tag{6.2}$$

Trong đó, N là số đoạn, x là tín hiệu đầu vào, y là tín hiệu đầu ra, $w(n)$ là công thức biểu diễn Hamming window.



Hình 6.4: Framing và windowing

3. Biến đổi Fourier: Thực hiện việc biến đổi từ miền thời gian sang miền tần số sử dụng phép biến đổi Fourier. Phép biến đổi thường được sử dụng trong quá trình này là FFT, công thức của FFT được định nghĩa như sau:

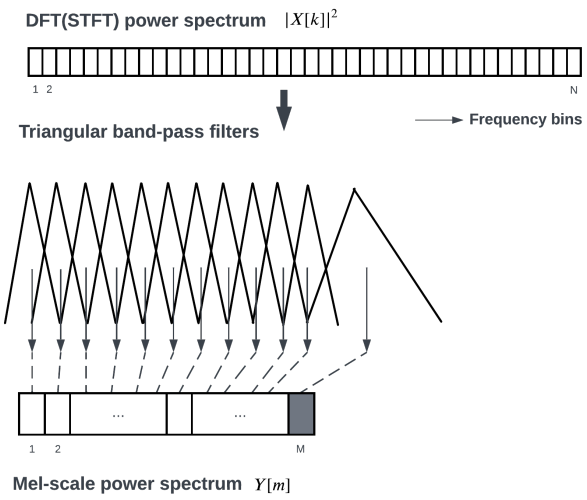
$$S(k) = \Sigma_{n=0}^{N} y(n) e^{-2\pi K \frac{n}{N}}, 0 \leq k \leq N - 1 \tag{6.3}$$

Trong đó, N là số lượng cửa sổ, K là số lượng hệ số cần tính toán.

4. Mel Frequence Warping: Sự cảm nhận của tai người theo tần số đối với âm thanh thuần khiết hay tiếng nói không tuân theo thang tuyến tính. Vì vậy, đối với mỗi âm có tần số thực tế f đo bằng Hz, có một tần số pitch cảm thụ được đo trên thang Mel. Thang đo Mel là phép biến đổi tuyến tính với tần số dưới 1 kHz và biến đổi logarit đối với tần số trên 1 kHz [4]. Chúng ta có thể sử dụng công thức xấp xỉ sau đây để tính Mel khi biết tần số đo bằng Hz:

$$mel(f) = 1127ln(1 + \frac{f}{100}) \tag{6.4}$$

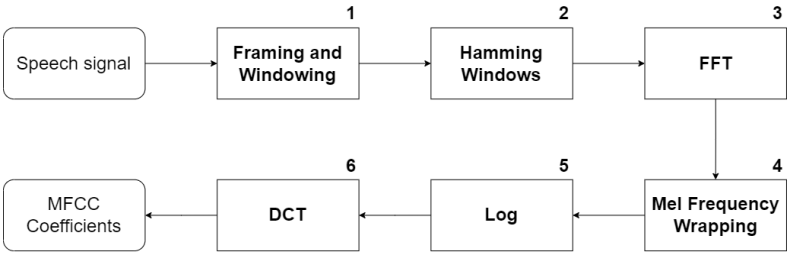
Sau đó, ta bình phương các giá trị thu được từ biến đổi Fourier và thu được phổ công suất (DFT power spectrum). Tiếp theo, áp dụng 1 tập các bộ lọc thông dải (Mel-scale filter) trên từng khoảng tần số (mỗi bộ lọc áp dụng trên 1 dải tần xác định) (xem Hình 6.5). Giá trị đầu ra của từng bộ lọc là năng lượng dải tần số mà nó bao phủ được, gọi là Mel-scale power spectrum. Ngoài ra, các bộ lọc dùng cho dải tần thấp thường hẹp hơn các bộ lọc dùng cho dải tần cao.



Hình 6.5: Mel-scale power spectrum

5. Log: Ta thực hiện tính log trên kết quả Mel-scale power spectrum nhằm giúp giảm các biến thể âm thanh không đáng kể trong nhận dạng giọng nói.

6. Biến đổi cosine rời rạc (Discrete cosine transform - DCT): Cuối cùng, ta sẽ thực hiện biến đổi cosine rời rạc để biến đổi các hệ số Mel spectrum thành các hệ số cepstral. DCT cho phép hầu hết thông tin được bảo toàn trong khi vẫn giảm số chiều dữ liệu bằng cách loại bỏ các hệ số có giá trị cao nhưng năng lượng thấp. Hầu hết thông tin tín hiệu được biểu diễn trong các hệ số MFCC đầu tiên, thông thường ta chỉ cần sử dụng từ 8-13 hệ số MFCC cho hệ thống nhận dạng giọng nói.



Hình 6.6: Lược đồ MFCC

6.2.2 Mô hình âm học

Mô hình âm học là một trong những thành phần quan trọng nhất cho hệ thống nhận dạng giọng nói, được dùng để mô hình hóa biểu diễn xác suất của các véc-tơ đặc trưng tính toán từ giọng nói ban đầu. Mục đích của nó là dự đoán các âm vị có khả năng xảy ra nhất từ âm thanh đầu vào. Giả sử rằng x là một chuỗi các véc-tơ âm học đầu vào và w_i ($i = 1, \cdots, K$) là một trong K khả năng có thể có quan sát được, kết quả nhận dạng được biểu diễn bởi:

$$w^* = argmax \frac{P(x|w_i)P(w_i)}{P(x)} \tag{6.5}$$

$$= argmax P(x|w_i)P(w_i) \tag{6.6}$$

Do đó, kết quả nhận dạng w^* sẽ cho khả năng xảy ra cao nhất trong số tất cả các từ ứng viên w_i. Xác suất $P(x|w_i)$ được ước lượng bởi mô hình âm học. Xác suất tiền tố $P(w_i)$ của chuỗi đầu ra độc lập với tín hiệu đầu vào và cần một mô hình ngôn ngữ để ước lượng. $P(X)$ là xác suất để quan sát được các véc-tơ âm học X, nó giống nhau đối với mỗi chuỗi âm vị w nên có thể được lược bỏ.

Trong quá trình xây dựng một mô hình âm học, việc lựa chọn hệ thống âm vị cơ bản để mô hình hóa âm thanh là rất quan trọng. Các hệ thống âm vị khác nhau có thể tạo ra sự khác biệt đáng kể về hiệu suất của hệ

thống nhận dạng giọng nói.

6.2.3 Mô hình ngôn ngữ

Những tiến bộ trong lĩnh vực nhận dạng giọng nói đã làm tăng nhu cầu về các mô hình ngôn ngữ vì lời nói không nhất thiết phải tuân theo các quy tắc ngữ pháp cứng nhắc. Văn hóa và con người ở các vùng miền khác nhau cũng làm ảnh hưởng đến phong cách nói của mỗi người. Vì vậy, cần phải có một mô hình ngôn ngữ tốt để giải quyết những vấn đề này trong thời gian thực. Các mô hình ngôn ngữ sử dụng các ràng buộc về cấu trúc ngữ pháp của ngôn ngữ để dự đoán xác suất xuất hiện của một từ đối với một chuỗi từ cụ thể. Những ràng buộc về cấu trúc này có thể khác nhau giữa các ngôn ngữ. Mô hình ngôn ngữ sẽ giúp xác định các từ hợp lệ và các trường hợp có thể sử dụng của chúng.

Mô hình ngôn ngữ được sử dụng phổ biến hiện nay là các mô hình ngôn ngữ n-gram và các mô hình ngôn ngữ dựa trên mạng nơ-ron nhân tạo.

6.2.4 Mô hình phát âm

Các từ điển phát âm mô tả cách một từ được phát âm và biểu diễn dưới dạng các âm vị. Từ điển phát âm là một phần của mô hình phát âm. Việc chuyển đổi tín hiệu giọng nói thành văn bản được thực hiện bằng cách nhận dạng từng âm vị nhỏ hơn được phát âm. Sau đó, mô hình phát âm sẽ xác định cách các đơn vị nhỏ này có thể được kết hợp để tạo thành các từ hợp lệ. Một cách để thiết kế mô hình phát âm cho các hệ thống nhận dạng giọng nói là sử dụng các quy tắc âm vị học đã được nghiên cứu và phát triển rộng rãi, trong đó phổ biến nhất là sử dụng các phiên âm quốc tế từ IPA. Khi mô hình âm học, ngôn ngữ và phát âm được phát triển, bộ giải mã sẽ sử dụng chúng để xuất ra văn bản tương ứng với tín hiệu âm thanh nhận được.

6.2.5 Bộ giải mã

Nhiệm vụ của bộ giải mã là tìm chuỗi từ W có khả năng xảy ra nhất sử dụng câu nói đầu vào và các mô hình âm học, phát âm, ngôn ngữ. Vấn đề này có thể được giải quyết bằng cách sử dụng các thuật toán quy hoạch động. Thuật toán Viterbi thường được sử dụng trong quá trình giải mã để tìm kiếm giả thuyết tốt nhất ứng với các trạng thái có được. Đối với các hệ thống nhận dạng có bộ từ vựng lớn, việc xét tất cả các khả năng có thể có trong quá trình đệ quy của thuật toán Viterbi đòi hỏi lượng tính toán cực kỳ lớn. Để giải quyết vấn đề này, chúng ta có thể sử dụng giải thuật *beam search*, chỉ những từ có xác suất xảy ra trên ngưỡng mới được xem xét khi mở rộng đến bước thời gian tiếp theo. Cách tiếp cận này sẽ giúp đẩy nhanh quá trình tìm kiếm, tuy nhiên sẽ đánh đổi bằng độ chính xác của quá trình giải mã.

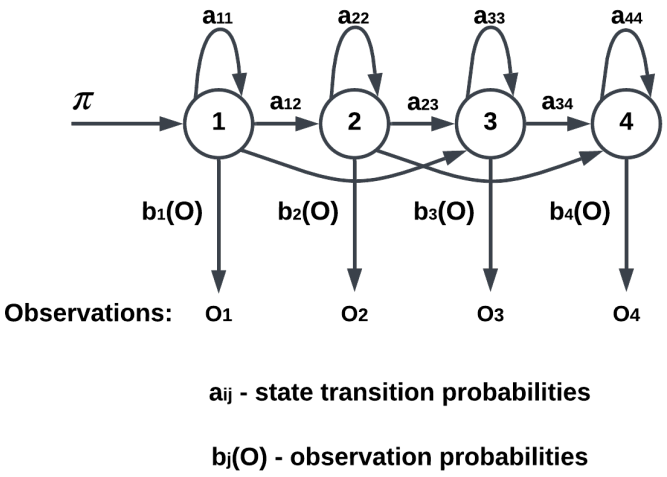
6.3 Một số phương pháp xây dựng mô hình nhận dạng

6.3.1 Mô hình Markov ẩn

Mô hình Markov ẩn (Hidden Markov Model - HMM) (Baum and Petrie, 1966) là một trong những kỹ thuật được sử dụng phổ biến và thành công nhất trong nhận dạng giọng nói. Nó rất linh hoạt và có thể dễ dàng thích ứng tùy theo cấu trúc yêu cầu. Nhiều hệ thống nhận dạng giọng nói hiện nay vẫn sử dụng HMM để xây dựng mô hình âm học nhằm biểu diễn các đặc trưng theo thời gian hiện diện trong tín hiệu giọng nói đầu vào. Mô hình Markov ẩn được sử dụng trong nhận dạng giọng nói vì tín hiệu giọng nói có thể được xem như là tín hiệu ổn định trong khoảng thời gian ngắn (ví dụ: 10ms). Do đó, giọng nói có thể được coi là một mô hình Markov. Một lý do khác khiến HMM phổ biến là chúng có thể được đào tạo tự động và sử dụng đơn giản, đồng thời khả thi về mặt tính toán.

Mô hình Markov ẩn được định nghĩa là một máy trạng thái (state ma-

chine). Các trạng thái của mô hình được biểu diễn dưới dạng các nút và các chuyển đổi được biểu diễn bằng các cạnh. Các yếu tố của mô hình Markov ẩn là xác suất chuyển đổi, xác suất quan sát và phân phối xác suất ban đầu. Số lượng trạng thái được thiết lập trong mô hình HMM là cố định và được xác định trước, các trạng thái này có thể khác so với số trạng thái ẩn trong tín hiệu giọng nói đầu vào. HMM giả định rằng tín hiệu giọng nói đã cho có thể được đặc trưng hóa bằng một quá trình tham số ngẫu nhiên hóa, và do đó các tham số của nó có thể được xác định một cách rõ ràng và chính xác. Thuật toán này là một phần mở rộng của giải thuật chuỗi Markov (Markov chain), có thể tạo ra các thông tin đầu ra có nghĩa bất kể trạng thái đầu vào là gì, kết quả đầu ra của HMM là một hàm xác suất trạng thái (Rabiner, 1989). Một ví dụ về HMM được thể hiện trong Hình 6.7.



