

LỜI CẢM ƠN

**KẾ HOẠCH THỰC HIỆN**

**MỤC LỤC**

<b>Danh sách bảng</b>	<b>i</b>
<b>Danh sách hình ảnh</b>	<b>ii</b>
<b>1 Mở đầu</b>	<b>1</b>
1.1 Tính cấp thiết của đề tài . . . . .	1
1.2 Mục tiêu và nhiệm vụ nghiên cứu . . . . .	1
1.3 Cách tiếp cận và phương pháp nghiên cứu . . . . .	1
1.4 Dự kiến kết quả đạt được . . . . .	2
1.5 Bố cục luận văn . . . . .	2
<b>2 Nội dung</b>	<b>3</b>
2.1 Tổng quan về học máy và học sâu . . . . .	3
2.1.1 Khái quát về học máy . . . . .	3
2.1.1.1 Giới thiệu . . . . .	3
2.1.1.2 Các loại học máy . . . . .	3
2.1.1.3 Học có giám sát . . . . .	3
2.1.1.4 Học không giám sát . . . . .	5
2.1.1.5 Học bán giám sát . . . . .	6
2.1.1.6 Học tăng cường . . . . .	6
2.1.1.7 Cách thức hoạt động . . . . .	7
2.1.2 Khái quát về học sâu . . . . .	8
2.1.2.1 Giới thiệu . . . . .	8
2.1.2.2 Cơ sở hình thành . . . . .	8
2.1.2.3 Ưu nhược điểm . . . . .	8
2.2 Mạng neuron nhân tạo . . . . .	9
2.2.1 Mạng neuron nhân tạo . . . . .	9
2.2.1.1 Giới thiệu . . . . .	9
2.2.1.2 Mạng thần kinh nhân tạo một lớp - ( <i>Single-layer Neural Network - Perceptron</i> ) . . . . .	11
2.2.1.3 Hàm kích hoạt - ( <i>Activation Function</i> ) . . . . .	13
2.2.1.4 Hàm kích hoạt Softmax - ( <i>Softmax Activation Function</i> ) . . . . .	15
2.2.1.5 Hàm mất mát - ( <i>Lost Function</i> ) . . . . .	15
2.2.1.6 Mạng thần kinh nhân tạo nhiều lớp - ( <i>Multi-Layer Neural Networks</i> ) . . . . .	16

2.2.1.7	Thuật toán lan truyền ngược - ( <i>Backpropagation Algorithm</i> ) . . . . .	18
2.2.1.8	Các vấn đề gặp phải khi huấn luyện mạng thần kinh . . . . .	28
2.2.2	Mạng thần kinh tích chập - CNN ( <i>Convolutional Neural Network</i> ) . . . . .	29
2.2.2.1	Giới thiệu . . . . .	29
2.2.2.2	Kiến trúc cơ bản . . . . .	30
2.2.2.3	Đệm - ( <i>Padding</i> ) . . . . .	33
2.2.2.4	Bước tiến - ( <i>Strides</i> ) . . . . .	33
2.2.2.5	Lớp ReLU . . . . .	34
2.2.2.6	Gộp - ( <i>Pooling</i> ) . . . . .	34
2.2.2.7	Cách sắp xếp các lớp trong mạng thần kinh tích chập . . . . .	35
2.2.2.8	Lan truyền ngược trong mạng thần kinh tích chập . . . . .	35
2.2.3	Mạng thần kinh hồi quy - RNN ( <i>Recurrent Neural Network</i> ) . . . . .	36
2.2.3.1	Giới thiệu . . . . .	36
2.2.3.2	Phân loại . . . . .	38
2.2.3.3	Kiến trúc . . . . .	38
2.2.3.4	Lan truyền ngược theo thời gian - BPTT ( <i>Backpropagation Through Time</i> ) . . . . .	41
2.2.3.5	Mạng thần kinh hồi quy hai chiều ( <i>Bidirectional Recurrent Neural Network</i> ) . . . . .	42
2.2.3.6	Mạng thần kinh hồi quy nhiều lớp ( <i>Multilayer Recurrent Network</i> ) . . . . .	43
2.2.4	Bộ nhớ dài-ngắn hạn - LSTM ( <i>Long Short-Term Memory</i> ) . . . . .	44
2.2.5	Bộ nhớ tái phát - GRU ( <i>Gated Recurrent Unit</i> ) . . . . .	46
2.3	Xử lý ngôn ngữ tự nhiên . . . . .	47
2.3.1	Xử lý ngôn ngữ tự nhiên . . . . .	47
2.3.1.1	Định nghĩa . . . . .	47
2.3.1.2	Các bước xử lý trong xử lý ngôn ngữ tự nhiên . . . . .	49
2.3.1.3	Một vài ứng dụng của xử lý ngôn ngữ tự nhiên . . . . .	49
2.3.2	Kỹ thuật nhúng từ ( <i>Word embedding</i> ) . . . . .	51
2.3.2.1	Vấn đề đặt ra . . . . .	51
2.3.2.2	Giới thiệu . . . . .	52
2.3.2.3	Một số phương pháp . . . . .	52
2.3.2.4	Mô hình Word2vec . . . . .	52
2.3.2.5	Mô hình GloVe . . . . .	56
2.3.2.6	Mô hình Fasttext . . . . .	57
2.3.3	Ngữ cảnh ( <i>Contextual</i> ) và vai trò trong NLP . . . . .	58

2.3.4	Mô hình Transformer . . . . .	60
2.3.4.1	Giới thiệu . . . . .	60
2.3.4.2	Kiến trúc Transformer . . . . .	61
2.3.4.3	Cơ chế Attention . . . . .	61
2.3.5	Tiếp cận nông và học sâu trong ứng dụng pre-training NLP . . .	63
2.3.5.1	Tiếp cận nông ( <i>Shallow approach</i> ) . . . . .	63
2.3.5.2	Học sâu ( <i>Deep learning</i> ) . . . . .	63
2.3.6	Mô hình BERT ( <i>Bidirectional Encoder Representations from Transformers</i> ) . . . . .	64
2.3.6.1	Giới thiệu BERT . . . . .	64
2.3.6.2	Pre-training BERT . . . . .	65
2.3.6.3	Tinh chỉnh ( <i>fine-tuning</i> ) BERT . . . . .	67
2.3.6.4	Những kiến trúc của BERT . . . . .	67
2.4	Xây dựng mô hình phát hiện từ ngữ độc hại . . . . .	68
2.4.1	Môi trường cài đặt và các thư viện sử dụng . . . . .	68
2.4.1.1	Môi trường cài đặt . . . . .	68
2.4.1.2	Thư viện . . . . .	68
2.4.2	Mô tả tập dữ liệu . . . . .	71
2.4.3	Tiền xử lý dữ liệu . . . . .	71
2.4.3.1	Xử lý các ký tự đặc biệt . . . . .	71
2.4.3.2	Quá trình EDA ( <i>Exploratory Data Analysis</i> ) . . . . .	71
2.4.4	Thiết lập mô hình . . . . .	72
2.4.4.1	Phương pháp đánh giá . . . . .	72
2.4.4.2	Chia dữ liệu . . . . .	72
2.4.4.3	Tokenization . . . . .	72
2.4.4.4	Word Embedding . . . . .	73
2.4.5	Huấn luyện mô hình và đánh giá kết quả . . . . .	73

**DANH SÁCH BẢNG**

2.1	Một số hàm thông dụng và lan truyền xuôi-ngược của nó [11] . . . . .	27
2.2	Một số bộ lọc thường dùng [14] . . . . .	32
2.3	Một số loại RNN [1] . . . . .	39
2.4	Ma trận tần suất đồng thời [6] . . . . .	56
2.5	Xác suất các từ cùng xuất hiện [6] . . . . .	57
2.6	Tập dữ liệu Kaggle - Toxic Comment Classification Challenge . . . . .	77
2.7	Xử lý các ký tự đặc biệt . . . . .	78
2.8	Ví dụ kết quả tokenization . . . . .	78

## DANH SÁCH HÌNH ẢNH

2.1	Học có giám sát . . . . .	4
2.2	Phân loại và hồi quy . . . . .	4
2.3	Học không giám sát . . . . .	5
2.4	Học bán giám sát . . . . .	6
2.5	Học tăng cường . . . . .	7
2.6	Cách hoạt động của mô hình học máy . . . . .	7
2.7	Đồ thị không có chu trình . . . . .	11
2.8	Cấu tạo mạng thần kinh nhân tạo đơn giản . . . . .	11
2.9	Phân biệt tuyến tính . . . . .	13
2.10	Giá trị trước khi kích hoạt và giá trị sau khi kích hoạt . . . . .	13
2.11	Đồ thị các hàm kích hoạt thông dụng . . . . .	14
2.12	Đồ thị các hàm kích hoạt tuyến tính từng phần . . . . .	14
2.13	Mạng thần kinh nhân tạo với lớp softmax làm lớp đầu ra . . . . .	15
2.14	Mạng thần kinh nhiều lớp . . . . .	17
2.15	Mạng thần kinh nhiều lớp với cách biểu diễn dưới dạng ma trận . . . . .	17
2.16	Đồ thị tính toán . . . . .	18
2.17	Mạng thần kinh với $n$ lớp, mỗi lớp có 2 nút . . . . .	19
2.18	Đồ thị tính toán của mạng ở hình 2.17 . . . . .	19
2.19	Áp dụng quy tắc xích trong thần kinh đơn giản . . . . .	20
2.20	Áp dụng quy tắc xích đa biến trong mạng thần kinh phức tạp . . . . .	21
2.21	Đồ thị đạo hàm các hàm kích hoạt thông dụng . . . . .	24
2.22	Hai lớp tuyến tính và kích hoạt nằm xen kẽ nhau . . . . .	26
2.23	Lớp tích chập với $d_p = 1$ . . . . .	30
2.24	Lớp tích chập sử dụng nhiều bộ lọc . . . . .	31
2.25	Đệm nửa và đệm đầy đủ . . . . .	33
2.26	Gộp tối đa . . . . .	34
2.27	Lan truyền ngược của bộ lọc là bộ lọc . . . . .	36
2.28	Mạng hồi quy [11] . . . . .	37
2.29	Kiến trúc của lớp ẩn trong mạng hồi quy . . . . .	37
2.30	Mạng hồi quy trong dự đoán từ [11] . . . . .	40
2.31	Kiến trúc mạng hồi quy hai chiều . . . . .	42
2.32	Kiến trúc mạng hồi quy nhiều lớp . . . . .	43
2.33	Kiến trúc của LSTM . . . . .	45
2.34	Kiến trúc của GRU . . . . .	46
2.35	Minh họa một số ứng dụng của xử lý ngôn ngữ tự nhiên . . . . .	50

2.36	Từ được dự đoán và các từ xung quanh . . . . .	52
2.37	Ví dụ về Word2vec . . . . .	53
2.38	Cách hoạt động của CBOW [9] . . . . .	54
2.39	Cách hoạt động của skip-gram . . . . .	55
2.40	Mô hình fasttext đối với CBOW [15] . . . . .	58
2.41	Mô hình fasttext đối với skip-gram [15] . . . . .	59
2.42	Kiến trúc của mô hình Transformer . . . . .	74
2.43	Phương trình Attention thể hiện dưới dạng ma trận . . . . .	75
2.44	Multi-head Attention minh họa dưới dạng ma trận . . . . .	75
2.45	Sự khác nhau giữa các kiến trúc mô hình pre-trained. BERT sử dụng bidi- rectional Transformer. OpenAI GPT sử dụng Transformer theo hướng trái sang phải. ELMo sử dụng kết hợp đầu ra của hai LSTM được huấn luyện độc lập theo hướng trái sang phải và phải sang trái. Trong ba mô hình này, chỉ có BERT học được biểu diễn của từ ngữ phụ thuộc lẫn nhau vào cả ngữ cảnh trước và sau ở tất cả các layer. . . . .	75
2.46	Biểu diễn đầu vào của BERT: Bao gồm các token embedding, token seg- ment và token position. . . . .	76
2.47	Minh họa việc tinh chỉnh BERT cho các tác vụ khác nhau. . . . .	76



### CHƯƠNG 1: MỞ ĐẦU

#### 1.1 TÍNH CẤP THIẾT CỦA ĐỀ TÀI

Với sự bùng nổ của Internet trong những năm gần đây, các diễn đàn, trang web, mạng xã hội, ... dần trở nên phổ biến và tiếp cận được nhiều hơn tới mọi người. Thông qua các kênh thông tin này, chúng ta có thể tiếp cận và chia sẻ thông tin một cách nhanh chóng hơn bao giờ hết. Các trang web và mạng xã hội này có một lượng lớn người sử dụng từ đa dạng các thành phần xã hội khác nhau. Đồng thời nội dung trên Internet cũng ngày càng trở nên phong phú và khó đoán hơn. Mọi người ngày nay coi mạng xã hội không chỉ là một kênh phương tiện giải trí mà còn là nơi giao lưu và chia sẻ thông tin với nhau. Không chỉ người lớn mới sử dụng Internet và mạng xã hội, trẻ em ngày nay cũng được tiếp cận với Internet từ sớm để phục vụ nhu cầu học tập và giải trí. Tuy nhiên ngoài những lợi ích có thể thấy được, Internet mang lại những nguy cơ không lường trước được cho những em nhỏ, và một trong số đó là những ngôn từ không phù hợp trên Internet, hay còn gọi là ngôn ngữ độc hại.