Hình 6.7: Mô hình HMM 4 trạng thái

Các ký hiệu dưới đây thường được sử dụng trong mô hình Markov ẩn:

- N: số trạng thái trong mô hình
- M: số ký hiệu quan sát có thể có

- T: độ dài của dãy quan sát (hay số ký hiệu trong dãy quan sát)
- q_t: trạng thái của mô hình tại thời điểm t
- $V = \{V_1, V_2, \ldots, V_M\}$: tập rời rạc các ký hiệu quan sát
- $\pi = \{\pi_1, \pi_2, \ldots, \pi_N\}$: tập các phân bố xác suất cho trạng thái khởi đầu, π_i là xác suất để trạng thái i được chọn tại thời điểm khởi đầu t=1

$$\begin{cases} \Sigma_{i=1}^{N} \pi_i = 1 \\ \pi_i \geq 0, i = 1, 2, \ldots, N \end{cases} \tag{6.7}$$

- $A = \{a_{ij}\}$: ma trận xác suất chuyển đổi trạng thái với a_{ij} là xác suất để trạng thái j xuất hiện ở thời điểm t+1 khi trạng thái i đã xuất hiện ở thời điểm t. Giả thiết rằng a_{ij} độc lập với thời gian t: $a_{ij} = P(q_{t+1} = j | q_t = i)$

$$\begin{cases} \Sigma_{j-1}^{N} a_{ij} = 1, i = 1, 2, \ldots, N \\ a_{ij} \geq 0 \end{cases} \tag{6.8}$$

- $B = \{b_j(v_k)\}$: các hàm đo xác suất phát sinh mẫu, đo xác suất v_k phát sinh khi mô hình ở trạng thái j

$$\begin{cases} \Sigma_{k=1}^{M} b_j(v_k) = 1, j = 1, 2, \ldots, N \\ b_j(v_k) \geq 0 \end{cases} \tag{6.9}$$

- $O = \{O_1, O_2, \ldots, O_T\}$: tập hợp các ký hiệu quan sát được theo thời gian, O_t là ký hiệu quan sát được ở thời điểm t

Dưới góc độ toán học, mô hình Markov ẩn có thể được định nghĩa là mô hình xác suất $\lambda = (A, B, \pi)$, A, B, π được gọi là bộ tham số của mô hình HMM. Hoạt động của HMM có thể được mô tả như sau: tại thời điểm $t = 1$, mô hình đang ở trạng thái q_1 nào đó và phát sinh một ký hiệu quan sát O_1, sau đó, tại $t = 2$, mô hình chuyển sang trạng thái q_2

và tiếp tục phát sinh ký hiệu quan sát O_2. Cứ tiếp tục như vậy cho đến thời điểm $t = T$, ta có dãy quan sát $O = \{O_1, O_2, \ldots, O_T\}$ và dãy trạng thái $Q = \{q_1, q_2, \ldots, q_T\}$. Dãy trạng thái Q phụ thuộc vào xác suất chọn trạng thái khởi đầu π_i và xác suất chuyển trạng thái a_{ij}.

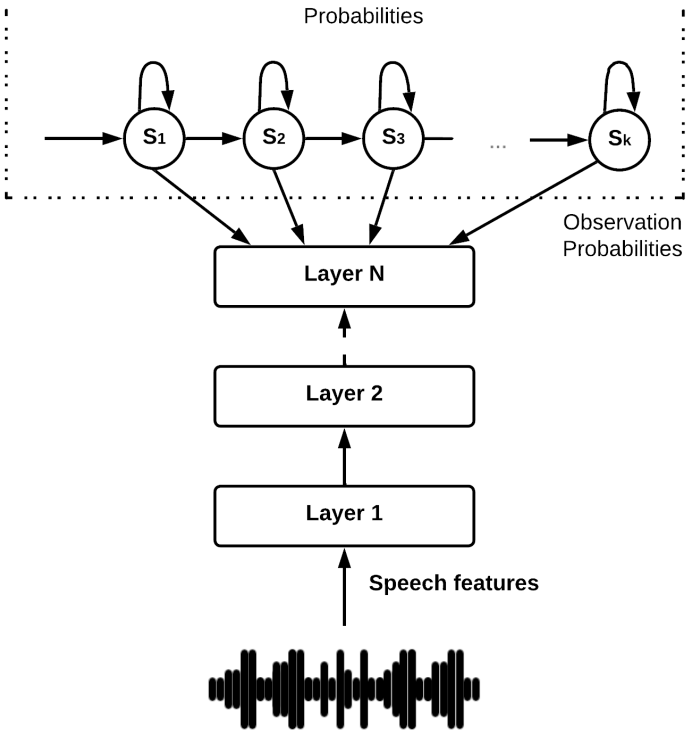
Mô hình Markov ẩn với mật độ liên tục (Continuous Density Hidden Markov Model - CDHMM) là một trong những mô hình cải tiến được sử dụng phổ biến của mô hình Markov ẩn. Mô hình này sử dụng thuật toán ước lượng hợp lý cực đại (Maximum likelihood) để đào tạo mô hình phục vụ cho quá trình nhận dạng. Sử dụng kỹ thuật này, ta có thể tính toán các biến thể có thể xảy ra trong và giữa các âm vị một cách linh hoạt và chính xác hơn.

6.3.2 Kết hợp mô hình Markov ẩn và mạng thần kinh nhân tạo

Trong thực tế, các hệ thống nhận dạng giọng nói dựa trên HMM có hiệu quả trong nhiều trường hợp, nhưng vẫn tồn đọng một số nhược điểm làm hạn chế khả năng ứng dụng của công nghệ nhận dạng giọng nói trong đời sống thực tế. Nhiều nghiên cứu đã được thực hiện nhằm khắc phục những hạn chế này bằng cách sử dụng mạng thần kinh nhân tạo (ANN) như một mô hình thay thế cho mô hình Markov ẩn trong nhận dạng giọng nói, nhưng các nghiên cứu này cho thấy ANN không thành công trong việc xử lý chuỗi tín hiệu giọng nói liên tục trong thời gian dài. Từ cuối những năm 1980 đến đầu những năm 1990, một số nhà nghiên cứu bắt đầu khám phá lĩnh vực nghiên cứu mới bằng cách kết hợp mô hình Markov ẩn và mạng thần kinh nhân tạo với các kiến trúc lai, cho thấy nhiều cải thiện đáng kể không chỉ đối với nhận dạng giọng nói mà còn trong nhiều khía cạnh xử lý giọng nói khác.

Một hệ thống nhận dạng giọng nói lai là sự kết hợp của Mô hình Markov ẩn, Mô hình hỗn hợp Gaussian (Gausian Mixture Model - GMM) với các loại mạng nơ-ron khác nhau bao gồm cả mạng học sâu (DNN) nhằm tận dụng các ưu điểm của từng kỹ thuật để cải thiện khả năng nhận dạng

giọng nói. Sự kết hợp của mạng học sâu và mô hình Markov ẩn là một trong những mô hình nhận dạng giọng nói có thể tận dụng khả năng mạnh mẽ trong việc học các biểu diễn đặc trưng tổng quát của mạng học sâu và khả năng lập mô hình tuần tự của HMM (Yu and Deng, 2015), các mô hình này được gọi chung là DNN-HMM. Trong mô hình này, các biểu diễn tín hiệu giọng nói được mô hình hóa bằng HMM và xác suất quan sát được ước lượng thông qua DNN. Mỗi nơ-ron đầu ra của DNN được huấn luyện để ước tính xác suất hậu tố từ các trạng thái mật độ liên tục của HMM trích xuất được dựa trên tín hiệu âm thanh. Hình 6.8 minh họa kiến trúc chung của loại mô hình này.



Hình 6.8: Kiến trúc chung của mô hình DNN-HMM

HMM đóng vai trò trích xuất các thuộc tính tuần tự của giọng nói và

DNN làm nhiệm vụ tính toán xác suất xảy ra của các chuỗi âm vị có thể có từ tín hiệu giọng nói. Loại mô hình này có hai ưu điểm chính: việc đào tạo có thể được thực hiện bằng cách sử dụng thuật toán Viterbi, thuật toán này giúp tìm ra con đường tối đa hóa xác suất xảy ra từ các giá trị quan sát được từ HMM; và việc giải mã nhanh chóng và hiệu quả khi triển khai trong thực tế. Shahin và cộng sự (Shahin et al., 2014) đã thực hiện so sánh các kỹ thuật nhận dạng âm vị dựa trên GMM-HMM và DNN-HMM, kết quả là DNN-HMM hoạt động tốt hơn GMM-HMM cho tất cả các thử nghiệm đối với cả giọng nói trong môi trường yên tĩnh và môi trường có tiếng ồn. DNN-HMM thể hiện độ chính xác trên 85% kể cả khi được sử dụng với giọng nói trong môi trường có tiếng ồn. Một so sánh khác giữa mô hình GMM-HMM và DNN-HMM cho cơ sở dữ liệu văn bản và giọng nói đa ngôn ngữ GlobalPhone được trình bày bởi Tach Belie (Tachbelie et al., 2020). Dựa trên kết quả thử nghiệm, mô hình DNN-HMM cho kết hợp tốt hơn hẳn các mô hình dựa trên GMM-HMM bất kể độ dài của tín hiệu giọng nói.

Một phương pháp tiếp cận khác được phát triển bởi Dahl et al., 2012 gọi là CD-DNN-HMM (Context-Dependent Pre-Trained Deep Neural Networks). Phương pháp này cho phép khởi tạo tham số cho mạng nơ-ron một cách hiệu quả hơn nhằm hỗ trợ tối ưu hóa và giảm lỗi tổng quát hóa. Phương pháp này được sử dụng trong nhận dạng giọng nói với bộ từ vựng lớn và mang lại kết quả tốt hơn so với các mô hình DNN-HMM thông thường.

Để tận dụng được các thông tin giọng nói trong quá khứ với thông tin ngữ cảnh dài hơn, các mạng nơ-ron mạnh mẽ hơn được sử dụng như attention, RNN, LSTM, TDNN hay Transformer. Các mạng này không giới hạn kích thước của ngữ cảnh, thông tin ngữ cảnh có thể được bảo lưu bên trong mạng bởi các kết nối và các trạng thái ẩn của mạng phụ thuộc vào toàn bộ lịch sử đầu vào. Cách tiếp cận này giúp khai thác thông tin ngữ cảnh dài hơn giữa các từ và cho phép các điều chỉnh thích hợp với từng bài toán cụ thể. Các phương pháp tiếp cận hơn cho phép mô hình được huấn luyện theo hướng bán giám sát (semi-supervised learning) (Xu

et al., 2014) hay tự giám sát (self-supervised learning) trên cả dữ liệu có gắn nhãn và dữ liệu chưa gắn nhãn. Việc này sẽ giúp mô hình tận dụng được lượng lớn dữ liệu giọng nói chưa gắn nhãn và cải thiện được độ chính xác nhận dạng.

Nhìn chung, hầu hết các hệ thống nhận dạng giọng nói kết hợp vẫn còn một số hạn chế sau:

- Kiến trúc gồm nhiều thành phần nhỏ hơn làm phức tạp hóa việc thiết kế, huấn luyện, tối ưu hóa và triển khai thực tế.
- Mỗi thành phần của hệ thống kết hợp đều có mô hình thống kê hoặc mạng nơ-ron riêng biệt, chúng có thể tạo ra các lỗi độc lập không thống nhất, làm ảnh hưởng tới khả năng dự đoán của mô hình và rất khó để cải thiện được độ chính xác nhận dạng.
- Hệ thống nhận dạng bị giới hạn trong khả năng xử lý và tính toán của HMM.

Tuy nhiên, hiện nay, nhiều dịch vụ nhận dạng giọng nói được triển khai trong thực tế vẫn được phát triển dựa trên HMM-DNN. Đó là bởi vì khả năng giải quyết các bài toán thực tế đòi hỏi ít tài nguyên tính toán và đào tạo hơn của các mô hình HMM-DNN so với các mô hình phát triển hoàn toàn dựa trên mạng học sâu.

6.3.3 Mô hình end-to-end

Tất cả những hạn chế của các hệ thống nhận dạng nhiều thành phần nói trên đã trở thành nguồn động lực cho các nhà nghiên cứu phát triển hệ thống mới khi toàn bộ hệ thống được đào tạo như một mô hình lớn duy nhất, sau này được gọi là mô hình nhận dạng đầu cuối (end-to-end). Những gì nó làm đơn giản chỉ là phân tích thông tin từ tín hiệu giọng nói đầu vào $X = \{x_1, x_2, \ldots, x_T\}$ và tạo ra chuỗi kết quả $Y = \{y_1, y_2, \ldots, y_U\}$ chỉ với một mô hình xác suất $Y = p(Y|X)$. Trong đó, x_t là thông tin âm học ở thời gian t, $y \in V$ và V là tập hợp tất cả các âm vị có thể nhận dạng.

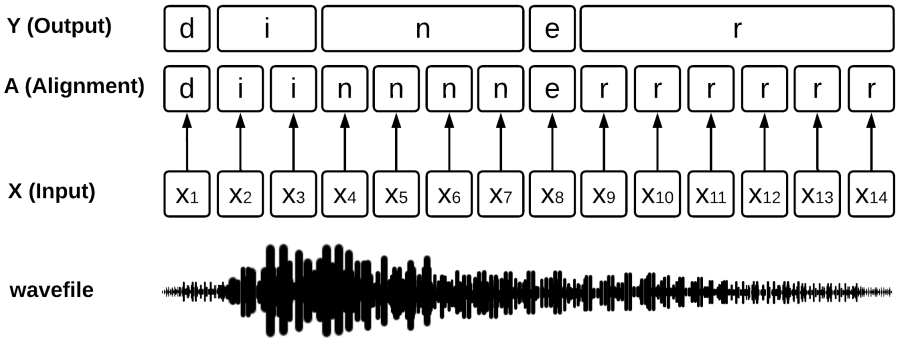
Mô hình nhận dạng đầu cuối đầu tiên như vậy được phát triển dựa trên thuật toán CTC (Connectionist Temporal Classification) được giới thiệu bởi Graves et al., 2006. Các mô hình dựa trên CTC có thể trực tiếp xuất ra văn bản nhận dạng được, trong khi các mô hình dựa trên HMM chủ yếu xuất ra các đơn vị nhỏ hơn như âm vị và cần phải xử lý rất nhiều sau đó để ra được kết quả. Việc áp dụng CTC đơn giản hóa đáng kể kiến trúc và thời gian đào tạo của mô hình. Mặc dù vẫn còn hạn chế, nó đã đặt nền móng cho việc phát triển các mô hình nhận dạng đầu cuối, tạo tiền đề cho việc xây dựng các mô hình tuần tự mạnh mẽ hơn (Chan et al., 2015). Các mô hình nhận dạng đầu cuối thường đòi hỏi nhiều dữ liệu đào tạo hơn so với các mô hình dựa trên HMM, tuy nhiên nó có thể cho kết quả tốt hơn hẳn mô hình dựa trên HMM khi có lượng dữ liệu huấn luyện đủ lớn. Hiện nay, mô hình nhận dạng đầu cuối đang là xu hướng phát triển trong nhận dạng giọng nói và ngày càng trở nên phổ biến hơn trong triển khai thực tế (Hannun et al., 2014).

Connectionist Temporal Classification

Connectionist Temporal Classification (CTC) được sử dụng để gắn nhãn dữ liệu có trình tự trong một kiến trúc mạng nơ-ron duy nhất mà không cần bất kỳ bước phân đoạn hay hậu xử lý nào. Ý tưởng cơ bản của CTC là giải thích các kết quả đầu ra của mạng dưới dạng phân phối xác suất có điều kiện trên tất cả các chuỗi ký tự đầu ra có thể có. Sử dụng phân phối này, ta có thể trực tiếp tối đa hóa xác suất của các chuỗi ký tự mong muốn bằng một hàm mục tiêu duy nhất (Hannun, 2017). CTC cho phép chuỗi đầu ra có cùng độ dài với chuỗi đầu vào, sau đó thực hiện thu gọn bằng cách kết hợp các ký tự liên tiếp giống nhau thành 1 ký tự duy nhất, kết quả là một chuỗi ngắn hơn được sinh ra.

Bây giờ, hãy tưởng tượng một người nào đó nói từ *dinner*, giả sử chúng ta có một hàm ánh xạ mỗi đặc trưng âm học đầu vào x_i với chữ cái tương ứng có khả năng xảy ra nhất. Chúng ta gọi chuỗi các chữ cái được tạo ra từ ánh xạ này là một alignment, vì nó cho chúng ta biết vị trí trong tín hiệu âm thanh mà mỗi chữ cái thể hiện. Một aligment được gọi là hợp lệ

nếu sau khi thu gọn ta thu được chuỗi đầu ra giống với chuỗi mục tiêu. Hình 6.9 minh họa một alignment như vậy, điều gì sẽ xảy ra nếu chúng ta tiến hành thu gọn chỉ bằng cách loại bỏ các chữ cái trùng lặp liên tiếp.

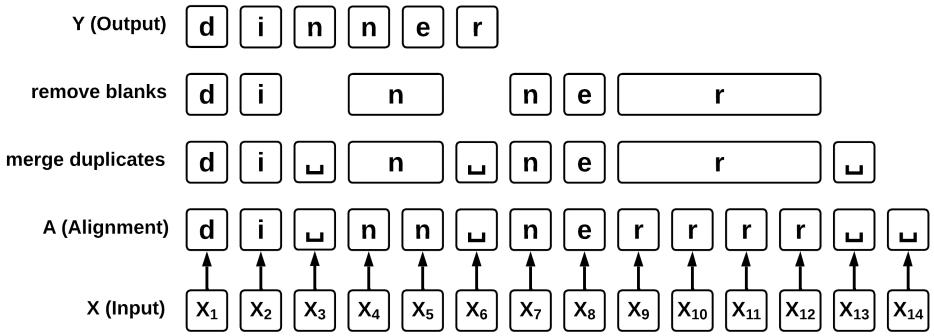


Hình 6.9: Ví dụ về alignment

Rõ ràng phương pháp thu gọn ngây thơ này sẽ cho ra đầu ra là *diner* chứ không phải *dinner*! Phương pháp này không thể xử lý các ký tự đôi. Một vấn đề khác nữa của cách tiếp cận này, đó chính là việc ánh xạ từng đặc trưng âm học đầu vào thành chữ cái cụ thể không thật sự hợp lý, bởi vì ta không nói chuyện liên tục mà còn có những khoảng lặng, ngắt nghỉ khi ta không nói gì hết. Chúng ta không muốn phải ánh xạ những khoảng lặng này thành các chữ cái không liên quan!

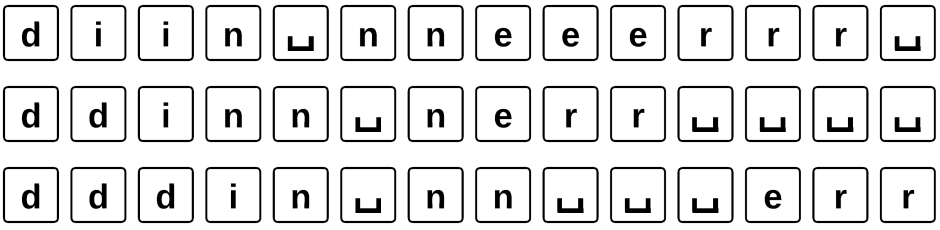
Thuật toán CTC giải quyết cả hai vấn đề trên bằng cách thêm vào một ký hiệu đặc biệt biểu diễn cho khoảng trống, chúng ta sẽ ký hiệu là ϵ. Khoảng trống có thể được sử dụng trong các alignment bất cứ khi nào chúng ta không muốn ánh xạ một chữ cái. Khoảng trống cũng có thể được sử dụng giữa các chữ cái; vì khi thu gọn ta chỉ loại bỏ các chữ cái trùng lặp liên tiếp, do đó các chữ cái giống nhau liên tiếp vẫn có thể xuất hiện ở chuỗi đầu ra. Tổng quát hóa, ta sẽ xác định ánh xạ B: A $\rightarrow$ Y giữa alignment A và đầu ra Y, trong đó A là 1 alignment hợp lệ của $X = [x_1, x_2, \ldots, x_T]$, $Y = [y_1, y_2, \ldots, y_U]$, $x_i, y_i \in L' = L \cup \{\epsilon\}$, với L là tập hợp tất cả các ký tự có thể nhận dạng. Ánh xạ này thu gọn tất cả

các chữ cái trùng lặp liên tiếp và sau đó loại bỏ tất cả các khoảng trống. Hình 6.10 minh họa quá trình thu gọn này.



Hình 6.10: Các bước thu gọn alignment

Quá trình thu gọn trong CTC là một hàm ánh xạ 1-N, nhiều alignment khác nhau có được ánh xạ về cùng một kết quả đầu ra. Điều này cũng dẫn đến tính chất khác của CTC: độ dài của 1 alignment A không được lớn hơn độ dài đầu ra Y. Hình 6.11 thể hiện một số alignment cho cùng kết quả đầu ra.



Hình 6.11: Nhiều alignment khác nhau có thể cho ra cùng kết quả sau thu gọn

Để huấn luyện một mô hình CTC, ta cần tìm cách tối đa hóa xác suất xảy ra của đầu ra Y khi biết đầu vào X, gọi là $P_{CTC}(Y \mid X)$. Nhưng như đã nói ở trên, một đầu ra Y có thể có nhiều alignment khác nhau, do đó chúng ta cần tính tổng xác suất của tất cả các alignment có thể có

sẽ thu gọn thành Y. Trước hết, ta sẽ tìm hiểu cách tính toán phân phối xác suất cho một alignment A bất kỳ. Bởi vì CTC giả sử các ký tự đầu ra ở mỗi bước thời gian độc lập hoàn toàn với nhau, nghĩa là ký tự đầu ra tại thời gian t, a_t sẽ không phụ thuộc vào bất kỳ ký tự nào trước đó a_i (i < t). Do đó:

$$P_{CTC}(A \mid X) = \prod_{t=1}^{T} p_t(a_t \mid X) \tag{6.10}$$

Sau đó, hàm mất mát cho một cặp đầu vào (X, Y) có thể được tính bằng:

$$P_{CTC}(Y \mid X) = \sum_{A \in \mathcal{A}_{X,Y}} \prod_{t=1}^{T} p_t(a_t \mid X) \tag{6.11}$$

trong đó $A_{X,Y}$ là tất cả các alignment khả thi giữa X và Y, p_t là phân phối xác suất tại thời gian t của mỗi alignment.

Một cách tiếp cận ngây thơ là tính toán xác suất cho từng alignment rồi sau đó cộng lại hoàn toàn không khả thi trong trường hợp có quá nhiều alignment hợp lệ. Do đó, ta thường dùng các giải thuật quy hoạch động để tính toán hàm mất mát CTC, với ý tưởng chình là nếu 2 alignment có cùng đầu ra ở cùng một bước, thì ta có thể kết hợp chúng (Graves et al., 2006).

Ngoài ra, ta có thể thêm ký tự trống ϵ vào trước và sau mỗi ký tự trong Y mà không làm thay đổi kết quả. Ta xét $Z = [\epsilon, y_1, \epsilon, y_2, \ldots, \epsilon, y_U, \epsilon]$, là Y với ký tự ϵ ở đầu, cuối và giữa mỗi ký tự. Gọi α là xác suất của tất cả alignment có thể có ở một nút. Khi đó, $\alpha_{s,t}$ là xác suất của chuỗi $Z_{1:s}$ sau t bước. Ta có thể tính toán giá trị $\alpha_{s,t}$ dựa vào các giá trị $\alpha_{s-2,t-1}; \alpha_{s,t-1}; \alpha_{s,t-1}$ (vì $Z_{1:s}$ chỉ có thể được phát triển từ 3 chuỗi con $Z_{1:s}, Z_{1:s-1}$ hoặc $Z_{1:s-2}$). Lúc này, sẽ xảy ra 2 khả năng:

- Trường hợp 1: Ta không thể bỏ qua ký tự z_{s-1} trong Z nếu z_{s-1} là 1 ký tự có trong Y hoặc z_{s-1} là ký tự trống ϵ giữa 2 ký tự liên tiếp

giống nhau trong Y. Khi đó, chuỗi $Z_{1:s}$ chỉ có thể được phát triển từ 2 chuỗi con $Z_{1:s-1}, Z_{1:s}$.

$$\alpha_{s,t} = (\alpha_{s-1,t-1} + \alpha_{s,t-1}) \cdot p_t(z_s \mid X) \tag{6.12}$$

- Trường hợp 2: Ta có thể bỏ qua ký tự z_{s-1} trong trường hợp $z_{s-1} = \epsilon$ giữa 2 ký tự liền kề khác nhau. Khi đó, chuỗi $Z_{1:s}$ có thể được phát triển từ 3 chuỗi con $Z_{1:s-2}, Z_{1:s-1}, Z_{1:s}$.

$$\alpha_{s,t} = (\alpha_{s-2,t-1} + \alpha_{s-1,t-1} + \alpha_{s,t-1}) \cdot p_t(z_s \mid X) \tag{6.13}$$

Với D là tập dữ liệu huấn luyện cho mô hình, hàm mất mát CTC được định nghĩa như sau:

$$L_{CTC}(Y, X) = -\sum_{(X,Y)\in D} log(P_{CTC}(Y \mid X)) \tag{6.14}$$

Trong quá trình dự đoán, các dự đoán từ tất cả các bước thời gian sẽ được sử dụng nó để tìm chuỗi đầu ra có khả năng nhất ứng với tín hiệu đầu vào:

$$Y^* = \underset{Y}{\operatorname{argmax}}\ P_{CTC}(Y \mid X) \tag{6.15}$$

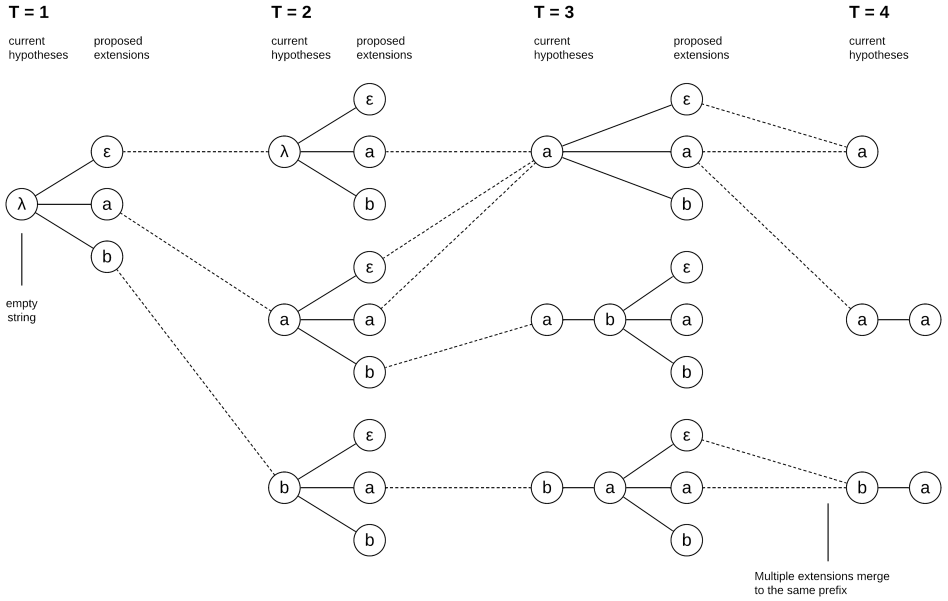
Việc kiểm tra từng alignment cụ thể rồi chọn ra chuỗi đầu ra có khả năng xảy ra nhất là không khả thi vì độ phức tạp tính toán sẽ tăng theo cấp số nhân với độ dài của aligment. Do đó, một heuristic có thể được sử dụng là lấy đầu ra có khả năng xảy ra nhất ở mỗi bước thời gian rồi kết hợp lại tạo thành alignment kết quả. Việc này sẽ giúp ta tìm ra được alignment có xác suất cao nhất:

$$A^* = \underset{A}{\operatorname{argmax}} \prod_{t=1}^{T} p_t(a_t \mid X) \tag{6.16}$$

Sau đó chúng ta sẽ tiến hành thu gọn aligment vừa tìm được để tạo chuỗi đầu ra tương ứng.