Chỉ mới gần đây thôi, vào năm 2020, Việt Nam đã lọt vào Top 5 những nước kém văn minh nhất thế giới trên Internet. Điều này chứng tỏ môi trường mạng Việt Nam tiềm ẩn nhiều nguy cơ cho những em nhỏ tiếp cận tới những ngôn ngữ không phù hợp với thuần phong mỹ tục, góp phần làm xấu đi hình ảnh đất nước trong mắt bạn bè quốc tế.

Nhằm góp phần ngăn chặn sự tiêu cực mà ngôn ngữ độc hại đem lại, nhóm chúng tôi xin đề xuất “Tìm hiểu bài toán nhận diện văn bản tiêu cực sử dụng học máy” áp dụng cho tiếng Việt làm đề tài tiểu luận chuyên ngành cho mình.

#### 1.2 MỤC TIÊU VÀ NHIỆM VỤ NGHIÊN CỨU

Mục tiêu: Xây dựng được mô hình phát hiện các từ ngữ độc hại.

Nhiệm vụ:

- Tìm hiểu các kiến trúc mạng neuron và các thuật toán dùng trong xử lý văn bản.
- Tìm hiểu về xử lý ngôn ngữ tự nhiên.
- Tìm hiểu các kỹ thuật nhúng từ (word embedding).
- Tìm hiểu các thư viện, module hỗ trợ học máy và học sâu như Tensorflow, Sklearn, ...

#### 1.3 CÁCH TIẾP CẬN VÀ PHƯƠNG PHÁP NGHIÊN CỨU

Cách tiếp cận:

- Sử dụng các lý thuyết học máy, học sâu và kỹ thuật nhúng từ.
- Sử dụng các thư viện, module hỗ trợ học máy và học sâu như Tensorflow, Sklearn, ...

Phương pháp nghiên cứu:

- Xử lý dữ liệu đầu vào trước khi thực hiện huấn luyện.
- Huấn luyện và đánh giá mô hình dựa trên dữ liệu đầu vào để đánh giá khả năng phát

hiện từ ngữ xấu của mô hình.

### 1.4 DỰ KIẾN KẾT QUẢ ĐẠT ĐƯỢC

Về lý thuyết: Nhóm mong muốn sau khi thực hiện nghiên cứu có thể học hỏi và hiểu sâu về nội dung lý thuyết của bài toán đã nêu. Đồng thời có cơ hội thực hành mô hình học sâu trong quá trình huấn luyện dữ liệu cho bài toán.

Về mặt sản phẩm: Nhóm mong muốn xây dựng được một mô hình có thể phát hiện các ngôn ngữ độc hại với một mức chính xác khả quan và có thể ứng dụng thực tế.

### 1.5 BỐ CỤC LUẬN VĂN

Bố cục bài tiểu luận chuyên ngành được tổ chức như sau:

1. Phần 1. Mở đầu
2. Phần 2. Nội dung: Gồm có 4 chương
  - (a) Chương 1: Tổng quan về học máy và học sâu
    - Khái quát về học máy
    - Khái quát về học sâu
  - (b) Chương 2: Mạng neuron nhân tạo
    - Mạng neuron nhân tạo
    - Mạng neuron tích chập - CNN (Convolutional Neural Network)
    - Mạng neuron hồi quy - RNN (Recurrent Neural Network)
    - Bộ nhớ dài-ngắn hạn - LSTM (Long Short-Term Memory)
    - Bộ nhớ tái phát - GRU (Gated Recurrent Unit)
  - (c) Chương 3: Xử lý ngôn ngữ tự nhiên và kỹ thuật nhúng từ
    - Kỹ thuật nhúng từ (Word embedding)
    - Mô hình Word2vec
    - Mô hình GloVe
    - Mô hình Fasttext
  - (d) Chương 4: Xây dựng mô hình phát hiện từ ngữ độc hại
    - Môi trường cài đặt và các thư viện sử dụng
    - Mô tả tập dữ liệu
    - Tiền xử lý dữ liệu
    - Thiết lập mô hình
    - Huấn luyện mô hình và đánh giá kết quả
3. Phần 3: Kết luận
4. Tài liệu tham khảo

## CHƯƠNG 2: NỘI DUNG

### 2.1 TỔNG QUAN VỀ HỌC MÁY VÀ HỌC SÂU

#### 2.1.1 Khái quát về học máy

##### 2.1.1.1 Giới thiệu

Học máy (*Machine Learning*) là một nhánh của trí tuệ nhân tạo (*AI*) tập trung vào việc tạo ra các thuật toán cho phép máy học từ dữ liệu và các thông tin có trước và tự cải thiện theo thời gian. Machine Learning cho phép máy có thể tự động học từ dữ liệu, cải thiện hiệu suất từ dữ liệu đã học được và tạo ra các dự đoán. Các thuật toán Machine Learning tạo ra các mô hình toán học hỗ trợ việc tạo ra các dự đoán hay quyết định với sự hỗ trợ từ các mẫu dữ liệu có trước hay là dữ liệu học (*training data*).[16]

Là một lĩnh vực công nghệ phát triển nhanh, học máy hiện tại được sử dụng cho rất nhiều lĩnh vực khác nhau, một số có thể kể tới là: nhận diện giọng nói, nhận diện hình ảnh hay hệ thống gợi ý sản phẩm (*recommender system*).

##### 2.1.1.2 Các loại học máy

Để có thể hiểu được cách thức mà học máy hoạt động, trước hết chúng ta cần biết về các phương pháp học máy và thuật toán, dưới đây là một số phương pháp thường dùng[16]:

- Học có giám sát (*Supervised Learning*).
- Học không giám sát (*Unsupervised Learning*).
- Học bán giám sát (*Semi-Supervised Learning*).
- Học tăng cường (*Reinforcement Learning*).

##### 2.1.1.3 Học có giám sát

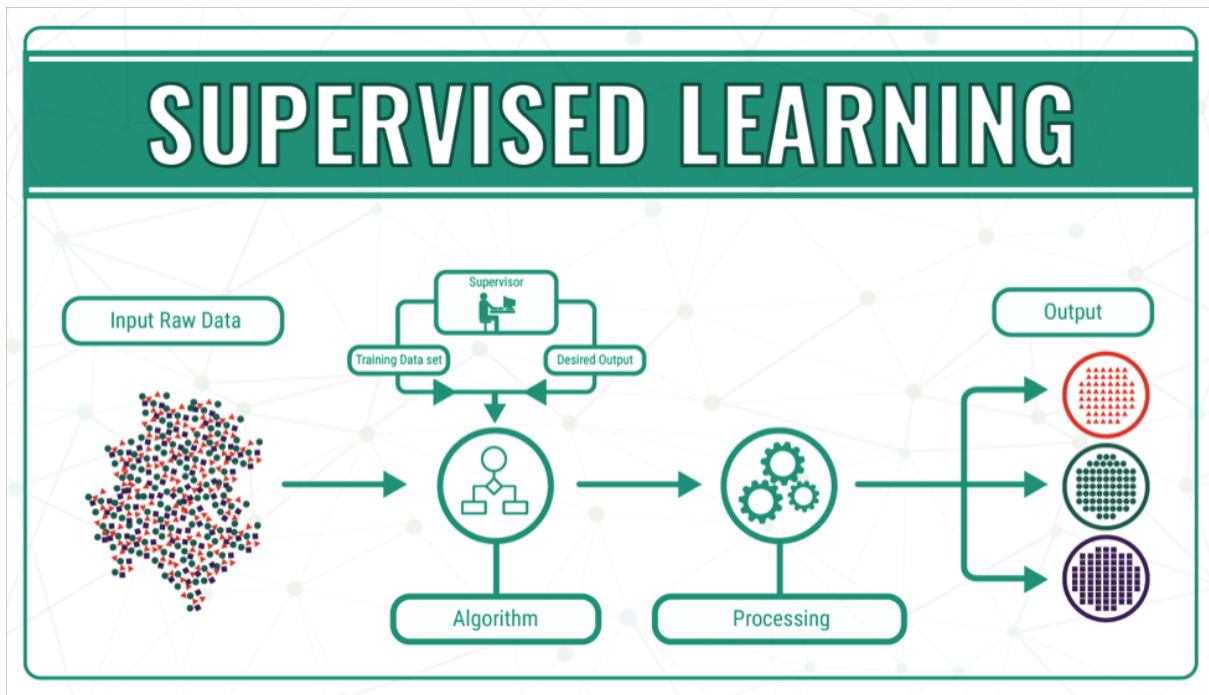
Các thuật toán và mô hình học có giám sát tạo ra các dự đoán dựa trên các dữ liệu đã được đánh nhãn. Mỗi mẫu dữ liệu huấn luyện đều bao gồm dữ liệu đầu vào (*input*) và dữ liệu đầu ra (*output*) tương ứng. Thuật toán học có giám sát phân tích dữ liệu huấn luyện và tạo ra các suy luận - hoặc có thể gọi là các suy đoán có cơ sở khi dự đoán cho các dữ liệu chưa biết trước.[16]

Đây là hướng tiếp cận phổ biến nhất khi nói về học máy, mô hình được “giám sát” bởi chúng cần được học và cung cấp các dữ liệu đã được đánh nhãn từ trước. Dữ liệu được đánh nhãn sẽ cung cấp thông tin về các khuôn mẫu (có thể là hình ảnh, phân loại, etc) để mô hình có thể nhận diện được từ dữ liệu.[16]

Và với phương pháp học có giám sát, chúng ta có hai phương pháp: phân loại (*classification*) và hồi quy (*regression*).

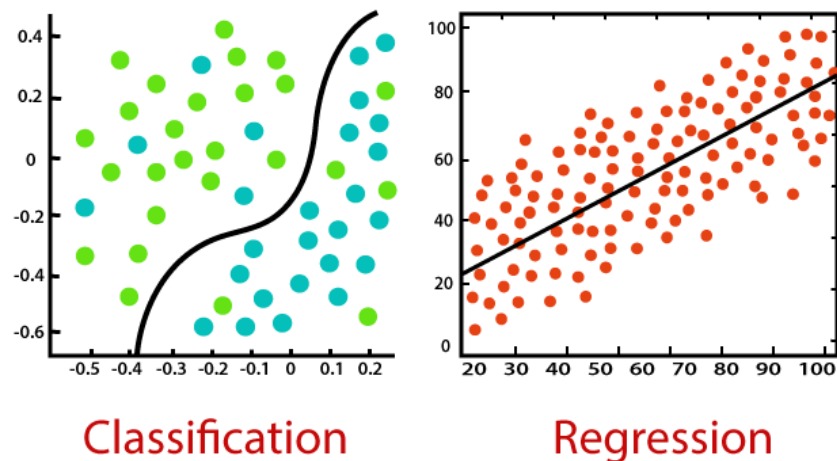
1. Phương pháp hồi quy: Hồi quy tìm sự tương quan giữa biến phụ thuộc và các biến độc lập, từ đó thuật toán hồi quy có thể dự đoán các biến liên tục (*continous variable*) chẳng hạn như chiều cao, cân nặng, ...[13]

Một số thuật toán hồi quy:



Hình 2.1: Học có giám sát

- Linear Regression
  - Decision Tree Regression
  - Random Forest Regression
  - Support Vector Regression
2. Phương pháp phân loại: Phân loại là thuật toán tìm ra các hàm số có thể chia dữ liệu thành nhiều nhóm dựa trên nhiều thông số khác nhau. Khi sử dụng thuật toán phân loại, máy sẽ học trên tập dữ liệu và phân loại dữ liệu vào nhiều nhóm dựa trên những gì đã học.
- Thuật toán phân loại chuyển các dữ liệu đầu vào thành dữ liệu đầu ra rời rạc (các



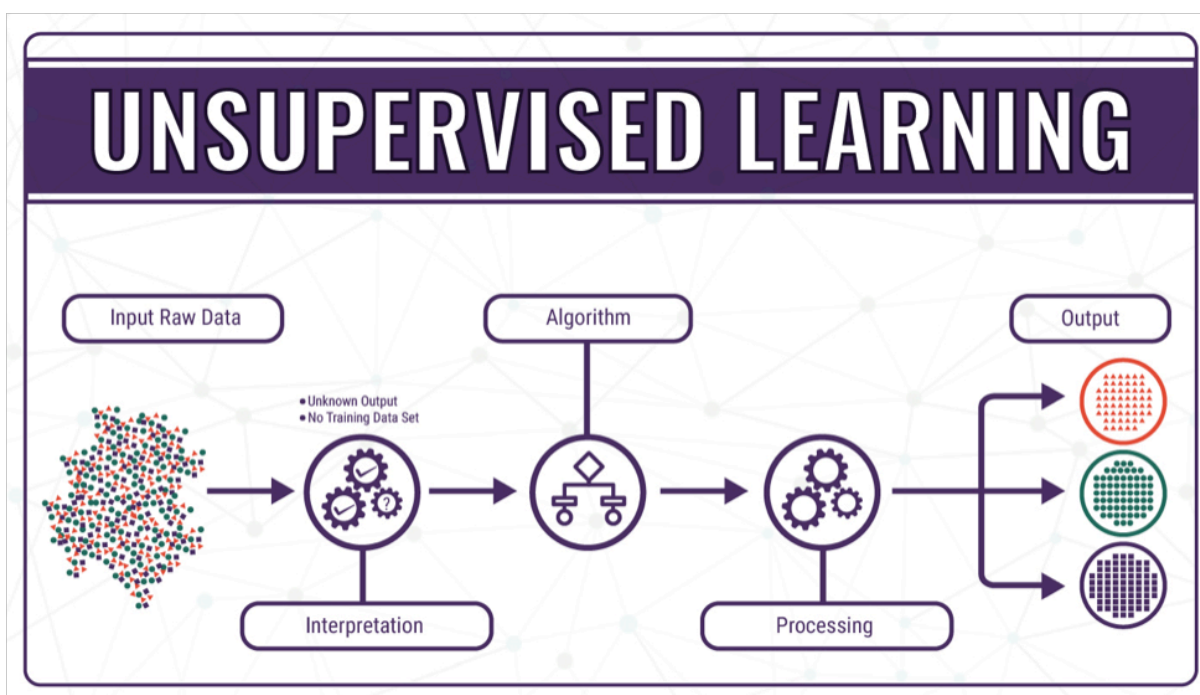
Hình 2.2: Phân loại và hồi quy

giá trị nhị phân như 0 và 1, *true* và *false*, ...). Thuật toán phân loại dự đoán khả năng xảy ra của một sự kiện bằng cách đưa dữ liệu vào hàm logit.[13]