Tuy nhiên, alignment có khả năng xảy ra nhất không phản ánh chuỗi đầu ra Y tương ứng cũng có khả năng xảy ra cao nhất. Đó là bởi vì nhiều alignment có thể cho ra cùng kết quả đầu ra, do đó cách tiếp cận trên đôi khi sẽ bỏ sót các đầu ra với tổng xác suất cao hơn nhiều. Hãy xét 1 ví dụ , giả sử các alignment $[a, \epsilon, b]$ và $[a, a, b]$ đều có xác suất thấp hơn alignment $[a, \epsilon, a]$, nhưng tổng 2 xác suất lại lớn hơn xác suất của $[a, \epsilon, a]$. Giải thuật tham lam sẽ cho ra chuỗi đầu ra không chính xác $Y^* = [aa]$ chứ không phải chuỗi chính xác $Y = [ab]$. Để khắc phục điều này, thuật toán cần xét đến thực tế là$[a, \epsilon, b]$ và $[a, a, b]$ đều có thể thu gọn về cùng một đầu ra.

Để khắc phục vấn đề này, chúng ta có thể sử dụng thuật toán beamsearch cho phép giữ lại các alignment ánh xạ đến cùng một đầu ra một cách thông minh. Thuật toán beamsearch sẽ thực hiện tính toán một tập hợp các giả thuyết mới ở mỗi bước, tập hợp giả thuyết mới được tạo ra từ tập hợp trước đó bằng cách mở rộng mỗi giả thuyết với tất cả các ký tự đầu ra có thể có và chỉ giữ lại các giả thuyết có khả năng nhất.



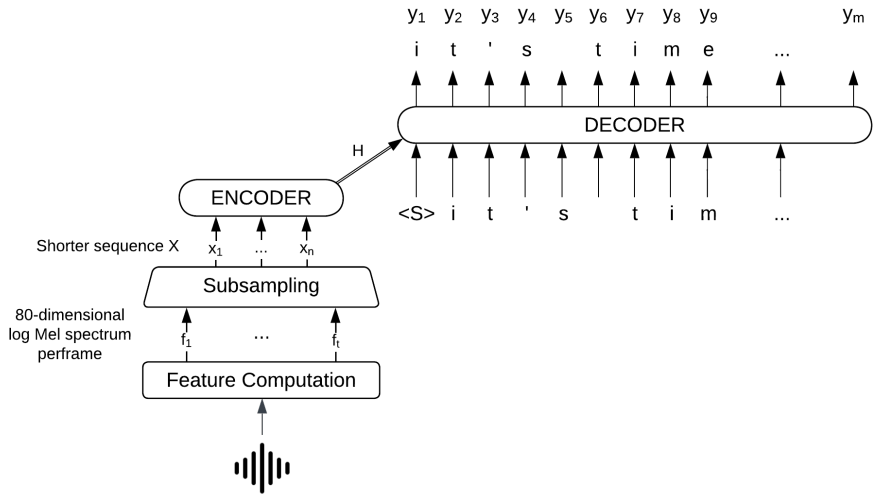
Hình 6.12: Giải thuật beam search

Các mô hình Sequence-to-Sequence

Mặc dù các mô hình CTC hoạt động rất tốt, tuy nhiên CTC lại có giả định chặt chẽ về tính độc lập hoàn toàn giữa các ký tự, khiến cho mô hình đưa ra dự đoán chỉ dựa vào các thông tin hiện tại mà không quan tâm đến những dự đoán trong quá khứ. Và khi đã hoàn thành việc dự đoán ở mỗi bước, mô hình không có cách nào để điều chỉnh các dự đoán đó. Thông thường khi giao tiếp với nhau, chúng ta có thể dựa vào những thông tin đã nghe được để bổ sung hoặc điều chỉnh lại những chỗ chưa nghe rõ hoặc bị nhầm lẫn trước đó nhằm có được thông tin chính xác nhất. Đây là một hạn chế tương đối lớn mà các mô hình nhận dạng giọng nói dựa trên CTC vẫn đang gặp phải.

Để khắc phục hạn chế này, một kiến trúc nhận dạng end-to-end khác là Sequence-to-Sequence đã được phát triển (Graves and Jaitly, 2014). Bài toán nhận dạng giọng nói có thể được tiếp cận tương tự như các bài toán dịch máy hay chuyển đổi văn bản khác, với đầu vào là giọng nói

có thể được xem như một chuỗi tín hiệu liên tục, còn đầu ra là chuỗi văn bản nhận dạng tương ứng. Các mô hình loại này thường được thiết kế theo kiến trúc encoder-decoder, trong đó encoder đảm nhiệm vai trò trích xuất các thông tin âm học theo thời gian từ tín hiệu giọng nói, sau đó decoder sẽ sử dụng các thông tin này phục vụ cho quá trình giải mã văn bản nhận dạng. Mô hình Sequence-to-Sequence cho phép dự đoán tuần tự theo thời gian, và ở mỗi bước dự đoán ta có thể sử dụng toàn bộ thông tin dự đoán trước đó. Nhược điểm chính của mô hình sequence-to-sequence là phải theo dõi các phụ thuộc trình tự dài và nếu văn bản nói quá dài, lên tới hàng chục, hàng trăm từ thì cần một lượng tài nguyên lớn để có thể lưu trữ và xử lý hết được các ký tự đã được dự đoán trong quá khứ. Những hạn chế này có thể được giải quyết một phần bằng cơ chế attention hoặc mã hóa phân cấp.



Hình 6.13: Kiến trúc tổng quát cho mô hình sequence-to-sequence

Hình 6.13 minh họa kiến trúc tổng quát của mô hình sequence-to-sequence, còn được gọi là AED (attention-based encoder decoder) hay LAS (Listen, Attend and Spell) sau hai bài báo đầu tiên áp dụng vào nhận dạng giọng nói (Chorowski et al., 2014, Chan et al., 2015). Đầu vào của mô hình

là chuỗi véc-tơ âm học $F = f_1, f_2, \ldots, f_t$, đầu ra có thể là chữ cái, âm vị, hay đơn vị từ và có dạng $y = (< SOS >, y_1, y_2, \ldots, y_m, < EOS >)$, trong đó <SOS> và <EOS> là hai ký tự đặc biệt thể hiện bắt đầu và kết thúc câu.

Kiến trúc Sequence-to-Sequence rất phù hợp đối với các bài toán mà chuỗi đầu vào và đầu ra có sự khác biệt rõ rệt về độ dài, như đối với giọng nói, chuỗi đặc trưng âm học rất dài được ánh xạ đến chuỗi văn bản có độ dài ngắn hơn rất nhiều. Thông thường, 1 từ đơn gồm trung bình khoảng 5 chữ cái, nhưng nếu nó kéo dài 1 giây, số lượng đặc trưng âm học có thể lên tới 100 (nếu mỗi đặc trưng được trích xuất trong phạm vị mỗi 10ms).

Chính vì sự khác biệt rất lớn này, các mô hình Sequence-to-Sequence thường có thêm 1 giai đoạn là subsampling nhằm rút ngắn chuỗi đặc trưng âm học trước giai đoạn mã hóa. Một giải thuật đơn giản nhất cho quá trình này là low frame rate (Pundak and Sainath, 2016): Tại bước thời gian i, ta xếp chồng f_i, f_{i-1}, f_{i-2} với nhau để tạo thành véc-tơ đặc trưng mới dài hơn gấp 3 lần. Như vậy, thay vì một véc-tơ đặc trưng m chiều sau mỗi 10ms, ta sẽ có một véc-tơ dài hơn với $3m$ chiều sau mỗi 30ms, với độ dài chuỗi chỉ còn $n = \frac{t}{3}$. Hoặc chúng ta có thể sử dụng các lớp convolution để rút ngắn và tổng hợp các thông tin từ chuỗi véc-tơ đặc trưng âm học ban đầu, đồng thời có thể cho mô hình "học" cách rút trích thông tin hiệu quả nhất.

Sau giai đoạn này, kiến trúc của mô hình hoàn toàn tương tự với các mô hình dịch máy hay xử lý văn bản, được xây dựng sử dụng RNN, LSTM hay Transformer.

Mô hình sequence-to-sequence thường được huấn luyện với hàm mất mát là cross-entropy kết hợp với phương pháp teacher forcing như trong các bài toán xử lý ngôn ngữ tự nhiên. Tại bước giải mã thứ i, mục tiêu của mô hình là tối đa hóa xác suất của ký tự mục tiêu y_i:

$$L_{CE} = -\sum_{i=1}^{m} logp(y_i|y_1, \ldots, y_{i-1}, X) \tag{6.17}$$

Trong quá trình suy diễn, xác suất của chuỗi đầu ra Y được ước lượng bằng công thức dưới đây:

$$p(y_1, \ldots, y_n) = \prod_{i=1}^{n} p(y_i|y_1, \ldots, y_{i-1}, X) \tag{6.18}$$

Chuỗi đầu ra có thể được xác định sử dụng giải thuật tham lam hay beamsearch như đã đề cập ở phần trước.

RNN-T

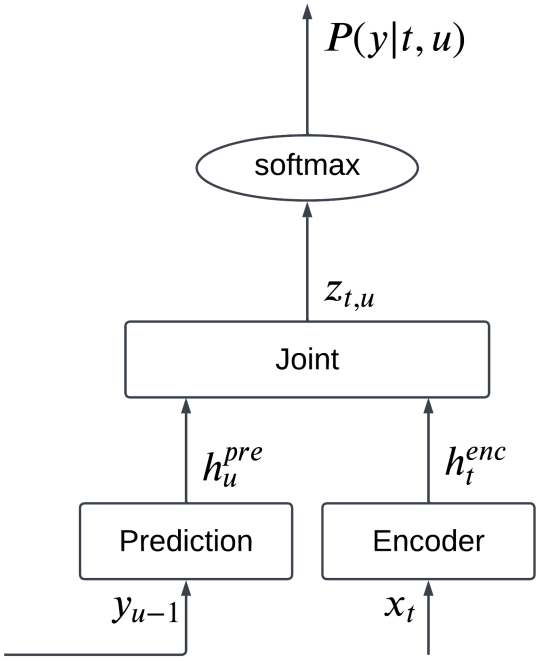
Do giả định về tính độc lập mạnh mẽ trong CTC (giả định rằng đầu ra tại thời điểm t độc lập với đầu ra tại thời điểm t - 1), các mô hình nhận dạng dựa trên CTC không đạt được độ chính xác cao như các mô hình nhận dạng dựa trên encoder-decoder. Tuy nhiên, các mô hình nhận dạng CTC có lợi thế là chúng có tốc độ giải mã nhanh hơn và có thể được sử dụng để nhận dạng trực tuyến. Nhận dạng trực tuyến có nghĩa là nhận dạng các từ liên tục ngay khi người đọc vừa nói thay vì phải đợi đến khi kết thúc để nhận dạng. Việc phát sóng trực tuyến là một ứng dụng quan trọng xuất hiện phổ biến trong thực tế, từ việc ra lệnh tới việc nhập liệu văn bản trực tiếp từ giọng nói, khi chúng ta muốn hệ thống có thể nhận dạng trong khi người dùng vẫn đang nói chuyện. Các mô hình Sequence-to-Sequence dựa trên cơ chế attention đòi hỏi phải thực hiện tính toán trên toàn bộ tín hiệu đầu vào trước để cung cấp thông tin ngữ cảnh trước khi bắt đầu quá trình giải mã. Ngược lại, thuật toán CTC cho phép nhận dạng các chữ cái hay âm vị từ trái sang phải ngay lập tức.