- Logistic Regression
- K-Nearest Neighbors(KNN)
- Naïve Bayes
- Decision Tree Classification
- Random Forest Classification

### 2.1.1.4 Học không giám sát

Các thuật toán học không giám sát khám phá các mối quan hệ trong dữ liệu không được đánh nhãn. Trong trường hợp này, mô hình được cung cấp dữ liệu nhưng không biết được dữ liệu đầu ra mong muốn, mô hình phải dự đoán dựa trên các bằng chứng gián tiếp mà không có chỉ dẫn nào. Mô hình không được huấn luyện với các “giá trị đúng” và phải tự tìm ra các khuôn mẫu.[16]



**Hình 2.3:** Học không giám sát

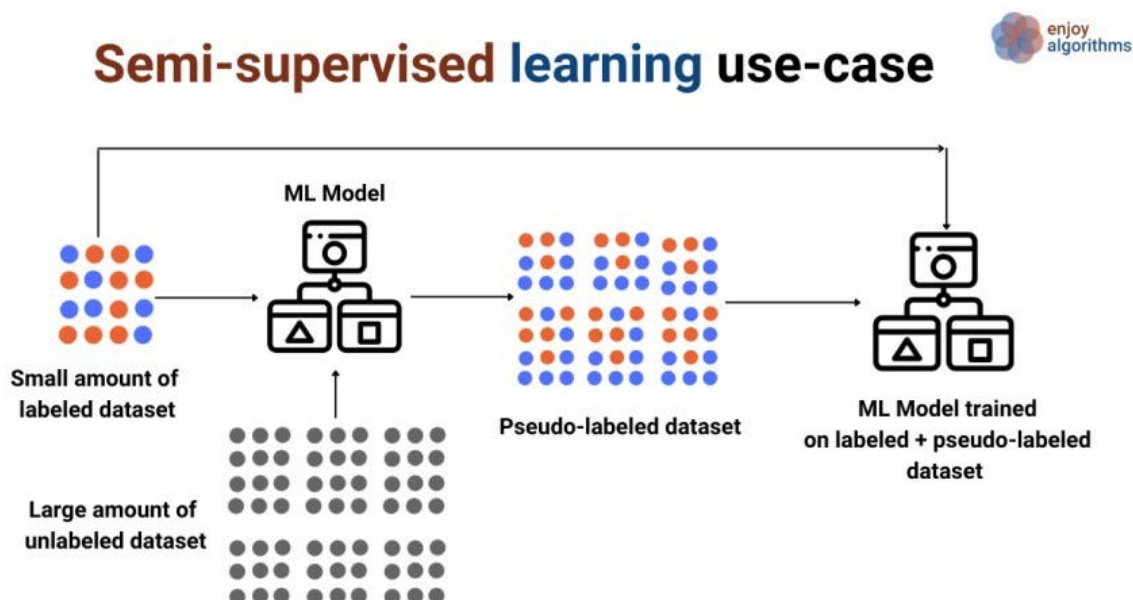
Một trong những loại học không giám sát phổ biến nhất chính là gom cụm (*clustering*), thực hiện gom nhóm các dữ liệu giống nhau. Phương pháp này thường được dùng trong phân tích khám phá và có thể tìm ra các khuôn mẫu hay xu hướng bị ẩn giấu.

Một số thuật toán học không giám sát:

- K-Means
- K-Medoids
- Fuzzy C-Means
- Gaussian Mixture

### 2.1.1.5 Học bán giám sát

Trong học bán giám sát, dữ liệu huấn luyện sẽ được chia thành 2 phần: một tập dữ liệu nhỏ sẽ chứa các dữ liệu được đánh nhãn và tập dữ liệu lớn hơn chứa các dữ liệu không đánh nhãn.



Hình 2.4: Học bán giám sát

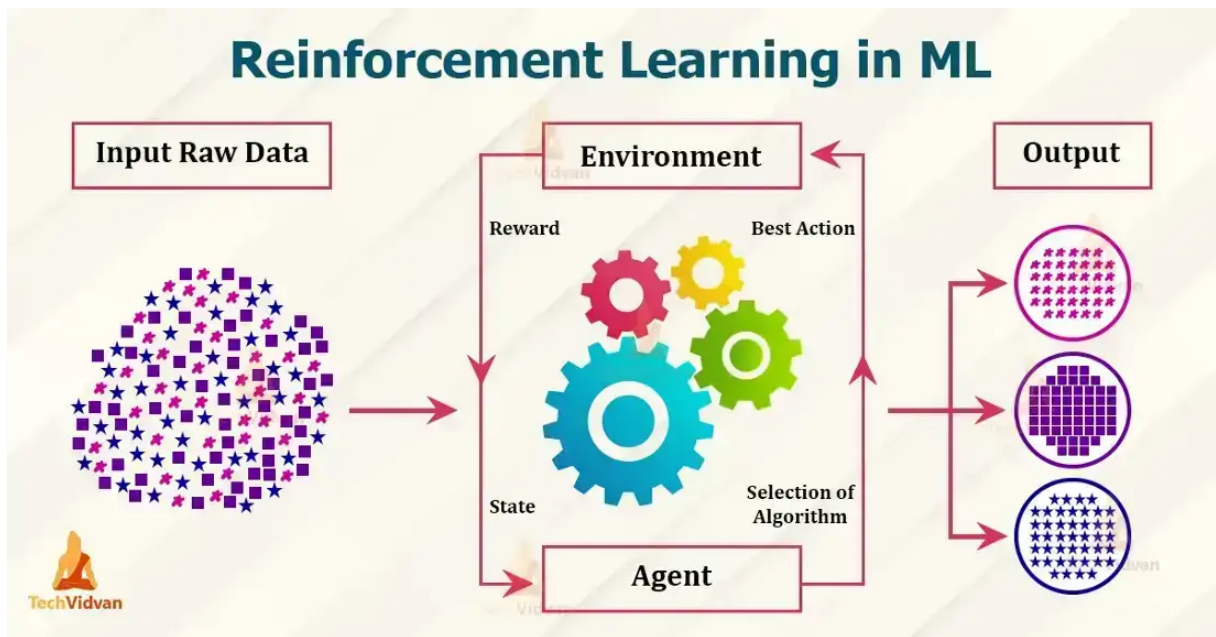
Trong trường hợp này, mô hình sẽ dùng dữ liệu có đánh nhãn để tạo ra các suy luận về dữ liệu chưa được đánh nhãn, cung cấp các kết quả chính xác hơn các mô hình học có giám sát thông thường.

Hướng tiếp cận này đang dần trở nên phổ biến, nhất là với những công việc sử dụng các tập dữ liệu lớn. Học bán giám sát không yêu cầu nhiều dữ liệu được đánh nhãn, dễ dàng cài đặt, và hoạt động với chi phí hiệu quả hơn các phương pháp học có giám sát, rất lí tưởng cho những công việc phải xử lý lượng lớn dữ liệu.[16]

### 2.1.1.6 Học tăng cường

Học tăng cường liên quan tới việc chương trình nên hoạt động như thế nào để có được kết quả tốt nhất. Nói ngắn gọn, các mô hình học tăng cường sẽ tìm cách tốt nhất có thể để tối ưu kết quả trong một số tình huống nhất định. Quá trình này là một quá trình thử đi thử lại liên tục. Và do không có dữ liệu huấn luyện, máy phải học từ chính những lỗi sai của chúng và đưa ra lựa chọn khác để dẫn tới kết quả tối ưu.

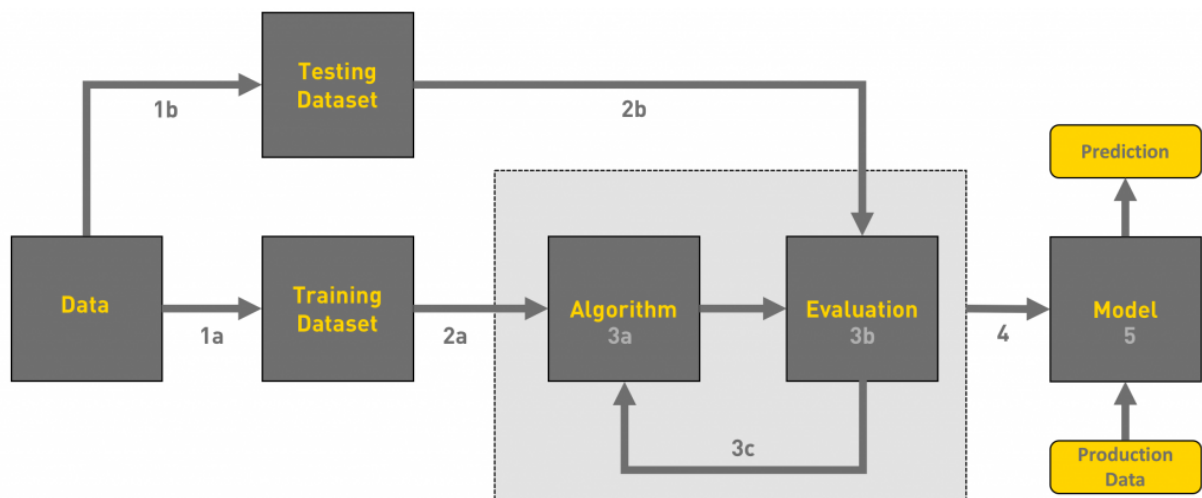
Phương pháp này thường được dùng trong các ngành robot và trò chơi điện tử. Các trò chơi điện tử thể hiện rõ ràng mối quan hệ giữa hành động và kết quả, và có thể đánh giá thành công thông qua điểm. Vì vậy, chúng là một cách thức thích hợp để cải thiện thuật toán học tăng cường.



Hình 2.5: Học tăng cường

#### 2.1.1.7 Cách thức hoạt động

Hệ thống học máy xây dựng mô hình dự đoán bằng cách học các dữ liệu có sẵn từ trước và dự đoán đầu ra cho dữ liệu mới mỗi khi nhận được.



Hình 2.6: Cách hoạt động của mô hình học máy

Quá trình học máy sẽ gồm 3 giai đoạn[2]:

##### 1. Giai đoạn 1:

- Trước khi có thể huấn luyện một mô hình học máy, chúng ta cần phải có dữ liệu. Ở giai đoạn này, chúng ta trước hết phải thu thập dữ liệu và thực hiện tiền xử lý, nhằm đảm bảo dữ liệu không có sai sót khi đưa vào huấn luyện.
- Khi đã có dữ liệu, chúng ta sẽ chia dữ liệu thành nhiều phần, có thể là 3 phần

(training, valid, test) hoặc 2 phần (training, test) để có thể sử dụng với mục đích tương ứng.

### 2. Giai đoạn 2:

- Sau khi đã có dữ liệu, việc tiếp theo cần làm đó chính là lựa chọn thuật toán và mô hình phù hợp. Việc lựa chọn mô hình có thể ảnh hưởng rất lớn đến kết quả cuối cùng.
- Sau khi đã có mô hình, chúng ta truyền dữ liệu đã chuẩn bị để mô hình có thể học và tự đánh giá.

### 3. Giai đoạn 3:

- Sau khi mô hình hoàn thiện, chúng ta tiến hành kiểm tra độ chính xác của mô hình sử dụng tập dữ liệu test đã chuẩn bị từ trước.
- Từ kết quả trên có thể đánh giá lại độ hiệu quả của mô hình và sử dụng mô hình khác nếu cần thiết.

## 2.1.2 Khái quát về học sâu

### 2.1.2.1 Giới thiệu

Học sâu (*deep learning*) có thể được xem là một nhánh của học máy. Nếu như ở học máy, các hệ thống máy sẽ học dựa trên tập dữ liệu và cải thiện nó dựa trên các thuật toán thì ở học sâu, quá trình học sẽ dựa trên các hệ thống mạng thần kinh (*neural network*) - dựa trên bộ não người - để có thể bắt chước khả năng tư duy của bộ não con người.[17]

### 2.1.2.2 Cơ sở hình thành

Bộ não con người và máy tính ngay từ bản chất đã rất khác nhau, máy tính có thể dễ dàng tính toán những con số mà con người khó tính được, còn con người có thể xử lý những công việc mang tính tư duy mà máy tính không thể thực hiện. Các mạng tế bào thần kinh của con người đã tiến hóa qua hàng triệu năm và hoạt động theo cách mà con người cũng không thể hiểu hết được. Nhưng với những kiến thức đã có được về mạng thần kinh sinh học, chúng ta đã tạo ra mạng thần kinh nhân tạo dựa trên những hiểu biết đó. Tuy nhiên mạng thần kinh nhân tạo ban đầu lại không được đón nhận do sự thiếu hiệu quả so với các thuật toán truyền thống do sự hạn chế về máy móc cũng như các thuật toán truyền thống cho ra kết quả tốt với những tập dữ liệu nhỏ hơn.