Nếu muốn thực hiện nhận dạng giọng nói trực tuyến, chúng ta cần một cách hiệu quả để cải thiện khả năng nhận dạng của CTC nhằm loại bỏ

giả định độc lập có điều kiện, cho phép nó biết về lịch sử các từ đã được nhận dạng trước đó. Mô hình RNN-Transducer (RNN-T), được thể hiện trong Hình 6.14, là một mô hình như vậy (Battenberg et al., 2017; Graves, 2012). RNN-T gồm 3 thành phần chính: một mô hình âm học (encoder); một thành phần mô hình ngôn ngữ riêng biệt được gọi là bộ dự đoán (predictor) cho phép quan sát và lấy thông tin từ lịch sử nhận dạng trước đó; cuối cùng là bộ kết hợp (joint network) kết hợp thông tin từ mô hình âm học và bộ dự đoán để tính toán phân phối xác suất chung trên không gian từ vựng và âm học. Tại mỗi thời điểm t, bộ mã hóa cho ra một véc-tơ embedding h_t^{enc} tương ứng với tín hiệu đầu vào $x_1, \ldots, x_t$. Bộ dự đoán nhận đầu vào là chuỗi ký tự được nhận dạng trước đó (không tính khoảng trắng) $y_1, \ldots, y_{u-1}$ và xuất ra một véc-tơ embedding h_u^{pred} ở bước u. Sau đó, thông tin trên không gian từ vựng ở thời gian t và bước u có thể được tính toán bởi mạng kết hợp:

$$h_{t,u}^{joint} = f_{joint}(h_t^{enc}, h_u^{pred}) \tag{6.19}$$

Phân phối xác suất trên không gian từ vựng ở thời gian t và bước u được tính bằng một lớp softmax để dự đoán ký tự tiếp theo. Mô hình được huấn luyện bằng giải thuật lan truyền ngược với hàm mục tiêu là tổng xác suất $P(Y|X)$ của tất cả đường dẫn tương ứng với chuỗi mục tiêu.



Hình 6.14: Kiến trúc mô hình RNN-Transducer

6.4 Các phương pháp đánh giá

6.4.1 Tốc độ xử lý

Hệ số thời gian thực (Real-time Factor - RTF) là phương pháp được sử dụng phổ biến nhất để đánh tốc độ xử lý của hệ thống nhận dạng giọng nói. RTF có thể được tính bằng cách sử dụng công thức sau:

$$RTF = \frac{P}{I} \tag{6.20}$$

trong đó P là thời gian hệ thống cần để xử lý đầu vào và I là thời lượng của âm thanh đầu vào. Nếu RTF nhỏ hơn hoặc bằng 1, thì âm thanh đầu vào có thể được xử lý ở "thời gian thực". RTF phụ thuộc nhiều vào cả phần cứng và kiến trúc nhận dạng được sử dụng. RTF không chỉ được

sử dụng trong việc đánh giá tốc độ nhận dạng giọng nói mà còn được sử dụng để đánh giá bất kỳ hệ thống xử lý nào với đầu vào là âm thanh hay video.

6.4.2 Độ chính xác

Phương pháp đánh giá tiêu chuẩn cho hệ thống nhận dạng giọng nói là tỷ lệ lỗi từ (word error rate - WER). Tỷ lệ lỗi từ được đánh giá dựa trên mức độ mà chuỗi từ được hệ thống nhận dạng trả về (chuỗi từ giả thuyết) khác với chuỗi từ tham chiếu, phản ánh mức độ chính xác của hệ thống nhận dạng giọng nói. WER được phát triển dựa trên khoảng cách Levenshtein, đánh giá ở cấp độ từ thay vì cấp độ âm vị.

WER có thể được tính bằng cách căn chỉnh chuỗi từ tham chiếu với chuỗi từ giả thuyết bằng khoảng cách Levenshtein. Ta sẽ biết được số lần thay thế từ, chèn từ và xóa từ tối thiểu cần thiết để ánh xạ giữa 2 chuỗi tham chiếu và giả thuyết. Tỷ lệ lỗi từ sau đó được xác định như sau (lưu ý rằng vì có khả năng chèn từ, tỷ lệ lỗi có thể lớn hơn 100%):

$$WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C} \tag{6.21}$$

Trong đó:

- S (substitution) là số lượng từ cần thay thế (ví dụ từ "nhạt" bị nhận dạng nhầm thành "nhạc")
- D (deletion) là số lượng từ bị xóa đi (ví dụ "mẹ đi đây" bị nhận dạng thành “mẹ đây”)
- I (insertion) là số từ được thêm vào (ví dụ “google” bị nhận dạng thành "gu gồ")
- C (correction) là số từ đã được nhận dạng đúng
- N là tổng số từ trong chuỗi tham chiếu (N=S+D+C)

Chúng ta hãy cùng xét ví dụ sau để hiểu rõ hơn về cách tính toán tỷ lệ lỗi từ:

REF:	mẹ	ĐI	đây	con	ở	nhà	TRÔNG	em NHÉ
HYP:	mẹ		đây	con	ở	nhà THÌ	TRỒNG	em NHÁ
Eval:		D				I	S	S

Ta có thể thấy câu văn trên có 2 phép thế, 1 phép xóa và 1 phép chèn. Do đó, tỷ lệ là lỗi từ sẽ là:

$$WER = \frac{2+1+1}{9} = 33.33\% \tag{6.22}$$

Độ chính xác của từ (word accuracy - WAcc) là một biến thể của WER cũng có thể được sử dụng để đánh giá hiệu suất của ASR. Nó có thể được tính bằng công thức sau:

$$WAcc = 1 - WER = \frac{N-S-D-I}{N} = \frac{C-I}{N} \tag{6.23}$$

Chương 7

Ứng dụng của học sâu trong bài toán tổng hợp tiếng nói

7.1 Giới thiệu

Text-To-Speech (TTS), tức bài toán chuyển văn bản thành giọng nói, là một lĩnh vực nghiên cứu phổ biến trong lĩnh vực ngôn ngữ học (âm thanh và văn bản), học máy với nhiều ứng dụng trong thực tế đời sống và công nghiệp. Phát triển một hệ thống TTS dường như là một nhiệm vụ khó khăn, vì nó đòi hỏi sự hiểu biết về ngôn ngữ và quá trình sinh giọng nói của con người, cũng như chuyên môn trong nhiều lĩnh vực như ngôn ngữ học, âm học, xử lý tín hiệu kỹ thuật số và học máy.

Quá trình phát triển của lĩnh vực TTS có nguồn gốc từ thế kỷ 12. Về mặt hệ thống được xây dựng trên nền tảng máy tính hiện đại, hệ thống tổng hợp giọng nói đầu tiên ra đời vào nửa sau của thế kỷ 20. Các hệ thống này tận dụng các phương pháp cổ điển bao gồm *tổng hợp khớp* (articulatory synthesis), *tổng hợp định hình* (formant synthesis) và *tổng hợp nối* (concatenative synthesis).

Tổng hợp khớp mô phỏng hoạt động của các bộ phận khớp của con người như môi, lưỡi, thanh môn và đường thanh âm chuyển động để tạo

ra tiếng nói. Về lý thuyết, vì phương pháp này mô hình hóa cách con người tạo ra tiếng nói, tổng hợp khớp có thể là phương pháp thành công nhất để tổng hợp tiếng nói. Tuy nhiên, trong thực tế, mô hình hóa các hành vi khớp nối này là vô cùng khó khăn. Chẳng hạn, việc thu thập dữ liệu để mô phỏng bộ khớp nối là rất khó. Kết quả là, tổng hợp khớp tạo ra giọng nói chất lượng thấp hơn so với các phương pháp ra đời sau, tổng hợp định hình và tổng hợp nối.

Tổng hợp nối tạo ra giọng nói bằng cách sử dụng mô hình bộ lọc nguồn (source-filter model) và một bộ các quy tắc. Các nhà ngôn ngữ học tạo ra những quy tắc này để phù hợp với cấu trúc định hình và các đặc điểm phổ khác của lời nói một cách chính xác nhất có thể. Do đặc tính phụ thuộc vào quy tắc này, việc tổng hợp định hình không cần một kho ngữ liệu lớn và sử dụng ít tài nguyên tính toán hơn. Tuy nhiên, bởi vì những quy tắc đó, trong khi khó xác định, không thể bắt chước hoàn toàn giọng nói của con người, giọng nói được tạo ra sẽ nghe kém tự nhiên hơn và có thể chứa các tiếng rè, giật.

Việc nối ghép các đoạn giọng nói được lưu trữ trong cơ sở dữ liệu được sử dụng trong **tổng hợp nối**. Cơ sở dữ liệu thường được tạo thành từ các đơn vị giọng nói được ghi lại bởi diễn viên lồng tiếng, từ toàn bộ câu đến âm tiết. Hệ thống TTS cố gắng nối các đơn vị giọng nói, khớp từng truy vấn và làm mượt kết quả được nối để tạo ra dạng sóng giọng nói. Tổng hợp có nhược điểm là yêu cầu cơ sở dữ liệu lớn bao gồm các bản ghi âm, và giọng nói đầu ra kém tự nhiên và thiếu cảm xúc, mặc dù nó có khả năng bắt chước giọng gốc của diễn viên.

Để giải quyết những thiếu sót của các phương pháp trước đây, **tổng hợp giọng nói tham số thống kê** (statistical parametric speech synthesis) được đề xuất, dự đoán các tham số như phổ, tần số cơ bản và thời lượng cho tổng hợp giọng nói bằng cách sử dụng học máy thống kê. Thay vì trực tiếp tạo ra dạng sóng thông qua quá trình ghép nối, ý tưởng cơ bản là tạo ra các thông số âm thanh cần thiết để tạo ra giọng nói trước tiên, sau đó khôi phục giọng nói từ các thông số âm thanh đã tạo bằng các

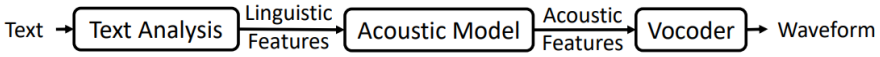
thuật toán. Mô hình hỗn hợp Gaussian và mô hình Markov ẩn đã được sử dụng trong SPSS. Một nhược điểm của SPSS là nó vẫn chưa đủ mạnh để tạo ra giọng nói tự nhiên, có âm thanh của con người.

Với sự tiến bộ nhanh chóng của học sâu, ***TTS dựa trên mạng nơ-ron*** (neural TTS) đã được đề xuất, trong đó mạng nơ-ron (sâu) được sử dụng làm mô hình xương sống để tổng hợp giọng nói. WaveNet (van den Oord et al., 2016), có thể được coi là mô hình neural TTS hiện đại đầu tiên, được đề xuất để tạo ra dạng sóng trực tiếp từ thông tin ngôn ngữ. Các phương pháp khác, chẳng hạn như DeepVoice 1/2 (Arık et al., 2017a; Arık et al., 2017b), tiếp tục sử dụng ba thành phần của tổng hợp tham số thống kê, nhưng thay thế chúng bằng các mô hình dựa trên mạng nơ-ron. Ngoài ra, một số mô hình end-to-end nhất định (ví dụ: Tacotron 1/2 (Shen et al., 2018; Wang et al., 2017), Deep Voice 3 (Ping et al., 2018) và FastSpeech 1/2 (Ren et al., 2021; Ren et al., 2019)) đã được đề xuất dựa trên đơn giản hóa các mô-đun phân tích văn bản và trực tiếp lấy các chuỗi ký tự hoặc âm vị làm đầu vào, cũng như để đơn giản hóa các đặc tính âm học với các phổ âm sắc. ClariNet (Ping et al., 2019), FastSpeech 2s (Ren et al., 2021) và EATS (Donahue et al., 2021) là các ví dụ về hệ thống TTS end-to-end hoàn toàn, trong đó chúng tạo âm thanh trực tiếp từ văn bản.

Ưu điểm của tổng hợp giọng nói dựa trên mạng nơ-ron so với các hệ thống TTS trước đây bao gồm chất lượng giọng nói cao cả về độ dễ nghe, dễ hiểu được và tính tự nhiên, cũng như ít phụ thuộc vào quá trình tiền xử lý của con người và xây dựng các thuộc tính đầu vào. Vì vậy, mục tiếp theo sẽ tập trung vào giới thiệu các hệ thống neural TTS phổ biến và đạt được các kết quả ấn tượng trong lĩnh vực Text-To-Speech.

7.2 Các thành phần chính trong hệ thống Text-To-Speech

Mô-đun phân tích văn bản (text analysis), mô hình âm học (acoustic model) và bộ mã hóa (vocoder) là ba thành phần thiết yếu của hệ thống TTS hiện đại, được minh họa trong Hình 7.1.



Hình 7.1: 3 thành phần chính trong hệ thống TTS (Tan et al., 2021)

7.2.1 Text Analysis

Để tổng hợp giọng nói dễ dàng hơn, phân tích văn bản, còn được gọi là mô-đun frontend trong TTS, chuyển đổi văn bản đầu vào thành các thuộc tính ngôn ngữ, cung cấp thông tin đa dạng, dồi dào về cách phát âm và sự nhịp nhàng. Do khả năng mô hình hóa mạnh của các mô hình neural network, các chuỗi ký tự hoặc âm vị được lấy trực tiếp làm đầu vào để tổng hợp, và do đó vai trò mô-đun phân tích văn bản bị giảm đáng kể. Chuẩn hóa văn bản, phân đoạn từ, gắn nhãn một phần giọng nói, dự đoán sự nhịp nhàng và chuyển đổi từ grapheme sang âm vị là các tác vụ phổ biến trong phân tích văn bản.

Mặc dù các mô hình neural TTS làm đơn giản hóa mô-đun phân tích văn bản, một số yếu tố dự đoán độ nhịp nhàng được lên trên bộ mã hóa văn bản (ký tự hoặc âm vị) trong các mô hình TTS, chẳng hạn như dự đoán cao độ, thời lượng, ngắt cụm từ, hơi thở hoặc các khoảng tạm dừng. Các phương pháp khác để kết hợp các tính năng về độ nhịp nhàng bao gồm 1) bộ mã hóa tham chiếu, học các biểu diễn về độ nhịp nhàng từ giọng đọc tham chiếu; 2) text pre-training, trong đó self-supervised pre-training được sử dụng để học các cách trình bày văn bản tốt với thông tin về độ nhịp nhàng ngầm; và 3) kết hợp thông tin mặt cú pháp bằng cách sử dụng các phương pháp mô hình hóa chuyên dụng như graph neural network.

Sau mô-đun phân tích văn bản ở trên, chúng ta có thể tạo ra các thuộc tính ngôn ngữ và sử dụng chúng làm đầu vào cho các giai đoạn tiếp theo của hệ thống TTS, chẳng hạn như các mô hình âm thanh trong SPSS hoặc vocoder. Ta thường có thể tạo ra các đặc điểm ngôn ngữ bằng cách kết hợp các đầu ra của phân tích văn bản ở nhiều cấp độ, chẳng hạn như cấp độ âm vị, âm tiết, từ, cụm từ và câu.

7.2.2 Acoustic Model

Các mô hình âm học được thiết kế để tạo ra các thuộc tính âm học từ các thuộc tính ngôn ngữ hoặc trực tiếp từ các âm vị hoặc ký tự đầu vào, sau đó được chuyển đổi thành dạng sóng âm bằng cách sử dụng vocoder. Một số mô hình âm học tiêu biểu trong neural TTS dựa trên kiến trúc mạng học sâu sẽ được giới thiệu trong các phần sau.

RNN-based Models

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$
$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

Kiến trúc RNN đã được sử dụng trong các công trình trước đây áp dụng học sâu vào các mô hình âm học, với phổ biến nhất là Tacotron (Wang et al., 2017), sử dụng encoder-attention decoder framework và lấy các ký tự chữ cái hoặc âm vị làm đầu vào và đầu ra là các biểu đồ quang phổ tuyến tính (linear-spectrograms), đồng thời sử dụng thuật toán Griffin-Lim để tạo ra dạng sóng âm giọng nói. Sử dụng thêm một mô hình WaveNet, Tacotron 2 (Shen et al., 2018) được đề xuất để tạo ra các phổ mel và chuyển đổi các phổ mel sang dạng sóng âm. Tacotron 2 vượt trội hơn các phương pháp tiếp cận trước đây như TTS nối, TTS tham số và neural TTS, chẳng hạn như Tacotron, về chất lượng giọng nói. Theo sau đó, một loạt các mô hình cải tiến Tacotron theo nhiều cách khác nhau đã được đề xuất.

CNN-based Models

Mạng nơ-ron tích chập là một biến thể của feed-forward ANN của các perceptron nhiều lớp sử dụng rất ít tiền xử lý cho đầu vào của mạng. Đầu vào phải có kích thước nhất định và CNN sử dụng mẫu liên kết giữa các nơ-ron của nó để tạo ra đầu ra có kích thước cố định. CNN và các kiến trúc mạng học sâu phát triển dựa trên CCN loại này thường được sử dụng trong lĩnh vực xử lý hình ảnh và nhận dạng giọng nói.

Tương tự như hướng tiếp cận sử dụng RNN, một vài công trình đầu nghiên cứu về ứng dụng neural network cho TTS sử dụng kiến trúc CNN cho mạng âm học, bao gồm: DeepVoice (Arık et al., 2017a), DeepVoice 2 (Arık et al., 2017b), DeepVoice 3 (Ping et al., 2018), and ClariNet (Ping et al., 2019).

Transformer-based Models

Transformer (Vaswani et al., 2017) là một mô hình học sâu sử dụng cơ chế self-attention để đánh giá mức độ quan trọng của từng yếu tố trong dữ liệu đầu vào một cách khác nhau. Transformer, giống như mạng nơ-ron hồi quy, được thiết kế để xử lý dữ liệu đầu vào tuần tự như ngôn ngữ tự nhiên cho các tác vụ như dịch và tóm tắt văn bản. Transformer, không giống như RNN, không phải lúc nào cũng xử lý dữ liệu theo cùng một thứ tự. Mặt khác, cơ chế self-attention cung cấp ngữ cảnh cho bất kỳ vị trí nào trong chuỗi đầu vào. Ví dụ: nếu dữ liệu đầu vào là một câu ngôn ngữ tự nhiên, Transformer không cần xử lý phần đầu của câu trước phần cuối cùng. Thay vào đó, nó phát hiện ngữ cảnh cung cấp cho mỗi từ trong cụm từ đang xử lý. Vì đặc điểm này cho phép mức độ song song hóa cao hơn so với RNN, nên thời gian huấn luyện mô hình được giảm xuống đáng kể.

TransformerTTS (N. Li et al., 2019) tạo ra quang phổ mel từ các âm vị bằng cách sử dụng kiến trúc encoder-attention-decoder dựa trên Transformer (Vaswani et al., 2017). Họ cho rằng kiến trúc encoder-attention-decoder dựa trên RNN như Tacotron 2 gặp phải hai vấn đề: 1) Do tính

chất lặp lại, cả bộ mã hóa và bộ giải mã dựa trên RNN không thể được huấn luyện song song và bộ mã hóa dựa trên RNN không thể được song song trong quá trình sử dụng, giảm độ hiệu quả của cả quá trình huấn luyện và quá trình sử dụng. 2) RNN không hiệu quả trong việc mô phỏng sự phụ thuộc dài trong chuỗi văn bản và giọng nói vì chúng thường rất dài. TransformerTTS sử dụng cấu trúc mô hình cốt lõi của Transformer để giải quyết những vấn đề này. FastSpeech sử dụng mạng feed-forward Transformer để tạo ra các quang phổ mel song song, tăng tốc độ trong quá trình sử dụng. Nó cũng sử dụng một bộ điều chỉnh độ dài (length regulator) để thu hẹp sự khác biệt về độ dài giữa các chuỗi âm vị và biểu đồ mel-phổ. Leng regulator sử dụng một công cụ dự báo thời lượng để dự báo thời lượng của từng âm vị, sau đó mở rộng chuỗi ẩn của âm vị để phù hợp với độ dài của chuỗi quang phổ mel và cho phép quá trình tạo được diễn ra song song. Đối với chất lượng và sự nhịp nhàng của giọng nói được tạo ra, việc biết cách lấy nhãn thời lượng để huấn luyện leng regulator là rất quan trọng. FastSpeech 2/2s và FastPitch được giới thiệu như những cách để cải thiện FastSpeech hơn nữa.

Generative Acoustic models

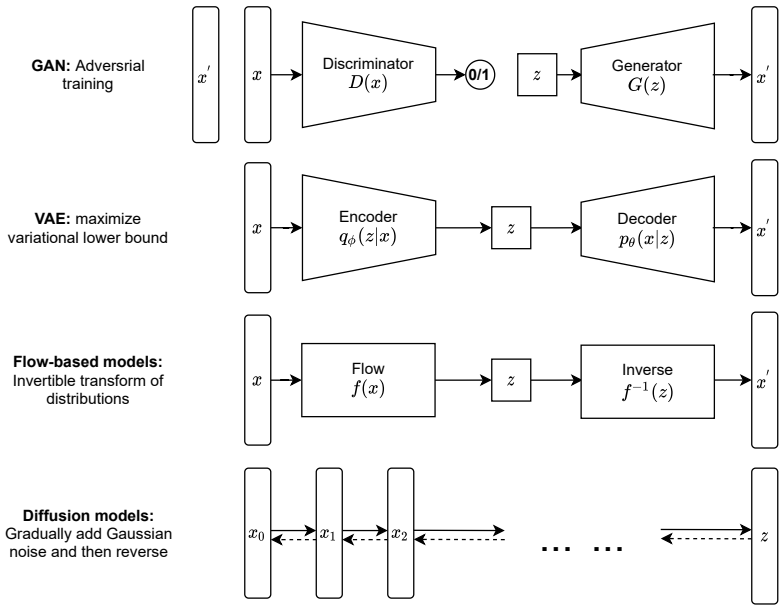
Nhiều loại mô hình sinh khác nhau đã được áp dụng thành công cho mô hình âm thanh trong neural TTS. Một số ví dụ:

- Mô hình dựa trên GAN: *Generative Adversarial Network* (GAN) (Goodfellow et al., 2014) là một cách để tạo mô hình dữ liệu, đây là một bài toán học không có giám sát, như một bài toán học có giám sát. Discriminator model học cách cho biết sự khác biệt giữa dữ liệu thực tế và các mẫu không có thật được tạo ra bởi mô hình sinh. Trong khi chơi trò chơi minimax, hai mô hình sẽ được huấn luyện. GAN được sử dụng trong TTS-Styification (Ma et al., 2019) và Multi-SpectroGAN (S.-H. Lee et al., 2021).
- Mô hình dựa trên VAE: Bằng cách tối đa hóa evidential lower bound, *variational autoencoders* (VAE) (Kingma & Welling, 2014)

tối ưu hóa không rõ ràng giá trị log-likelihood của dữ liệu (ELBO). GMVAE-Tacotron (Hsu et al., 2019) và BVAE-TTS (Y. Lee et al., 2021) đều dựa trên VAE.

- Mô hình flow-based: Một loạt các phép biến đổi có thể đảo ngược được sử dụng để xây dựng một mô hình sinh dựa trên flow. Bởi vì mô hình học trực tiếp phân phối dữ liệu $p(x)$, nên hàm mất mát chính là negative log-likelihood. Flow-TTS (Miao et al., 2020) và Glow-TTS (Kim et al., 2020) đều dựa trên mô hình flow-based.

- Các mô hình diffusion-based: Lấy cảm hứng từ nhiệt động lực học không cân bằng (non-equlibrium thermodynamic). Chúng tạo ra một chuỗi Markov các bước khuếch tán để dần dần đưa giá trị nhiễu ngẫu nhiên vào dữ liệu, sau đó học cách đảo ngược quá trình khuếch tán để tạo ra các mẫu dữ liệu mong muốn từ nhiễu. Mô hình diffusion-based, không giống như VAE hoặc mô hình flow-based, được huấn luyện bằng cách sử dụng phương pháp tiếp cận tập hợp và biến ẩn có số chiều lớn (giống như dữ liệu gốc ban đầu). Diff-TTS (Jeong et al., 2021), Grad-TTS (Popov et al., 2021) và PriorGrad (S.-g. Lee et al., 2022) đều dựa trên mô hình diffusion-based.



Hình 7.2: Tổng quát các hướng tiếp cận khác nhau của mạng sinh[1]

Hình 7.2 miêu tả luồng dữ liệu của các kiến trúc mạng sinh.

7.2.3 Vocoder

Các bộ vocoder tạo ra sóng âm từ các thuộc tính ngôn ngữ hoặc các thuộc tính âm học. Vì dạng sóng âm giọng nói của con người rất dài, việc tạo dạng sóng tự động hồi quy mất nhiều thời gian. Do đó, các mô hình sinh, như minh họa ở trên, được sử dụng trong việc tạo ra dạng sóng.

Autoregressive Vocoders: WaveNet (van den Oord et al., 2016) là bộ mã hóa dựa trên mạng nơ-ron đầu tiên tạo ra các điểm sóng âm tự động thông qua tích chập cách khoảng (dilated). WaveNet chỉ phụ thuộc vào việc huấn luyện từ đầu đến cuối và về cơ bản không có kiến thức trước về tín hiệu âm thanh. WaveNet tạo ra các dạng sóng giọng nói có điều kiện dựa trên các đặc điểm ngôn ngữ, cũng như một số dự án tiếp theo

[1]Source: https://lilianweng.github.io/lil-log/2021/07/11/diffusion-models.html

sử dụng WaveNet như một bộ mã hóa, trong khi WaveNet có thể chỉ đơn giản được mở rộng để sinh quang phổ tuyến tính và quang phổ mel. Mặc dù WaveNet mang lại chất lượng thoại tốt nhưng tốc độ suy luận của nó rất kém. Do đó, nhiều nghiên cứu xem xét các vocoder nhẹ và nhanh hơn, chẳng hạn như SampleRNN, Char2Wav và WaveRNN.

Flow-based Vocoders: Các mô hình dựa trên luồng được sử dụng trong neural TTS có thể được phân loại thành hai nhóm dựa trên hai cách tiếp cận riêng biệt: 1) biến đổi tự hồi quy và 2) biến đổi lưỡng cực (ví dụ: WaveGlow, FloWaveNet).