Nhưng trong những năm gần đây, với sự phát triển của dữ liệu lớn (*big data*) cùng với sức mạnh tính toán của máy tính đã được cải thiện, các thuật toán truyền thống không thể nào tận dụng hết được chúng, từ đó mạng thần kinh nhân tạo dần được ứng dụng nhiều hơn, cũng như sự phổ biến của học sâu.[9]

### 2.1.2.3 Ưu nhược điểm

Học sâu có một số ưu điểm vượt trội:

- Kiến trúc mạng nơ-ron linh hoạt, có thể dễ dàng thay đổi để phù hợp với nhiều vấn đề khác nhau.



- Có khả năng giải quyết nhiều bài toán phức tạp với độ chính xác rất cao.
- Tính tự động hoá cao, có khả năng tự điều chỉnh và tự tối ưu.
- Có khả năng thực hiện tính toán song song, hiệu năng tốt, xử lý được lượng dữ liệu lớn.

Học sâu vẫn còn một số hạn chế gắn liền với nó ví dụ như:

- Cần có khối lượng dữ liệu rất lớn để tận dụng tối đa khả năng của học sâu.
- Chi phí tính toán cao vì phải xử lý nhiều mô hình phức tạp.
- Chưa có nền tảng lý thuyết mạnh mẽ để lựa chọn các công cụ tối ưu cho học sâu.

Tuy học sâu có hiệu năng và độ chính xác vượt trội, tuy nhiên để đạt được nó cần một lượng dữ liệu lớn và các mô hình phức tạp. Vì vậy, việc lựa chọn sử dụng học sâu hay không phụ thuộc nhiều vào mục tiêu của dự án, lượng dữ liệu và tài nguyên, ...[17]

## 2.2 MẠNG NEURON NHÂN TẠO

### 2.2.1 Mạng neuron nhân tạo

#### 2.2.1.1 Giới thiệu

Bộ não con người và máy tính từ bản chất đã phù hợp với các loại công việc khác nhau. Ví dụ, việc tính căn bậc ba của một số lớn rất dễ dàng đối với máy tính, nhưng với con người thì vô cùng khó khăn. Trong khi đó, việc nhận diện các đối tượng trong một hình ảnh là một việc đơn giản đối với con người, nhưng lại rất khó khăn để con người có thể thực hiện một thuật toán tự động. Chỉ trong những năm gần đây, học sâu mới đã cho thấy độ chính xác trong một số công việc vượt qua cả của con người. Trên thực tế, các kết quả gần đây từ các thuật toán học sâu vượt qua con người trong việc nhận diện hình ảnh (chỉ trong một số trường hợp cụ thể), điều mà cách đây 20 năm được đánh giá là bất khả thi bởi các chuyên gia thị giác máy tính.[11]

Nhiều mạng thần kinh sinh học (*biological neural networks*) đã cho thấy hiệu suất tính toán, nhận dạng một cách vượt trội dựa vào các kết nối thần kinh. Điều này cho thấy rằng mạng thần kinh sinh học đạt được hiệu suất ấy nhờ vào những kết nối đủ sâu, rộng và phức tạp. Thật không may là các mạng thần kinh sinh học được kết nối theo cách mà chúng ta chưa hiểu rõ. Trong những trường hợp ít ỏi mà chúng ta hiểu về cấu trúc mạng thần kinh sinh học, chúng ta đã áp dụng, tạo ra và phát triển mạng thần kinh nhân tạo dựa trên những hiểu biết đó. Một ví dụ điển hình về loại kiến trúc này là việc sử dụng mạng thần kinh tích chập (*convolutional neural network*) cho việc nhận diện hình ảnh. Kiến trúc này đã được truyền cảm hứng từ các thí nghiệm của Hubel và Wiesel vào năm 1959 về tổ chức của các tế bào thần kinh trong vỏ não thị giác của mèo. Tiền đề cho mạng thần kinh tích chập là mạng thần kinh neocognitron, một mạng thần kinh được phát triển dựa trên thí nghiệm này.[11]

Cấu trúc kết nối tế bào thần kinh của con người đã tiến hóa qua hàng triệu năm để tối ưu hóa hiệu suất tính toán cho việc tồn tại. Và sự tồn tại này thì liên kết chặt chẽ với cảm

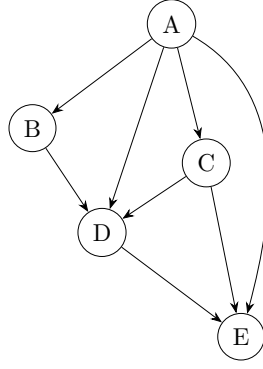
giác và trực giác của con người theo một cách mà máy móc không thể giả lập lại được. Sinh lý học thần kinh là một lĩnh vực còn rất non trẻ và chúng ta chỉ mới hiểu biết rất ít về cách hoạt động thực sự của não bộ. Và khi lĩnh vực này phát triển mạnh sẽ mang đến những thành tựu khác được áp dụng khoa học máy tính, và ta có thể mong chờ rằng những thành tựu ấy sẽ thành công rực rỡ như sự thành công của mạng thần kinh tích chập.[11]

Một phần lớn thành tựu gần đây của các mạng thần kinh nhân tạo là do sự ra đời của dữ liệu lớn (*big data*) cùng sức mạnh tính toán của máy tính hiện đại đã vượt qua giới hạn của các thuật toán học máy truyền thống, các thuật toán mà không thể tận dụng đầy đủ sức mạnh tính toán hiện nay. Đôi khi, các thuật toán học máy truyền thống lại tốt hơn khi mà các tập dữ liệu nhỏ hơn, các thuật toán này lại dễ dàng hơn trong giải thích mô hình, và xu hướng tạo ra những đặc trưng có thể giải thích bằng sự hiểu biết trong lĩnh vực đó. Với dữ liệu không đủ lớn, ta sử dụng nhiều mô hình truyền thống và chọn ra mô hình tốt nhất thường sẽ mang lại kết quả tốt hơn so với việc trực tiếp sử dụng mạng thần kinh nhân tạo. Điều này là một trong những lý do tại sao tiềm năng của mạng thần kinh nhân tạo không được đánh giá cao khi nó mới được phát triển.

Thời đại dữ liệu lớn đã được kích hoạt bởi sự tiến bộ trong công nghệ thu thập dữ liệu. Gần như mọi việc chúng ta làm ngày nay, bao gồm việc mua một món hàng, sử dụng điện thoại, hoặc nhấp chuột vào một trang web, đều được thu thập và lưu trữ ở một nơi nào đó. Hơn nữa, sự phát triển của bộ xử lý đồ họa (*Graphics Processor Units – GPU*) đã cho phép xử lý hàng trăm triệu các phép tính mỗi giây và nhân các ma trận với hàng triệu phần tử, điều mà bộ xử lý trung tâm (*CPU*) không thể đạt được. Những tiến bộ này cho phép chúng ta sử dụng các thuật toán học sâu (*deep learning*) được nghiên cứu từ hai thập kỷ trước chỉ với sự tinh chỉnh không đáng kể. Trong quá khứ, nếu cần một tháng để kiểm thử một thuật toán, thì tối đa ta chỉ có thể kiểm thử mười hai biến thể trong một năm. Điều này từng hạn chế việc kiểm thử chuyên sâu để điều chỉnh nghiệm trong các thuật toán mạng thần kinh nhân tạo. Đặc biệt, các tiến bộ nhanh chóng liên quan đến ba trụ cột trong ngành là: dữ liệu, tính toán, và thử nghiệm đã mở ra một tương lai rộng mở của học sâu. Vào cuối thế kỷ này, người ta dự kiến rằng máy tính sẽ có khả năng huấn luyện mạng thần kinh nhân tạo với số lượng tế bào thần kinh tương đương với não bộ con người.[11]

Mạng thần kinh có dạng đồ thị không có chu trình (*directed acyclic graph*) (Hình 2.7) bao gồm các cạnh và các nút. Các cạnh của mạng là các tham số. Các nút chứa một giá trị cố định (nếu là nút đầu vào) hoặc giá trị được tính toán từ các cặp nút - cạnh kết nối đến nó. Nên có thể xem mạng thần kinh chỉ là sự tính toán các hàm từ các nút đầu vào đến các nút đầu ra.

Trong mạng một lớp, một tập hợp các nút đầu vào được kết nối trực tiếp với một hoặc

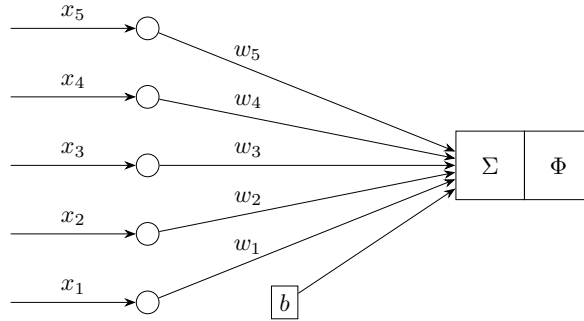


**Hình 2.7:** Đồ thị không có chu trình

nhiều nút đầu ra. Trong mạng nhiều lớp, các nơron được sắp xếp thành nhiều lớp liên tiếp nhau, trong đó các lớp đầu vào và đầu ra được tách ra bởi một nhóm các lớp ẩn các nút.

#### 2.2.1.2 Mạng thần kinh nhân tạo một lớp - (Single-layer Neural Network - Perceptron)

Mạng thần kinh nhân tạo đơn giản nhất có một lớp được gọi là perceptron (Hình 2.8). Mạng này bao gồm một lớp đầu vào và một nút đầu ra. Ta ký hiệu đầu vào bằng vector  $\mathbf{x}$  và đầu ra là một số  $y$ . Lớp đầu vào chứa  $d$  nút nên có thể viết đầu vào là một vector  $\mathbf{x} = [x_1 \dots x_d]$ .



**Hình 2.8:** Cấu tạo mạng thần kinh nhân tạo đơn giản

Để đơn giản ta quy định đầu ra  $y$  là một biến nhị phân (*binary*)  $y \in \{-1, +1\}$ . Lớp đầu vào là các nút được liên kết với các cạnh có trọng số  $w_1, \dots, w_d$  ( $\mathbf{w} = [w_1, \dots, w_d]$ ), tất cả các cạnh này được kết nối với một nút đầu ra duy nhất. Một hàm tuyến tính được hình thành từ đầu vào  $\mathbf{x}$  và trọng số  $\mathbf{w}$  là  $\mathbf{w}^T \mathbf{x} = \sum_{i=1}^d w_i x_i$  sẽ tính giá trị đầu ra rồi đưa giá trị này vào hàm  $\text{sign}(\cdot)$ . Vậy nên giá trị dự đoán  $\hat{y}$  là

$$\hat{y} = f(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x}) = \text{sign}\left(\sum_{i=1}^d w_i x_i\right) \quad (2.1)$$

Hàm  $\text{sign}$  chuyển đổi một giá trị thực thành  $+1$  hoặc  $-1$ , nghĩa là bài toán perceptron trong ví dụ trên giống với bài toán phân loại nhị phân (*binary classification*). Ta chọn

một hàm mất mát *loss function* để đánh giá độ lệch giữa giá trị quan sát  $y$  với giá trị dự đoán  $\hat{y}$ . Ban đầu hàm  $f(\cdot)$  với trọng số  $\mathbf{w}$  được tạo ngẫu nhiên. Với mỗi đầu vào  $\mathbf{x}$ , ta thay đổi trọng số  $\mathbf{w}$  để giá trị đầu ra giống dữ liệu huấn luyện, làm hàm  $f(\cdot)$  có độ chính xác cao hơn. Quá trình này lặp đi lặp lại để nhận được trọng số  $\mathbf{w}$  tốt nhất, làm cho tổng mất mát trên toàn bộ dữ liệu huấn luyện là thấp nhất được gọi là quá trình học (*learning*).

Thời điểm sơ khai khi lĩnh vực học máy chưa phát triển, hàm mất mát của thuật toán chưa được tối ưu hoá mà chỉ được xây dựng trên những kinh nghiệm một cách tự nhiên (*heuristic*). Với đầu vào là  $\mathbf{x}$ , giá trị dự đoán sẽ là  $\hat{y} = f(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x})$ . Khi giá trị dự đoán  $\hat{y}$  khác giá trị quan sát  $\hat{y} \neq y \in \{-1, +1\}$  thì trọng số  $\mathbf{w}$  sẽ được cập nhật dựa trên  $(y - \hat{y})$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(y - \hat{y})\mathbf{x} \quad (2.2)$$

Với số  $\alpha$  được gọi là tỉ lệ học (*learning rate*). Thuật toán perceptron bây giờ là quá trình duyệt qua toàn bộ dữ liệu một cách ngẫu nhiên, lặp đi lặp lại với hữu hạn lần (mỗi chu kỳ lặp được gọi là 1 *epoch*). Mỗi lần duyệt qua một điểm sẽ cập nhật lại trọng số  $\mathbf{w}$ , sao cho khi học xong toàn bộ tập dữ liệu sẽ cho ra một mô hình (*model*) dự báo đúng trên toàn bộ tập dữ liệu, khi đó ta gọi thuật toán đạt được sự hội tụ (*convergence*). Giá trị của  $(y - \hat{y})$  luôn là  $2y$  khi  $\hat{y} \neq y$ , nên ta có thể viết lại

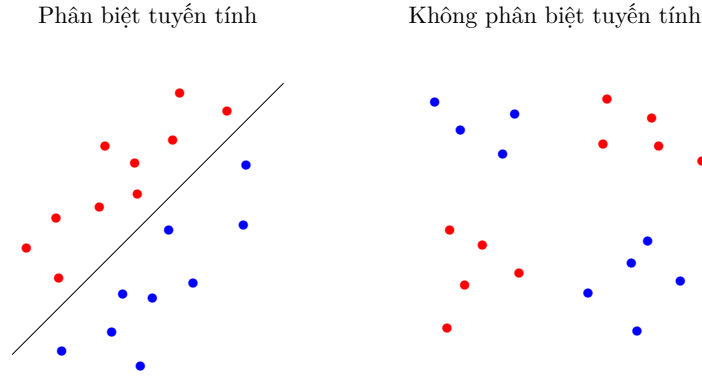
$$\mathbf{w} \leftarrow \mathbf{w} + 2\alpha y \mathbf{x} \sim \mathbf{w} + \alpha y \mathbf{x} \quad (2.3)$$

Bài toán perceptron cũng được coi là một mô hình tuyến tính (*linear model*), bởi vì phương trình  $\mathbf{w}^T \mathbf{x} = 0$  định nghĩa một siêu phẳng (*hyperplane*), với  $\mathbf{w} = [w_1, \dots, w_d]$  là một vector  $d$  chiều trực giao (*perpendicular*) với siêu phẳng. Siêu phẳng này chia không gian thành hai nửa, các điểm dữ liệu hoặc thuộc về nửa dương (*positive*) khi  $\mathbf{w}^T \mathbf{x} > 0$  hoặc thuộc về nửa âm (*negative*) khi  $\mathbf{w}^T \mathbf{x} < 0$ . Mô hình này chạy tốt khi tập dữ liệu là phân biệt tuyến tính (*linearly separable*) (Hình 2.9), nghĩa là có tồn tại một siêu phẳng chia các điểm dữ liệu thành hai phần, mỗi phần thuộc về một class khác nhau. Thuật toán này đã được Rosenblatt chứng minh là hội tụ khi và chỉ khi tập dữ liệu là phân biệt tuyến tính.[11]

Đối với tập dữ liệu có phân phối nhị phân (*binary class distribution*) bị mất cân bằng (*imbalanced*), như trong bài toán phân loại nhị phân là trường hợp trung bình của các điểm có nhãn  $\{-1, +1\}$  khác 0

$$\underbrace{\sum_i y_i}_{\neq 0} \neq \underbrace{\sum_i \hat{y}_i}_{=0} = \sum_i \text{sign}(\mathbf{w}^T \mathbf{x}_i) \quad (2.4)$$

Khi tập dữ liệu mất cân bằng càng nhiều, thuật toán perceptron càng mất chính xác. Để khắc phục điều này, ta cộng thêm một lượng nhỏ  $b$  (*bias*) vô giá trị dự đoán để làm cho



**Hình 2.9:** Phân biệt tuyến tính

giá trị này bị mất cân bằng

$$\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x} + b) = \text{sign}\left(\sum_{i=1}^d w_i x_i + b\right) \quad (2.5)$$

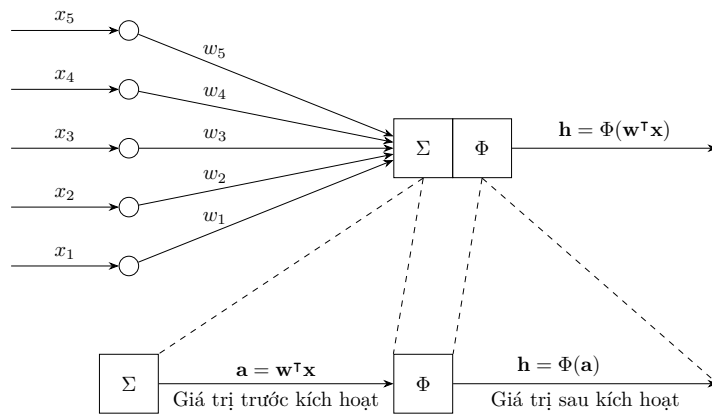
Và ta sẽ điều chỉnh  $b$  để kết quả dự báo mất cân bằng giống với cách mà giá trị quan sát mất cân bằng.

### 2.2.1.3 Hàm kích hoạt - (Activation Function)

Hàm kích hoạt được định nghĩa là một giá trị dự đoán  $\Phi$  có dạng

$$\hat{y} = \Phi(\mathbf{w}^T \mathbf{x}) \quad (2.6)$$

Ta chia một nút bất kì (khác nút đầu vào) thành 2 giai đoạn, giá trị của nút trước khi áp dụng hàm kích hoạt gọi là *pre-activation value*, ngược lại là *post-activation value* (hình 2.10).



**Hình 2.10:** Giá trị trước khi kích hoạt và giá trị sau khi kích hoạt

Hàm kích hoạt thường thấy nhất là hàm tuyến tính. Hàm tuyến tính thường được sử dụng ở các nút đầu ra vì thường chúng ta mong muốn đầu ra là một số thực nào đó.

$$\Phi(v) = v \quad (2.7)$$

## CHƯƠNG 2. NỘI DUNG

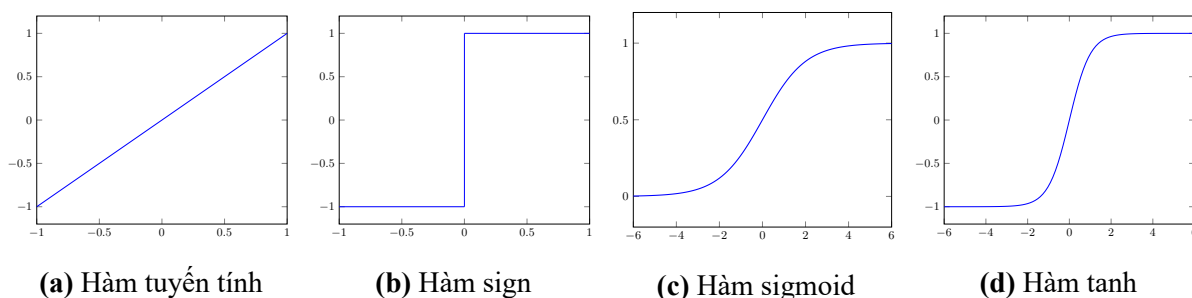
Hàm kích hoạt phi tuyến được sử dụng trong những ngày đầu phát triển của mạng thần kinh nhân tạo là hàm sign, hàm sigmoid, hàm tanh (hình 2.11).

$$\Phi(v) = \text{sign}(v) \quad [\text{hàm sign}] \quad (2.8)$$

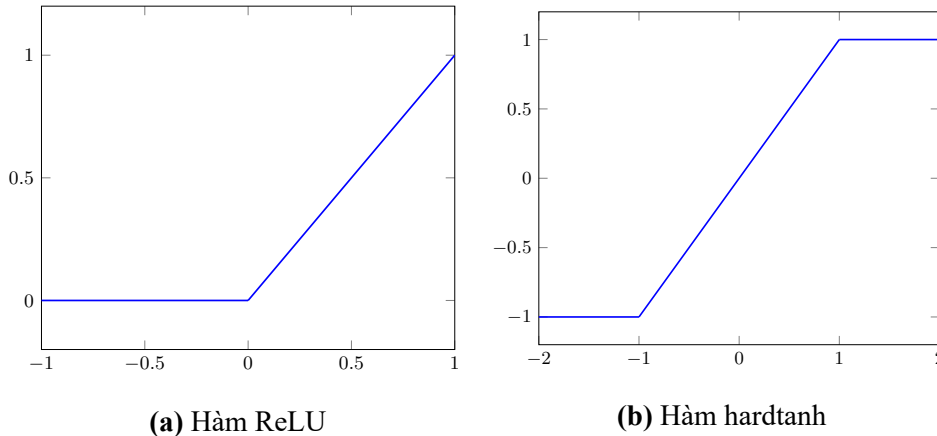
$$\Phi(v) = \frac{1}{1 + e^{-v}} \quad [\text{hàm sigmoid}] \quad (2.9)$$

$$\Phi(v) = \frac{e^{2v} - 1}{e^{2v} + 1} \quad [\text{hàm tanh}] \quad (2.10)$$

Hàm sigmoid cho ra giá trị từ  $(0, 1)$  sẽ phù hợp khi dùng cho đầu ra có dạng xác suất.



**Hình 2.11:** Đồ thị các hàm kích hoạt thông dụng



**Hình 2.12:** Đồ thị các hàm kích hoạt tuyến tính từng phần

Đồ thị hàm tanh khá giống hàm sigmoid nhưng khác ở điểm nó có đầu ra từ  $[-1, 1]$ , và thường được ưu tiên hơn hàm sigmoid vì nó cho cả giá trị âm. Điểm yếu của các hàm phi tuyến là chúng quá dễ đạt đến giá trị bão hoà (giá trị sẽ không thay đổi nhiều trong khoảng lân cận) khi mà giá trị đầu vào lớn đến một mức nào đó, ví dụ hàm sigmoid khi  $x > 6$  thì giá trị  $y_{x>6}$  sẽ không khác nhau nhiều. Nhưng ngược lại hàm phi tuyến lại giúp ta chuẩn hoá giá trị đầu ra về một khoảng xác định (ví dụ đối với hàm sigmoid là  $(0, 1)$ ).

Trong những năm gần đây, hàm kích hoạt tuyến tính từng phần (*piecewise linear activation functions*) được ưu tiên sử dụng hơn, vì nó không có điểm bão hoà, và các

hàm này sẽ được tính nhanh hơn trong máy tính, giúp tối ưu đáng kể hiệu suất. Ví dụ là hàm ReLU và hàm hardtanh (hình 2.12).

$$\Phi(v) = \max(v, 0) \quad [\text{hàm Rectified Linear Unit (ReLU)}] \quad (2.11)$$

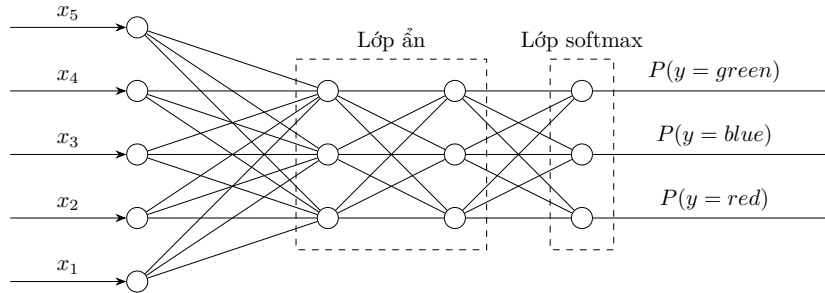
$$\Phi(v) = \max(\min(v, 1), -1) \quad [\text{hàm hardtanh}] \quad (2.12)$$

#### 2.2.1.4 Hàm kích hoạt Softmax - (Softmax Activation Function)

Hàm kích hoạt softmax là một hàm đặc biệt vì nó thường được sử dụng trong lớp đầu ra để ánh xạ  $k$  giá trị thực thành  $k$  xác suất của các sự kiện rời rạc, và tổng xác suất của các sự kiện này là 1. Ví dụ, trong bài toán phân loại sản phẩm vào  $k$  lớp khác nhau, trong đó mỗi sản phẩm cần được đánh nhãn là một trong  $k$  lớp đó.

$$\Phi(v_i) = \frac{\exp(v_i)}{\sum_{j=1}^k \exp(v_j)} \quad \forall i \in \{1, \dots, k\} \quad (2.13)$$

Hình 2.13 là ví dụ của hàm softmax với các giá trị đầu ra là  $v_1, v_2, v_3$ , và lớp softmax (*softmax layer*) biến đổi các giá trị này thành xác suất của 3 lớp khác nhau. Đối với lớp softmax, các cạnh nối với lớp này sẽ có hệ số là 1.



**Hình 2.13:** Mạng thần kinh nhân tạo với lớp softmax làm lớp đầu ra

#### 2.2.1.5 Hàm mất mát - (Lost Function)

Thông thường, hàm mất mát được định nghĩa phụ thuộc vào đầu ra của mô hình và dữ liệu hiện có. Các hàm mất mát thường dùng được phân thành 2 loại [12]:

- Hàm mất mát dùng cho hồi quy (*Regression Losses*)

1. Mean bias error loss

$$L_{MBE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i) \quad (2.14)$$

2. Mean absolute error loss - L1 loss

$$L_{MAE} = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \quad (2.15)$$

3. Mean squared error loss - L2 loss

$$L_{MAE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (2.16)$$

và một số hàm khác như Root mean squared error loss, Huber loss, ...

• Hàm mất mát dùng cho phân loại (*Classification Losses*)

1. Zero-One loss

$$L_{ZeroOne}(y, \hat{y}) = \begin{cases} 1 & \text{if } y \cdot \hat{y} < 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.17)$$

2. Hinge loss

$$L_{Hinge}(y, \hat{y}) = \max(0, 1 - (y \cdot \hat{y})) \quad (2.18)$$

3. Cross Entropy loss

$$L_{CrossEntropy} = - \sum_{i=1}^N y_i \log(\hat{y}_i) \quad (2.19)$$

và một số hàm khác như Ramp loss, Cosine Similarity loss, ...