GAN-based Vocoders: rất nhiều vocoder sử dụng GAN để đảm bảo chất lượng âm thanh tạo ra, bao gồm WaveGAN, GAN-TTS và HiFi-GAN.

Diffusion-based Vocoders: DiffWave (Kong et al., 2021), WaveGrad (Chen et al., 2021) và PriorGrad (S.-g. Lee et al., 2022) là một số nghiên cứu gần đây sử dụng mô hình xác suất khuếch tán (DDPM hoặc Diffusion) (Hsu et al., 2019) cho các vocoder. Ý tưởng cơ bản là sử dụng quá trình khuếch tán thuận và nghịch để hình thành ánh xạ giữa dữ liệu và các phân bố ẩn: trong quá trình khuếch tán, mẫu dữ liệu dạng sóng dần dần được thêm vào một số giá trị nhiễu ngẫu nhiên cho đến khi nó trở thành một giá trị nhiễu theo phân phối Gaussian; trong quá trình ngược lại, nhiễu Gaussian ngẫu nhiên được biến đổi dần dần thành mẫu dữ liệu dạng sóng. Diffusion-based vocoder có thể tạo ra giọng nói với chất lượng giọng nói tuyệt vời, nhưng do quá trình lặp đi lặp lại kéo dài, chúng có tốc độ suy luận khá chậm. Do đó, một số nghiên cứu về mô hình khuếch tán tập trung vào việc giảm thời gian sử dụng trong khi vẫn giữ được chất lượng sinh giọng nói.

7.3 Các hệ thống Text-To-Speech cho Tiếng Việt

Trong (Ha et al., 2010), VOS, một hệ thống tổng hợp giọng nói dựa trên ghép nối, được giới thiệu. Nhóm tác giả sử dụng âm tiết làm đơn vị tổng hợp cơ bản cho các hệ thống TTS tiếng Việt. Nhược điểm chính của phương pháp này là quá trình tổng hợp lựa chọn đơn vị đòi hỏi một lượng đáng kể lưu trữ cơ sở dữ liệu giọng nói. Hơn nữa, ngữ điệu và nhịp điệu của âm thanh do mô hình sinh ra đạt được chất lượng kém.

VTed (Trang et al., 2013), một hệ thống TTS Tiếng Việt được phát triển dựa trên nền tảng Mary TTS, đã sử dụng phương pháp SPSS với kiến trúc dựa trên mô hình Markov ẩn. Kết quả của mô hình này cho thấy sự cần thiết của việc phải nghiên cứu sâu hơn về cấp độ âm để nâng cao khả năng tổng hợp tự động giọng nói của tiếng Việt.

Trong (Lam et al., 2019), nhóm "ZALO" đã sử dụng học sâu để cải thiện hệ thống TTS của họ. Hệ thống của họ dựa trên Tacotron 2 và WaveGlow. Khi so sánh với phương pháp SPSS, kết quả thu được của phương pháp này vượt trội hơn hẳn. Tuy nhiên, tập dữ liệu mà họ sử dụng để đào tạo mô hình đã không được công khai.

Trong VLSP 2020 [2], giọng nói tổng hợp (T. T. T. Nguyen et al., 2020) của đội giỏi nhất đã được sinh bằng Tacotron 2 và Hifigan đạt 89,3% so với tiếng nói của con người về mặt tự nhiên. Tuy nhiên, họ vẫn gặp vấn đề lớn với những câu dài, đặc biệt là cách phát âm không đúng chuẩn và cách phát âm không tốt của các từ vay, mượn.

Hệ thống TTS với mô hình âm học Tacotron 2 được coi là phương pháp học sâu cổ điển trong lĩnh vực TTS. DCA-Tacotron 2 (Battenberg et al., 2020) là phiên bản cải tiến của Tacotron 2 sử dụng cơ chế location-relative attention mới có tên là Dynamic Convolution Attention để giải quyết các vấn đề về căn chỉnh giữa văn bản và giọng nói. Mặc dù có

[2] https://vlsp.org.vn/vlsp2020/eval

những cải tiến, DCA-Tacotron 2 vẫn không thể giải quyết việc tạo ra các văn bản dài. Do đó, các mô hình tổng hợp mới nhất được dự đoán vẫn sẽ hoạt động tốt hơn.

Theo (Popov et al., 2021), hệ thống TTS sử dụng Grad-TTS làm mô hình âm học có thể giải quyết tất cả các vấn đề được mô tả ở trên (giọng nói không tự nhiên, cần cơ sở dữ liệu rất lớn, thiếu cảm xúc và không có khả năng xử lý các câu dài). Mặt khác, Grad-TTS gần đây đã hoạt động đặc biệt tốt trên các bộ dữ liệu cho ngôn ngữ tiếng Anh. Tuy nhiên, khả năng hoạt động tốt trên bộ dữ liệu tiếng Việt của Grad-TTS vẫn còn là một câu hỏi mở, vì tiếng Việt được coi là một ngôn ngữ có nhiều thách thức trong bài toán TTS do những đặc điểm khác biệt đáng kể so với tiếng Anh.

Phụ lục A

Bảng các thuật ngữ

Thuật ngữ	**Ý nghĩa**
Acoustic Model	Mô hình âm học
Activation	Kích hoạt
Addictive attention	Cơ chế chú ý tổng hợp từ xung quanh
Alignment	Căn chỉnh
Attention	Cơ chế chú ý
Attention head	Các đầu ra của cơ chế chú ý
Attention score	Trọng số để tính tổng các thành phần trong cơ chế attention
Autoregressive	Tính chất tự hồi quy
Aggregate	Kết hợp
Batch	Bó
Beam search	Tìm kiếm theo chùm
Cacbon	Nguyên tố Cacbon
Clasifier	Bộ phân loại
Collapse	Sụp đổ
Connected components	Những thành phần có liên kết
Connectionist Temporal Classification	Phân loại theo thời gian kết nối
Context	Ngữ cảnh
Computational graph	Đồ thị tính toán
Convolutional Neural Network	Mạng nơ-ron tích chập
Cross-attention	Cơ chế chú ý chéo
Dataset	Tập dữ liệu
Decoder	Bộ giải mã
Deep Belief Network	Mạng niềm tin sâu
Deep Boltzmann Machine	Máy Boltzmann sâu
Deep Neural Network	Mạng nơ-ron sâu

Dependency parsing	Phân tích sự phụ thuộc về cú pháp
Diffusion	Quá trình khuếch tán
Distribution	Phân phối
Downstream task	Nhiệm vụ chính
Embedding matrix	Ma trận nhúng
Embedding space	Không gian nhúng
Embedding vector	Véc-tơ nhúng
Encoder	Bộ mã hóa
Error	Độ lỗi
Exploding gradient	Bùng nổ độ dốc
Feed-forward	Lan truyền tuyến tính
Finetuning	Hiệu chỉnh
Fully-connected	Kết nối đầy đủ
Gated Graph Neural Networks	Mạng nơ-ron đồ thị có cổng
Gated Recurrent Neural Networks	Mạng nơ-ron hồi quy có cổng
Gated Recurrent Unit	Nút hồi tiếp có cổng
Gaussian Mixture Mode	Mô hình hỗn hợp Gaussian
Generative adversarial network	Mạng nơ-ron sinh
Geometric Deep Learning	Học sâu hình học
Gradient	Độ dốc
Gradient Descent	Suy giảm độ dốc
GraphSAGE	Mạng nơ-ron đồ thị lấy mẫu và kết hợp
Graph Attention Networks	Mạng nơ-ron đồ thị có cơ chế chú ý
Graph Convolution Networks	Mạng nơ-ron đồ thị tích chập
Graph Neural Networks	Mạng nơ-ron đồ thị
Head	Đầu (đầu danh sách, đầu mũi tên,...)
Hidden Markov Model	Mô hình Markov ẩn

Hydrogen	Nguyên tố Hi-đrô
Hypothesis	Giả thuyết
Inductive bias	Thiên kiến quy nạp
Input	Đầu vào
Key	Khóa
Language inference	Suy luận ngôn ngữ
Language Model	Mô hình ngôn ngữ
Latent space	Không gian tiềm ẩn
Latent variable	Biến tiềm ẩn
Layer	Lớp
Linear transformation	Biến đổi tuyến tính
Line graph (Edge graph)	Đồ thị đường hay đồ thị cạnh
Long Short Term Memory	Bộ nhớ ngắn dài hạn
Loss function (Cost function)	Hàm mất mát
Mask	Mặt nạ
Mask language modelling	Mô hình hóa ngôn ngữ bị thiếu
Maximum Likelihood Estimation	Ước tính hợp lý tối đa
Mesage Passing Neural Networks	Mạng nơ-ron truyền tin
Multihop	Di chuyển nhiều bước xa khỏi vị trí ban đầu
Multilayer Perceptron	Mạng nơ-ron nhiều lớp kết nối đầy đủ
Named-entity recognition	Nhận dạng thực thể được đặt tên
Negative Sampling	Lấy mẫu âm tính
Next sentence prediction	Dự đoán câu văn tiếp theo
Non-saturating	Không bão hòa
Non-teacher forcing	Kỹ thuật không sử dụng nhãn mà sử dụng kết quả dự báo tại thời điểm trước trong quá hình huấn luyện các mô hình sequence-to-sequence

Normalization	Lớp chuẩn hóa
Output	Đầu ra
Oxygen	Nguyên tố Oxy
Part-of-Speech tagging	Gán nhãn từ loại
Permutation equivariance	Hoán vị tương đương
Permutation invariance	Hoán vị bất biến
Pooling	Tổng hợp
Positional encoding	Mã hóa thông tin vị trí
Pre-training	Tiền huấn luyện, thường là trên dữ liệu lớn
Pretrained model	Mô hình đã được huấn luyện trước
Pronunciation Model	Mô hình phát âm
Query	Truy vấn
Question answering	Bài toán trả lời câu hỏi
Random walk	Bước đi ngẫu nhiên
Receptive field	Trường tiếp nhận xung quanh
Recurrent Neural Networks	Mạng nơ-ron hồi quy
Regressor	Bộ hồi quy
Residual connection	Kết nối dư thừa
Restricted Boltzmann Machine	Máy Boltzmann hạn chế
Scaled dot-product attention	Cơ chế chú ý bằng hiệu chỉnh tích vô hướng
Score	Điểm số
Self-attention	Cơ chế tự chú ý đến các thành phần xung quanh
Self-supervised	Học tự giám sát
Semi-supervised	Học bán giám sát
Sequence-to-sequence	Nhận đầu vào là chuỗi và cho ra chuỗi
Sparse graphs	Đồ thị thưa

Spectrogram	Biểu diễn quang phổ của âm thanh
Speech to Text	Chuyển đổi giọng nói thành văn bản
State-of-the-art	Hiện đại nhất, mới nhất
Stochatic Gradient Descent	Suy giảm độ dốc ngẫu nhiên
Teacher forcing	Kỹ thuật sử dụng nhãn trong quá hình huấn luyện các mô hình sequence-to-sequence
Text-to-Speech	Nhận dạng văn bản thành giọng nói
Training	Quá trình huấn luyện
Testing	Quá trình kiểm thử
Token	Cụm ký tự liên tục
Transformer	Mô hình ngôn ngữ Transformer
Value	Giá trị
Vanishing Gradient	Biến mất độ dốc
Variational Autoencoders	Bộ mã hóa tự động có biến ẩn
Vector	Véc-tơ
Vocoder	Mô hình chuyển hóa từ quang phổ sang sóng âm

Bibliography

Anusuya, M. A., & Katti, S. K. (2011). Front end analysis of speech recognition: A review. *I. J. Speech Technology*, *14*(2), 99–145.

Arık, S. Ö., Chrzanowski, M., Coates, A., Diamos, G., Gibiansky, A., Kang, Y., Li, X., Miller, J., Ng, A., Raiman, J., Sengupta, S., & Shoeybi, M. (2017a). Deep voice: Real-time neural text-to-speech. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning* (pp. 195–204). PMLR.

Arık, S. Ö., Diamos, G., Gibiansky, A., Miller, J., Peng, K., Ping, W., Raiman, J., & Zhou, Y. (2017b). Deep voice 2: Multi-speaker neural text-to-speech. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2966–2974.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bahdanau, D., Cho, K., & Bengio, Y. (2015a). Neural machine translation by jointly learning to align and translate. In Y. Bengio & Y. LeCun (Eds.), *3rd international conference on learning representations, ICLR 2015, san diego, ca, usa, may 7-9, 2015, conference track proceedings*.

Bahdanau, D., Cho, K., & Bengio, Y. (2015b). Neural machine translation by jointly learning to align and translate. In Y. Bengio & Y. LeCun (Eds.), *3rd international conference on learning representations, ICLR 2015, san diego, ca, usa, may 7-9, 2015, conference track proceedings*.

Battenberg, E., Chen, J., Child, R., Coates, A., Gaur, Y., Li, Y., Liu, H., Satheesh, S., Seetapun, D., Sriram, A., & Zhu, Z. (2017). Exploring neural transducers for end-to-end speech recognition. *CoRR*, *abs/1707.07413*.