2.2.1.6 Mạng thần kinh nhân tạo nhiều lớp - (*Multi-Layer Neural Networks*)

Đối với mạng thần kinh nhân tạo một lớp thì ai cũng có thể thấy được toàn bộ quá trình tính toán, bởi vì nó chỉ gồm hai lớp là lớp đầu vào và lớp đầu ra. Nhưng đối với mạng thần kinh nhiều lớp có sự tồn tại của các lớp ẩn, thì quá trình tính toán giữa các lớp ẩn này là hoàn toàn không nhìn thấy được.

Kiến trúc cụ thể của mạng thần kinh nhiều lớp được gọi là mạng thần kinh truyền thẳng (*feed-forward networks*). Trong mạng thần kinh truyền thẳng, các lớp liên tiếp truyền dữ liệu cho nhau theo hướng từ lớp đầu vào đến lớp đầu ra, tất cả các nút trong một lớp được kết nối với các nút của lớp tiếp theo.

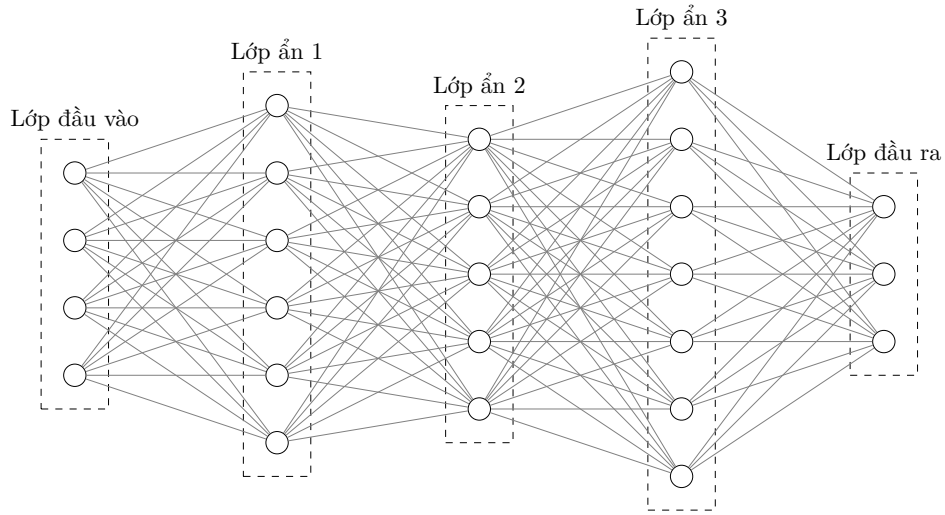
Ta có thể biểu diễn sự kết nối giữa các lớp này bằng phép nhân ma trận. Nếu mạng thần kinh có  $k$  lớp ẩn, trong mỗi lớp có  $p_1 \dots p_k$  nôt, thì vector kết quả  $\mathbf{h}_1 \dots \mathbf{h}_k$  ở mỗi lớp ẩn này sẽ có số chiều (*dimension*) lần lượt là  $p_1 \dots p_k$ . Xét lớp đầu vào và lớp ẩn đầu tiên, nếu  $d$  là số nôt của lớp đầu vào,  $W_1$  là ma trận chứa các trọng số  $w_i$  từ lớp đầu vào đến lớp ẩn đầu tiên, thì ma trận này có kích thước là  $p_1 \times d$ . Tương tự nếu ta xét kết nối giữa lớp ẩn  $r$  và lớp ẩn  $r + 1$ , thì ma trận trọng số  $W_{r+1}$  sẽ có kích thước là  $p_{r+1} \times p_r$ . Nếu lớp đầu ra của mạng có  $o$  nôt thì ma trận trọng số  $W_{k+1}$  sẽ có kích thước  $o \times p_k$ . Nếu gọi  $\Phi$  là hàm kích hoạt thì

$$\mathbf{h}_1 = \Phi(W_1 \mathbf{x}) \quad [\text{Từ lớp đầu vào đến lớp ẩn đầu tiên}] \quad (2.20)$$

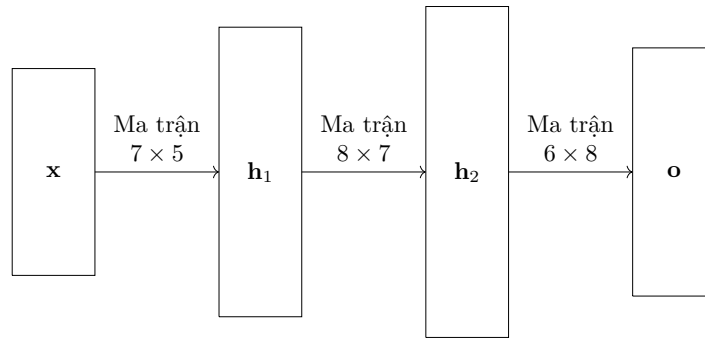
$$\mathbf{h}_{p+1} = \Phi(W_{p+1} \mathbf{h}_p) \quad [\text{Giữa các lớp ẩn}] \quad (2.21)$$

$$\mathbf{o} = \Phi(W_{k+1} \mathbf{h}_k) \quad [\text{Từ lớp ẩn cuối cùng đến lớp đầu ra}] \quad (2.22)$$





**Hình 2.14:** Mạng thần kinh nhiều lớp



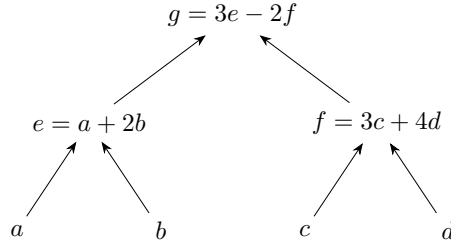
**Hình 2.15:** Mạng thần kinh nhiều lớp với cách biểu diễn dưới dạng ma trận

Nếu ta sử dụng hàm kích hoạt tuyến tính ( $\Phi(x) = x$ ) cho toàn bộ nút trong mạng, thì đầu ra **o** có thể được viết lại thành

$$\begin{aligned}
 \mathbf{o} &= W_{k+1} \mathbf{h}_k \\
 &= W_{k+1} W_k \mathbf{h}_{k-1} \\
 &\dots \\
 &= \underbrace{W_{k+1} W_k \dots W_1}_{W_{xo}} \mathbf{x}
 \end{aligned} \tag{2.23}$$

Nghĩa là dù mô hình mạng của ta có nhiều lớp, nhưng mô hình cuối cùng cũng chỉ tương đương với mô hình mạng một lớp tuyến tính (*single-layer linear network*), điều này làm mất đi điểm mạnh của mô hình mạng nhiều lớp.

Ngoài cách biểu diễn bằng phép nhân ma trận, ta có thể biểu mạng thần kinh nhân tạo bằng đồ thị tính toán (*computational graph*) (Hình 2.16).



**Hình 2.16:** Đồ thị tính toán

Nếu ta định nghĩa  $g(\cdot)$  là hàm tính giá trị của một nút ở lớp  $m$ ,  $f(\cdot)$  là hàm tính giá trị nút ở lớp  $m + 1$ , thì giá trị của nút ở lớp  $m$  đó sẽ có dạng  $f(g_1, \dots, g_k)$ , với tham số của các hàm chính là trọng số của các cạnh. Nghĩa là ta có thể xây dựng một hàm số tổng quát kết nối từ lớp đầu vào, thông qua tất cả các lớp ẩn, đến lớp đầu ra. Quá trình học máy bây giờ được rút gọn thành việc lựa chọn các tham số cho các hàm này sao cho giá trị thực sự và giá trị dự đoán trùng nhau. Nhưng để tìm được một hàm tổng quát (*closed-form*) ngắn gọn để biểu diễn cả mạng thần kinh nhân tạo là điều gần như không thể, bởi vì đây là các hàm lồng vào nhau nên khi càng có nhiều lớp ẩn thì hàm tổng quát này càng phức tạp. Chính điều này dẫn đến việc ta không thể áp dụng thuật toán xuống dốc (*gradient descent*) để tìm tham số tối ưu. Thuật toán lan truyền ngược (*backpropagation algorithm*) được sinh ra để giải quyết vấn đề này.[11]

#### 2.2.1.7 Thuật toán lan truyền ngược - (*Backpropagation Algorithm*)

Như đã nói bên trên, khi biểu diễn mạng thần kinh nhân tạo dưới dạng đồ thị tính toán, để tính giá trị của một nút ta cần phải tính giá trị của tất cả các nút ở lớp trước đó. Ví dụ mạng thần kinh nhân tạo với mỗi lớp chỉ có duy nhất một nút, hàm để tính giá trị của nút ở lớp  $m$  là  $g(x)$ , ở lớp  $m + 1$  là  $f(x)$ , nếu ta chọn  $f(x) = g(x)$  là hàm sigmoid, thì giá trị của nút ở lớp  $m + 1$  là

$$f(g(x)) = \frac{1}{1 + \exp \left[ -\frac{1}{1 + \exp(-x)} \right]} \quad (2.24)$$

Nếu ta tiếp tục tăng số lớp lên, thì giá trị của nút ở lớp  $m + n$  sẽ có dạng

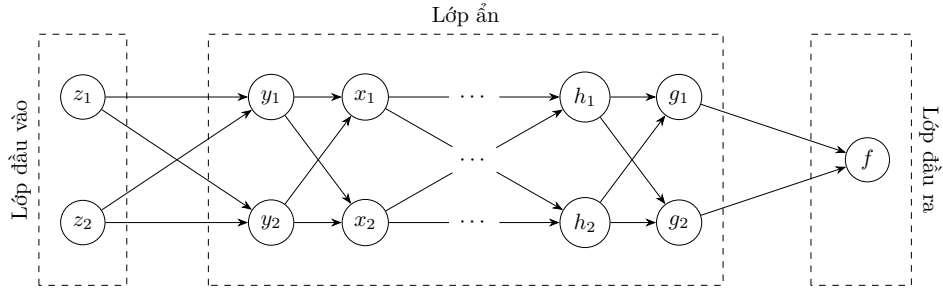
$$f(\dots z(x)) = \frac{1}{1 + \exp \left[ -\frac{1}{1 + \exp \left[ -\dots - \frac{1}{1 + \exp(-x)} \right]} \right]} \quad (2.25)$$

Đối với các thuật toán học máy truyền thống, ta phải tìm hàm  $\hat{y} = f(\mathbf{x}, \mathbf{w})$ , rồi ta chọn một hàm mất mát để đánh giá sai số giữa  $\hat{y}$  và  $y$ . Hàm mất mát này sẽ có dạng

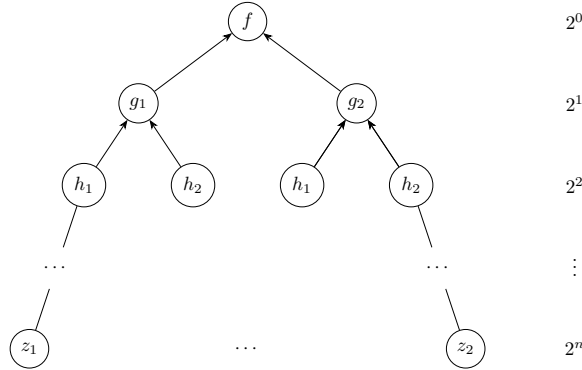
## CHƯƠNG 2. NỘI DUNG

$L = L(y, \hat{y}) = L(y, \mathbf{x}, \mathbf{w})$ . Sau đó ta phải khai triển đại số và rút gọn  $L$ , rồi áp dụng thuật toán xuống đồi để tối ưu vector  $\mathbf{w}$ . Để đơn giản hoá, thay vì tính đạo hàm của hàm mất mát  $L$ , ta sẽ tính đạo hàm của  $\hat{y}$ , rồi từ đó suy ngược lại đạo hàm của  $L(y, \hat{y})$ . Đối với mạng thần kinh nhân tạo, việc tìm ra công thức rút gọn của  $\hat{y}$  gần như là không thể vì nó thường là các hàm phi tuyến lồng vô nhau, nghĩa là công thức của hàm này sẽ rất dài và phức tạp (ví dụ trong hàm 2.25) dẫn đến việc tính đạo hàm này sẽ trở nên khó khăn, và độ khó khăn sẽ tăng lên càng cao nếu như ta tăng số lượng lớp. Đây là vấn đề đầu tiên khi dùng phương pháp giải thuật toán học máy truyền thống để tối ưu mạng thần kinh nhân tạo.

Vấn đề thứ hai gặp phải là độ phức tạp sẽ tăng khi ta tăng số lượng nút trong lớp. Xét một mạng thần kinh có  $n$  lớp, mỗi lớp có 2 nút, biểu diễn đồ thị tính toán ở hình 2.18.



**Hình 2.17:** Mạng thần kinh với  $n$  lớp, mỗi lớp có 2 nút



**Hình 2.18:** Đồ thị tính toán của mạng ở hình 2.17

Nhìn đồ thị tính toán, để tìm công thức tổng quát của hàm  $f$  dựa trên đầu vào  $z_1, z_2$ , ta phải khai triển các hàm lồng nhau. Ở tầng đầu tiên ta không có hàm lồng nhau nên số hàm lồng nhau phải khai triển là  $2^0 = 1$ , ở tầng thứ hai có một lớp lồng nhau là  $f(g_1, g_2)$  nên số hàm phải khai triển là  $2^1 = 2$ , tương tự khi ta đi qua từng lớp đến lớp cuối cùng  $n$ , thì tổng số hàm lồng nhau mà ta phải khai triển là  $2^n$ . Độ phức tạp khi tính toán sẽ tăng theo cấp số nhân khi mà số lượng lớp tăng lên.

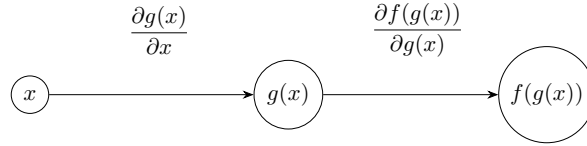
Để giải quyết vấn đề đầu tiên, nghĩa là không phải tính các hàm lồng nhau, người ta

đưa ra giải pháp là áp dụng quy tắc xích (*chain rule*) liên tục lặp đi lặp lại để tính đạo hàm ở từng lớp kế nhau (đạo hàm ở lớp  $m$  đối với các nút ở lớp  $m - 1$ ), việc này làm giảm độ phức tạp vì chỉ phải tính đạo hàm trên một lớp duy nhất, không cần phải khai triển toàn bộ hàm. Bắt đầu tính đạo hàm ở lớp cuối cùng  $n$ , sau đó ta truyền giá trị đạo hàm xuống lớp kế nó (lớp  $n - 1$ ) để tính đạo hàm ở lớp này, rồi lặp lại quá trình này cho đến đầu đầu vào. Quá trình này gọi là lan truyền ngược, và việc tính toán theo cách số học (*numerically*) cho từng lớp kế nhau sẽ dễ dàng hơn nhiều với việc tính toán kiểu đại số (*algebraically*) trên toàn mạng rồi thay số vào.

Nhắc lại quy tắc xích

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x} \quad (2.26)$$

Đối với mạng thần kinh đơn giản (hình 2.19) với một nút đầu vào, một nút ẩn và một nút



**Hình 2.19:** Áp dụng quy tắc xích trong thần kinh đơn giản

đầu ra, áp dụng quy tắc xích làm đơn giản hoá việc tính đạo hàm trên toàn mạng bằng việc tính đạo hàm trên từng cạnh rồi nhân với nhau. Đối với mạng thần kinh phức tạp hơn gồm nhiều lớp ẩn, mỗi lớp gồm nhiều nút, thì ta phải áp dụng quy tắc xích đa biến (*multivariate chain rule*)

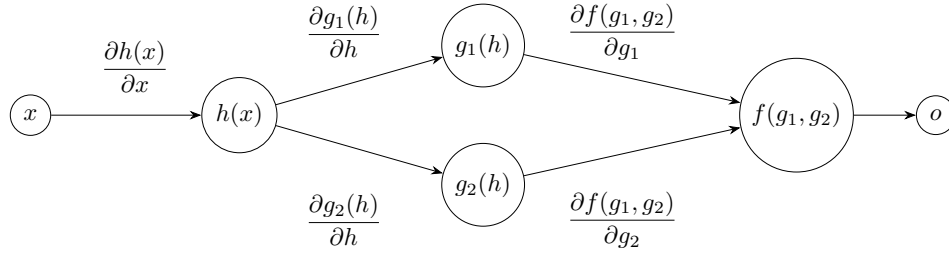
$$\frac{\partial f(g_1(x), \dots, g_k(x))}{\partial x} = \sum_{i=1}^k \frac{\partial f(g_1(x), \dots, g_k(x))}{\partial g_i(x)} \cdot \frac{\partial g_i(x)}{\partial x} \quad (2.27)$$

Áp dụng quy tắc xích đa biến vào mạng (hình 2.20), ta được

$$\frac{\partial o}{\partial x} = \frac{\partial f(g_1, g_2)}{\partial g_1} \cdot \frac{\partial g_1(h)}{\partial h} \cdot \frac{\partial h(x)}{\partial x} + \frac{\partial f(g_1, g_2)}{\partial g_2} \cdot \frac{\partial g_2(h)}{\partial h} \cdot \frac{\partial h(x)}{\partial x} \quad (2.28)$$

Dựa vào quy tắc xích đa biến, Charu C. Aggarwal đã nêu ra bổ đề về tổng hợp của các cạnh (*pathwise aggregation lemma*) [11] như sau: gọi  $y(i)$  là giá trị của nút  $i$  trong mạng, gọi  $z(i, j) = \partial y(j) / \partial y(i)$  là đạo hàm của cạnh  $(i, j)$ , gọi tập hợp các đường đi khác nhau xuất phát từ điểm  $s$  và kết thúc ở điểm  $t$  là  $\mathcal{P}$ , giá trị của đạo hàm  $\partial y(t) / \partial y(s)$  được tính bằng tích của các cạnh nối từ  $s$  đến  $t$ , rồi lấy tổng theo các đường đi khác nhau dẫn từ  $s$  đến  $t$

$$\frac{\partial y(t)}{\partial y(s)} = \sum_{P \in \mathcal{P}} \prod_{(i,j) \in P} z(i, j) \quad (2.29)$$



**Hình 2.20:** Áp dụng quy tắc xích đa biến trong mạng thần kinh phức tạp

Giải pháp này chỉ giải quyết được vấn đề đầu tiên mà không giải quyết được vấn đề thứ hai. Trong ví dụ hình 2.20, có 2 con đường dẫn từ đầu vào đến đầu ra. Hai con đường này đều đi qua điểm  $h(x)$ , nên trong đạo hàm của hàm 2.28, ta thấy có sự xuất hiện 2 lần của đạo hàm  $\partial h(x)/\partial x$ , nghĩa là nếu một cạnh  $(i, j)$  xuất hiện trong  $n$  đường đi  $P_1, \dots, P_n \in \mathcal{P}$ , thì ta phải tính đạo hàm của cạnh  $(i, j)$   $n$  lần. Để giải quyết vấn đề tính trùng nhau này, người ta dùng phương pháp quy hoạch động (*dynamic programming*) trong lý thuyết đồ thị (*graph theory*).

Gọi  $S(i, j) = \partial y(j)/\partial y(i)$ , ta viết lại bổ đề 2.29 thành

$$S(s, t) = \sum_{P \in \mathcal{P}} \prod_{(i, j) \in P} z(i, j) \quad (2.30)$$

Chọn điểm  $i$  bất kỳ trong mạng, gọi  $j \in A(i)$  là tập hợp các điểm mà  $i$  kết nối đến. Nếu tất cả giá trị của điểm  $S(j, t)$  đã được tính, thì ta có hàm cập nhật  $S(i, t)$  dựa trên các giá trị đã tính  $S(j, t)$  là

$$S(i, t) = \sum_{j \in A(i)} z(i, j) S(j, t) \quad (2.31)$$

Với giá trị khởi tạo  $S(t, t) = 1$ , mã giả của thuật toán quy hoạch động để tính  $S(s, t)$  là

---

**Giải thuật 1** Mã giả tính  $S(s, t)$  bằng quy hoạch động [11]

---

- 1: Khởi tạo  $S(t, t) = 1$
  - 2: **repeat**
  - 3:     Chọn nút  $i$  thỏa tất cả giá trị  $S(j, t)$  đã được tính, với  $j \in A(i)$
  - 4:     Cập nhật  $S(i, t) \leftarrow \sum_{j \in A(i)} z(i, j) S(j, t)$
  - 5: **until** tất cả các nút đã được tính
- 

Trong hầu hết các thuật toán học máy truyền thống, người ta chỉ cần tìm lời giải một lần duy nhất là một hàm  $f$  dưới dạng rút gọn, sau đó chỉ cần thay các điểm dữ liệu khác

nhau là ta sẽ cập nhật được bộ tham số  $\mathbf{w}$  dựa theo thuật toán xuống đồi. Đối với thuật toán lan truyền ngược, ta phải chạy lại thuật toán với mỗi điểm khác nhau. Điều này nghe như khối lượng công việc sẽ nhiều hơn so với các thuật toán truyền thống, nhưng nó đã được chứng minh là tốt hơn nhiều so với cách giải truyền thống khi mà số lượng lớp và nút ở mỗi lớp lớn.

Như đã nói ở trên, sau khi đã tính được đạo hàm của  $\partial y(j)/\partial y(i)$ , ta phải suy ra đạo hàm của  $\partial L/\partial \mathbf{w}$ . Nếu mạng thần kinh có  $p$  nút đầu ra, thì các nút đầu ra sẽ là  $t_1, t_2, \dots, t_p$ , hàm mất mát bây giờ sẽ là  $L(y(t_1), y(t_2), \dots, y(t_p))$ . Gọi  $w_{ji}$  là tham số cạnh giữa hai nút  $j$  và  $i$ . Khi này đạo hàm riêng của hàm mất mát sẽ là

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial y(i)} \cdot \frac{\partial y(i)}{\partial w_{ji}} \quad (2.32)$$

$$= \left[ \sum_{k=1}^p \frac{\partial L}{\partial y(t_k)} \cdot \frac{\partial y(t_k)}{\partial y(i)} \right] \frac{\partial y(i)}{\partial w_{ji}} \quad (2.33)$$

Giá trị của  $\partial y(i)/\partial w_{ji}$  ( $j$  là điểm nối đến  $i$ ) có thể được tính theo hình 2.8. Giá trị của  $\partial y(t_k)/\partial y(i)$  được tính bằng thuật toán quy hoạch động bên trên. Giá trị của  $\partial L/\partial y(t_k)$  được tính như các thuật toán học máy truyền thống vì  $L = L(y(t_1), y(t_2), \dots, y(t_p))$ . Nhưng thay vì dùng thuật toán 1 để tính từng phần của 2.33, ta có thể biến đổi thuật toán này để tính  $\partial L/\partial y(i)$ .

Chọn điểm  $i$  bất kỳ trong mạng, gọi  $j \in A(i)$  là tập hợp các điểm mà  $i$  kết nối đến. Đặt  $\Delta(i) = \partial L/\partial y(i)$ , dựa vào 2.33, ta có thể tính  $\Delta(i)$  dựa trên các giá trị đã tính  $\Delta(j)$

$$\Delta(i) = \sum_{j \in A(i)} z(i, j) \Delta(j) \quad (2.34)$$

Khởi tạo đạo hàm mất mát với tất cả các nút đầu ra  $\Delta(t_k)$ , ta có

---

**Giải thuật 2** Mã giả tính  $\partial L/\partial \mathbf{w}$  bằng quy hoạch động [11]

---

- 1: Khởi tạo  $\Delta(t_k) = \frac{\partial L}{\partial y(t_k)}$  với mỗi  $k \in \{1, \dots, p\}$
  - 2: **repeat**
  - 3:     Chọn nút  $i$  thỏa tất cả giá trị  $\Delta(j)$  đã được tính, với  $j \in A(i)$
  - 4:     Cập nhật  $\Delta(i) \leftarrow \sum_{j \in A(i)} z(i, j) \Delta(j)$
  - 5: **until** tất cả các nút đã được tính
  - 6: **for each** cạnh  $(j, i)$  có hệ số  $w_{ji}$  **do**
  - 7:     Tính  $\frac{\partial L}{\partial w_{ji}} \leftarrow \Delta(i) \cdot \frac{\partial y(i)}{\partial w_{ji}}$
  - 8: **end for**
-

Sau khi tính  $\partial L / \partial \mathbf{w}$ , ta có thể áp dụng thuật toán xuống đồi để tối ưu tham số  $\mathbf{w}$

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \frac{\partial L}{\partial \mathbf{w}} \quad (2.35)$$

Thuật toán 2 chỉ có thể áp dụng trong mạng thần kinh nhân tạo không có hàm kích hoạt. Gọi  $B(i)$  là các nút kết nối đến nút  $i$ ,  $A(i)$  là các nút mà  $i$  kết nối đến. Giá trị của nút trước khi qua hàm kích hoạt là  $a(i)$ , sau khi qua hàm kích hoạt là  $h(i) = \Phi_i = \Phi(a(i))$ . Khai triển của nút  $i$  sẽ là một hàm tuyến tính các nút nối đến nó  $a(i) = \sum_{j \in B(i)} w_{ji} h(j)$ . Khi sử dụng hàm kích hoạt, ta phải biến đổi thuật toán theo một trong hai cách sau:

1. Cách đầu tiên là sử dụng giá trị sau khi kích hoạt (*post-activation*). Nếu ta đặt giá trị của nút  $i$  là  $y(i) = h(i) = \Phi(a(i))$  (giá trị sau hàm kích hoạt), thì trong thuật toán 2, ta phải chú ý

- Giá trị  $z(i, j)_{j \in A(i)} = \frac{\partial y(j)}{\partial y(i)}$  thay đổi thành

$$\begin{aligned} z(i, j) &= \frac{\partial \Phi(a(j))}{\partial \Phi(a(i))} \\ &= \frac{\partial \Phi(a(j))}{\partial a(j)} \cdot \frac{\partial a(j)}{\partial \Phi(a(i))} \\ &= \Phi'_j \cdot w_{ij} \end{aligned} \quad (2.36)$$

$$\text{nên } \Delta(i) \leftarrow \sum_{j \in A(i)} [\Phi'_j \cdot w_{ij}] \Delta(j)$$

- Thay đổi

$$\left[ \frac{\partial y(i)}{\partial w_{ji}} \right]_{j \in B(i)} = \frac{\partial \Phi(a(i))}{\partial w_{ji}} = \frac{\partial \Phi(a(i))}{\partial a(i)} \cdot \frac{\partial a(i)}{\partial w_{ji}} = \Phi'_i \cdot \Phi_j \quad (2.37)$$

$$\text{nên } \frac{\partial L}{\partial w_{ji}} \leftarrow \Delta(i) [\Phi'_i \cdot \Phi_j]$$

2. Cách thứ hai là sử dụng giá trị trước khi kích hoạt (*pre-activation*). Nếu ta đặt giá trị của nút  $i$  là  $y(i) = a(i)$  (giá trị trước hàm kích hoạt), thì mỗi khi tính giá trị  $a(i)$  từ  $a(j), j \in B(i)$ , ta phải dùng hàm kích hoạt lên các giá trị đầu vào này  $y(i) = a(i) = \sum_{j \in B(i)} \Phi(a(j))$ . Lưu ý, vì ta chọn cách biến đổi  $y(i) = a(i)$  nên ở lớp ẩn đầu tiên ta sẽ không phải tính hàm kích hoạt, và ở lớp đầu ra ta phải dùng hàm kích hoạt lên lớp này rồi mới đưa kết quả vào hàm mất mát.