Battenberg, E., Skerry-Ryan, R., Mariooryad, S., Stanton, D., Kao, D., Shannon, M., & Bagby, T. (2020). Location-relative attention mechanisms for robust long-form speech synthesis. *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6194–6198.

Baum, L. E., & Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, *37*(6), 1554–1563.

Bronstein, M. M., Bruna, J., Cohen, T., & Veličković, P. (2021). *Geometric deep learning: Grids, groups, graphs, geodesics, and gauges*. arXiv.

Chan, W., Jaitly, N., Le, Q. V., & Vinyals, O. (2015). Listen, attend and spell. *CoRR*, *abs/1508.01211*.

Chen, N., Zhang, Y., Zen, H., Weiss, R. J., Norouzi, M., & Chan, W. (2021). Wavegrad: Estimating gradients for waveform generation. *International Conference on Learning Representations*.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1724–1734.

Chorowski, J., Bahdanau, D., Cho, K., & Bengio, Y. (2014). End-to-end continuous speech recognition using attention-based recurrent nn: First results. *CoRR*, *abs/1412.1602*.

Dahl, G., Yu, D., Deng, L., & Acero, A. (2012). Context-dependent pretrained deep neural networks for large-vocabulary speech recognition. *Trans. Audio, Speech and Lang. Proc.*, *20*(1), 30–42.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q. V., & Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a

fixed-length context (A. Korhonen, D. R. Traum, & L. Màrquez, Eds.), 2978–2988.

David, E. E., & Selfridge, O. G. (1962). Eyes and ears for computers. *Proceedings of the IRE (Institute of Radio Engineers)*, *50*, 1093–1101.

Davis, S. B., & Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *28*(4).

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding (J. Burstein, C. Doran, & T. Solorio, Eds.).

Donahue, J., Dieleman, S., Binkowski, M., Elsen, E., & Simonyan, K. (2021). End-to-end adversarial text-to-speech. *International Conference on Learning Representations*.

Dwivedi, S. K., & Singh, V. (2013). Research and reviews in question answering system. *Procedia Technology*, *10*, 417–424.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, 1263–1272.

Goldberg, Y., & Levy, O. (2014). Word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, & K. Weinberger (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc.

Gori, M., Monfardini, G., & Scarselli, F. (2005). A new model for learning in graph domains. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, *2*, 729–734 vol. 2.

Graves, A. (2012). Sequence transduction with recurrent neural networks. *CoRR*, *abs/1211.3711*.

Graves, A., Fernández, S., Gomez, F. J., & Schmidhuber, J. (2006). Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In W. W. Cohen & A. W. Moore (Eds.), *Icml* (pp. 369–376). ACM.

Graves, A., & Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. *32*, 1764–1772.

Grover, A., & Leskovec, J. (2016). Node2vec: Scalable feature learning for networks. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 855–864.

Ha, V. Q. D., Tuan, N. M., Nam, C. X., Nhut, P. M., & Quan, V. H. (2010). VOS: The corpus-based etnamese text-to-speech system. *Journal of Research and Development on Information and Communication Technology.*

Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc.

Han, X., Jia, M., Chang, Y., Li, Y., & Wu, S. (2022). Directed message passing neural network (d-mpnn) with graph edge attention (gea) for property prediction of biofuel-relevant species. *Energy and AI*, *10*, 100201.

Han, X., Zhang, Z., Ding, N., Gu, Y., Liu, X., Huo, Y., Qiu, J., Yao, Y., Zhang, A., Zhang, L., et al. (2021). Pre-trained models: Past, present and future. *AI Open*, *2*, 225–250.

Hannun, A. (2017). Sequence modeling with ctc. *Distill.*

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., & Ng, A. Y. (2014). Deep speech: Scaling up end-to-end speech recognition.

Harary, F., & Norman, R. Z. (1960). Some properties of line digraphs. *Rendiconti del Circolo Matematico di Palermo*, *9*(2), 161–168.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hinton, G., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, *18*, 1527–54.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *6*(02), 107–116.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hsu, W.-N., Zhang, Y., Weiss, R., Zen, H., Wu, Y., Cao, Y., & Wang, Y. (2019). Hierarchical generative modeling for controllable speech synthesis. *Proceedings of International Conference on Learning Representations (ICLR)*.

Introducing ChatGPT [Accessed: 2023-03-01]. (2023). https://openai.com/blog/chatgpt

Jeong, M., Kim, H., Cheon, S. J., Choi, B. J., & Kim, N. S. (2021). Diff-TTS: A Denoising Diffusion Model for Text-to-Speech. *Proc. Interspeech 2021*, 3605–3609.

Jo, J., Kwak, B., Lee, B., & Yoon, S. (2022). Flexible dual-branched message-passing neural network for a molecular property prediction. *ACS Omega*, *7*(5), 4234–4244.

Jurafsky, D., & Martin, J. H. (2020). *Speech and language processing : An introduction to natural language processing, computational linguistics, and speech recognition*. Pearson Prentice Hall.

Kim, J., Kim, S., Kong, J., & Yoon, S. (2020). Glow-tts: A generative flow for text-to-speech via monotonic alignment search. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 8067–8077). Curran Associates, Inc.

Kingma, D. P., & Welling, M. (2014). Auto-encoding variational bayes. *Proceedings of International Conference on Learning Representations (ICLR).*

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.*

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Skip-thought vectors. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems.* Curran Associates, Inc.

Kong, Z., Ping, W., Huang, J., Zhao, K., & Catanzaro, B. (2021). Diffwave: A versatile diffusion model for audio synthesis. *International Conference on Learning Representations.*

Lam, V., Phan, K., Dinh, T., Khuong, D., & Nguyen, Q. (2019). Development of zalo vietnamese text-to-speech for vlsp 2019.

Lee, S.-g., Kim, H., Shin, C., Tan, X., Liu, C., Meng, Q., Qin, T., Chen, W., Yoon, S., & Liu, T.-Y. (2022). Priorgrad: Improving conditional denoising diffusion models with data-dependent adaptive prior. *International Conference on Learning Representations.*

Lee, S.-H., Yoon, H.-W., Noh, H.-R., Kim, J.-H., & Lee, S.-W. (2021). Multi-spectrogan: High-diversity and high-fidelity spectrogram generation with adversarial style combination for speech synthesis. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(14), 13198–13206.

Lee, Y., Shin, J., & Jung, K. (2021). Bidirectional variational inference for non-autoregressive text-to-speech. *International Conference on Learning Representations.*

Li, J., Deng, L., Gong, Y., & Haeb-Umbach, R. (2014). An overview of noise-robust automatic speech recognition (IEEE/ACM Transactions on Audio, Speech, and Language Processing). *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *22*, 745.

Li, N., Liu, S., Liu, Y., Zhao, S., & Liu, M. (2019). Neural speech synthesis with transformer network. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 6706–6713.

Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R. S. (2016). Gated graph sequence neural networks. In Y. Bengio & Y. LeCun (Eds.), *4th international conference on learning representations, ICLR 2016, san juan, puerto rico, may 2-4, 2016, conference track proceedings*.

Liu, Y., Zhang, Y., Wang, Y., Hou, F., Yuan, J., Tian, J., Zhang, Y., Shi, Z., Fan, J., & He, Z. (2023). A survey of visual transformers. *IEEE Transactions on Neural Networks and Learning Systems*, 1–21.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692*.

Luong, T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1412–1421.

Ma, S., Mcduff, D., & Song, Y. (2019). A generative adversarial network for style modeling in a text-to-speech system. *International Conference on Learning Representations*.

MacCartney, B. (2009). *Natural language inference*. Stanford University.

Manning, C. D. (2022). Human Language Understanding & Reasoning. *Daedalus*, *151*(2), 127–138.

Miao, C., Liang, S., Chen, M., Ma, J., Wang, S., & Xiao, J. (2020). Flow-tts: A non-autoregressive network for text to speech based on flow. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 7209–7213.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. In Y. Bengio & Y. LeCun (Eds.), *1st international conference on learning repre-*

sentations, ICLR 2013, scottsdale, arizona, usa, may 2-4, 2013, workshop track proceedings.

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets.

Nguyen, D. Q., & Nguyen, A. T. (2020). PhoBERT: Pre-trained language models for Vietnamese. *Findings of the Association for Computational Linguistics: EMNLP 2020*, 1037–1042.

Nguyen, T. T. T., Nguyen, H. K., Pham, Q. M., & Vu, D. M. (2020). Vietnamese text-to-speech shared task VLSP 2020: Remaining problems with state-of-the-art techniques. *Proceedings of the 7th International Workshop on Vietnamese Language and Speech Processing*, 35–39.

Openai [Accessed: 2023-03-01]. (2023). https://openai.com/

Pearson, K. (1905). The Problem of the Random Walk. *Nature*, *72*(1865), 294–294.

Ping, W., Peng, K., & Chen, J. (2019). Clarinet: Parallel wave generation in end-to-end text-to-speech. *International Conference on Learning Representations.*

Ping, W., Peng, K., Gibiansky, A., Arik, S. O., Kannan, A., Narang, S., Raiman, J., & Miller, J. (2018). Deep voice 3: 2000-speaker neural text-to-speech. *International Conference on Learning Representations.*

Popov, V., Vovk, I., Gogoryan, V., Sadekova, T., & Kudinov, M. (2021). Grad-tts: A diffusion probabilistic model for text-to-speech. *International Conference on Machine Learning*, 8599–8608.

Pundak, G., & Sainath, T. N. (2016). Lower frame rate neural network acoustic models. In N. Morgan (Ed.), *Interspeech* (pp. 22–26). ISCA.

Rabiner, L. R., & Juang, B.-H. (1993). *Fundamentals of speech recognition.* Englewood Cliffs, NJ: Prentice Hall.

Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286.

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), 9.

Reiser, P., Neubert, M., Eberhard, A., Torresi, L., Zhou, C., Shao, C., Metni, H., van Hoesel, C., Schopmans, H., Sommer, T., & Friederich, P. (2022). Graph neural networks for materials science and chemistry. *Communications Materials*, *3*(1), 93.

Ren, Y., Hu, C., Tan, X., Qin, T., Zhao, S., Zhao, Z., & Liu, T.-Y. (2021). Fastspeech 2: Fast and high-quality end-to-end text to speech. *International Conference on Learning Representations*.

Ren, Y., Ruan, Y., Tan, X., Qin, T., Zhao, S., Zhao, Z., & Liu, T.-Y. (2019). Fastspeech: Fast, robust and controllable text to speech. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc.

Rumelhart, D. E., & McClelland, J. L. (1987). Information processing in dynamical systems: Foundations of harmony theory. *Parallel distributed processing: Explorations in the microstructure of cognition: Foundations* (pp. 194–281).

Salakhutdinov, R., & Hinton, G. (2009). Deep boltzmann machines. *Proceedings of AISTATS 2009*, *5*, 448–455.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., & Battaglia, P. (2020). Learning to simulate complex physics with graph networks. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (pp. 8459–8468). PMLR.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, *20*(1), 61–80.

Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, *45*(11), 2673–2681.

Shahin, M. A., Ahmed, B., McKechnie, J., Ballard, K. J., & Gutierrez-Osuna, R. (2014). A comparison of gmm-hmm and dnn-hmm based pronunciation verification techniques for use in the assessment of childhood apraxia of speech. In H. Li, H. M. Meng, B. Ma, E. Chng, & L. Xie (Eds.), *Interspeech* (pp. 1583–1587). ISCA.

Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., Chen, Z., Zhang, Y., Wang, Y., Skerry-Ryan, R. J., Saurous, R. A., Agiomyrgiannakis, Y., & Wu, Y. (2018). Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4779–4783.

Shlomi, J., Battaglia, P., & Vlimant, J.-R. (2020). Graph neural networks in particle physics. *Machine Learning: Science and Technology*, *2*(2), 021001.

Stärk, H., Ganea, O., Pattanaik, L., Barzilay, R., & Jaakkola, T. (2022). Equibind: Geometric deep learning for drug binding structure prediction. *International Conference on Machine Learning*, 20503–20521.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

Tachbelie, M. Y., Abulimiti, A., Abate, S. T., & Schultz, T. (2020). Dnn-based speech recognition for globalphone languages. *ICASSP*, 8269–8273.

Tan, X., Qin, T., Soong, F., & Liu, T.-Y. (2021). A survey on neural speech synthesis.

Trang, N., d'Alessandro, C., Rilliard, A., & Tran, D. D. (2013). Hmm-based tts for hanoi vietnamese: Issues in design and evaluation. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2311–2315.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *SSW*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. *International Conference on Learning Representations*.

Wang, Y., Skerry-Ryan, R. J., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., Le, Q. V., Agiomyrgiannakis, Y., Clark, R. A. J., & Saurous, R. A. (2017). Tacotron: Towards end-to-end speech synthesis. *INTERSPEECH*.

Xu, H., Su, H., Siong, C. E., & Li, H. (2014). Semi-supervised training for bottle-neck feature based dnn-hmm hybrid systems. In H. Li, H. M. Meng, B. Ma, E. Chng, & L. Xie (Eds.), *Interspeech* (pp. 2078–2082). ISCA.

Yu, D., & Deng, L. (2015). *Automatic speech recognition: A deep learning approach*. Springer.

Zhang, M., & Chen, Y. (2018). Link prediction based on graph neural networks. *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 5171–5181.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, *1*, 57–81.