- Giá trị khởi tạo sẽ là

$$\begin{aligned} \frac{\partial L}{\partial y(t_r)} &= \frac{\partial L}{\partial a(t_r)} \\ &= \frac{\partial L}{\partial \Phi(a(t_r))} \cdot \frac{\partial \Phi(a(t_r))}{\partial a(t_r)} \\ &= L'(\Phi_{t_r}) \cdot \Phi'_{t_r} \end{aligned} \quad (2.38)$$

- Tương tự như trên, giá trị  $z(i, j)_{j \in A(i)}$  thay đổi thành

$$\begin{aligned} z(i, j) &= \frac{\partial a(j)}{\partial a(i)} \\ &= \frac{\partial a(j)}{\partial \Phi(a(i))} \cdot \frac{\partial \Phi(a(i))}{\partial a(i)} \\ &= w_{ij} \cdot \Phi'_i \end{aligned} \quad (2.39)$$

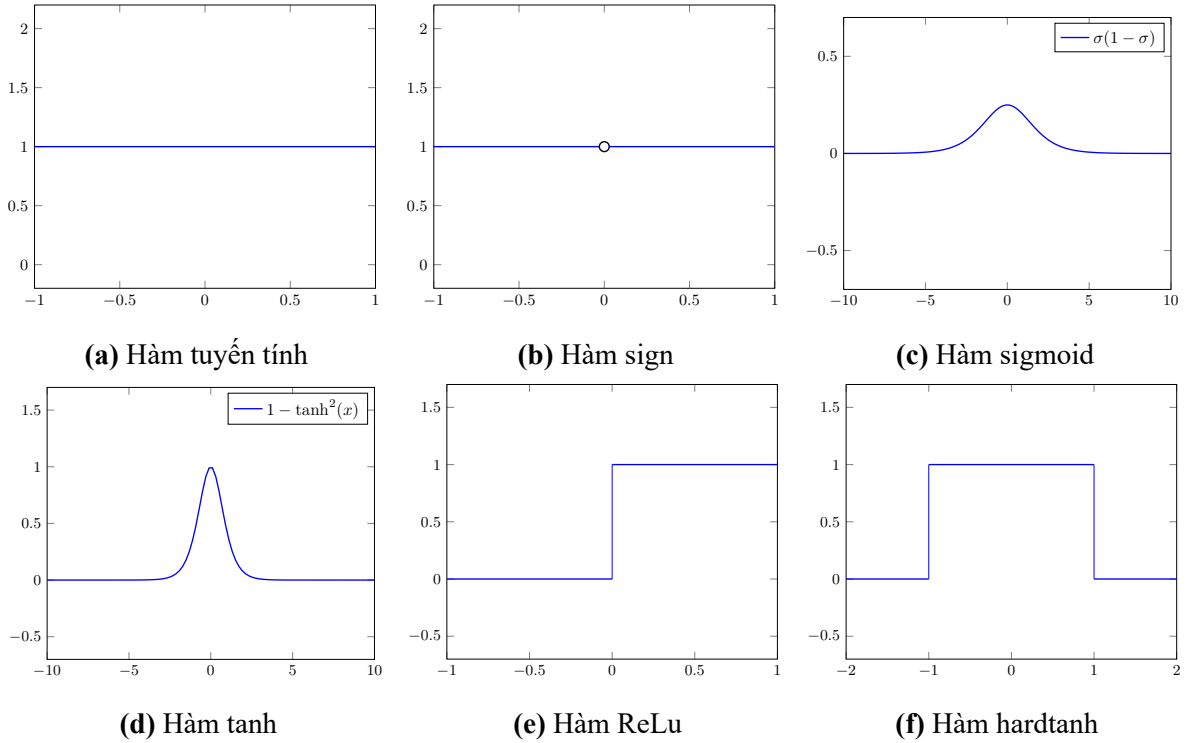
nên  $\Delta(i) \leftarrow \Phi'_i \sum_{j \in A(i)} w_{ij} \cdot \Delta(j)$

- Thay đổi

$$\left[ \frac{\partial y(i)}{\partial w_{ji}} \right]_{j \in B(i)} = \frac{\partial a(i)}{\partial w_{ji}} = \Phi_j \quad (2.40)$$

nên  $\frac{\partial L}{\partial w_{ji}} \leftarrow \Delta(i) \cdot \Phi_j$

Cả hai cách trên đều được dùng trong mạng thần kinh nhân tạo, nhưng cách sử dụng giá trị trước khi kích hoạt thường được dùng nhiều hơn. Một số đạo hàm của hàm kích hoạt  $\Phi'$  ở hình 2.21.



**Hình 2.21:** Đồ thị đạo hàm các hàm kích hoạt thông dụng

Đối với hàm kích hoạt softmax ở lớp đầu ra, ta không thể dùng thuật toán trên được mà phải thay đổi một chút ở lớp đầu ra khi đi qua hàm softmax. Gọi  $o_1, \dots, o_p$  là giá trị



xác suất của các biến đầu ra  $t_1, \dots, t_p$ , hàm softmax sẽ là

$$o_i = \frac{\exp(t_i)}{\sum_{j=1}^k \exp(t_j)} \quad (2.41)$$

Hàm softmax ở lớp đầu ra thường đi chung với hàm mất mát cross-entropy

$$L = - \sum_{i=1}^k y_i \log(o_i) \quad (2.42)$$

với  $y_i$  là mã hoá one-hot của giá trị quan sát. Đạo hàm của hàm mất mát là

$$\frac{\partial L}{\partial t_i} = \sum_{j=1}^k \frac{\partial L}{\partial o_j} \cdot \frac{\partial o_j}{\partial t_i} \quad (2.43)$$

Với đạo hàm của hàm mất mát được tính theo mỗi đầu ra  $o_j$

$$\frac{\partial L}{\partial o_j} = \frac{\partial [-y_j \log(o_j)]}{\partial o_j} = -\frac{y_j}{o_j} \quad (2.44)$$

Với hàm chỉ thị (*indicator function*)  $I(i, j)$ , đạo hàm của giá trị sau và trước lớp softmax được tính bằng cách: [8]

$$\frac{\partial \log(o_j)}{\partial t_i} = \frac{1}{o_j} \cdot \frac{\partial o_j}{\partial t_i} \quad (2.45)$$

$$\begin{aligned} \Leftrightarrow \frac{\partial o_j}{\partial t_i} &= o_j \cdot \frac{\partial \log(o_j)}{\partial t_i} = o_j \cdot \frac{\partial}{\partial t_i} \log \left[ \frac{\exp(t_j)}{\sum_{l=1}^k \exp(t_l)} \right] \\ &= o_j \cdot \left[ \frac{\partial t_j}{\partial t_i} - \frac{\partial}{\partial t_i} \log \left[ \sum_{l=1}^k \exp(t_l) \right] \right] \\ &= o_j \cdot \left[ \frac{\partial t_j}{\partial t_i} - \frac{\exp(t_i)}{\sum_{l=1}^k \exp(t_l)} \right] \\ &= o_j \cdot (I(i, j) - o_i) \end{aligned} \quad (2.46)$$

Thay vào phương trình 2.43, ta có

$$\begin{aligned} \frac{\partial L}{\partial t_i} &= \sum_{j=1}^k -\frac{y_j}{o_j} \cdot o_j \cdot (I(i, j) - o_i) \\ &= o_i \sum_{j=1}^k y_j - \sum_{j=1}^k I(i, j) y_j \\ &= o_i - y_i \end{aligned} \quad (2.47)$$

Thuật toán 2 sau khi sửa đổi để sử dụng với hàm kích hoạt  $\Phi$  là hoàn chỉnh để lập trình mạng thần kinh nhân tạo trên máy tính. Tuy nhiên thuật toán này không sử dụng các tính toán trên ma trận nên không thể tận dụng được sức mạnh phần cứng của GPU hay các thiết bị XLA. Vì thế ta cần thay đổi thuật toán về dạng ma trận (phương trình 2.20) để tối ưu sức mạnh tính toán.

Ta quy ước đạo hàm đối với một vector sẽ được viết dưới dạng ma trận sắp xếp theo mẫu số (*denominator layout*). Ví dụ ta có hai vector cột  $\mathbf{h}$  và  $\mathbf{x}$ , ma trận của đạo hàm là

$$\left[ \frac{\partial \mathbf{h}}{\partial \mathbf{x}} \right]_{ij} = \frac{\partial h_j}{\partial x_i} \quad (2.48)$$

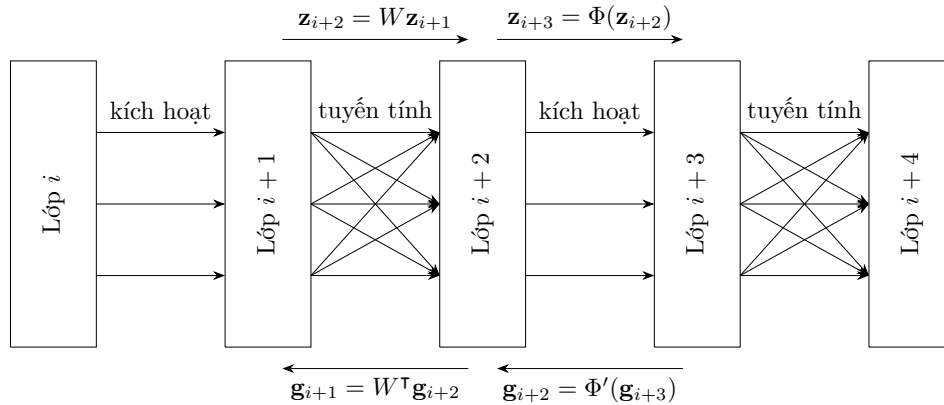
Ma trận này chính là ma trận Jacobi chuyển vị  $\left[ \frac{\partial \mathbf{h}}{\partial \mathbf{x}} \right] = J(\mathbf{h}, \mathbf{x})^\top$ .

Nếu  $\mathbf{o} = F_k(F_{k-1}(\dots F_1(\mathbf{x})))$ , với đầu vào của hàm  $F_i$  có chiều là  $n_i$ , đầu ra của hàm  $F_i$  có chiều là  $n_{i+1}$ , thì quy tắc xích khi viết dưới dạng ma trận sắp xếp theo mẫu số sẽ là

$$\underbrace{\left[ \frac{\partial \mathbf{o}}{\partial \mathbf{x}} \right]}_{n_1 \times n_{k+1}} = \underbrace{\left[ \frac{\partial \mathbf{h}_1}{\partial \mathbf{x}} \right]}_{n_1 \times n_2} \underbrace{\left[ \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1} \right]}_{n_2 \times n_3} \dots \underbrace{\left[ \frac{\partial \mathbf{h}_{k-1}}{\partial \mathbf{h}_{k-2}} \right]}_{n_{k-1} \times n_k} \underbrace{\left[ \frac{\partial \mathbf{h}_k}{\partial \mathbf{h}_{k-1}} \right]}_{n_k \times n_{k+1}} \quad (2.49)$$

Quy tắc xích này có thứ tự đảo ngược so với quy tắc xích bình thường vì ta đang quy ước ma trận của đạo hàm sẽ dưới dạng sắp xếp theo mẫu số.

Mỗi nút trong mạng thần kinh nhân tạo sẽ được tính qua hai bước là bước tổng tuyến tính và bước sử dụng hàm kích hoạt, nên ta sẽ chia mỗi lớp trong mạng thành hai lớp riêng biệt, lớp đầu tiên là lớp tuyến tính (phương trình 2.20), sau đó là lớp kích hoạt (hình 2.22).



**Hình 2.22:** Hai lớp tuyến tính và kích hoạt nằm xen kẽ nhau

## CHƯƠNG 2. NỘI DUNG

Gọi  $\mathbf{z}_i$  là vector giá trị của lớp  $i$ ,  $\mathbf{g}_i$  là vector đạo hàm  $[\partial L / \partial \mathbf{z}_i]$ , ta có

$$\mathbf{z}_{i+1} = W \mathbf{z}_i \quad [\text{Lan truyền xuôi ở lớp tuyến tính}] \quad (2.50)$$

$$\mathbf{z}_{i+1} = \Phi(\mathbf{z}_i) \quad [\text{Lan truyền xuôi ở lớp kích hoạt}] \quad (2.51)$$

$$\mathbf{g}_i = \left[ \frac{\partial \mathbf{z}_{i+1}}{\partial \mathbf{z}_i} \right] \cdot \mathbf{g}_{i+1} \quad [\text{Lan truyền ngược}] \quad (2.52)$$

Một số hàm thường gặp trong mạng thần kinh khi lan truyền xuôi và lan truyền ngược (được tính dựa vào phương trình lan truyền ngược bên trên), với phép toán  $\odot$  là tích Hadamard (*Hadamard product* hay *element-wise product*), vector  $\mathbf{1}_s$  là  $\mathbf{1}$ .

**Bảng 2.1:** Một số hàm thông dụng và lan truyền xuôi-ngược của nó [11]

Hàm	Kiểu hàm	Lan truyền xuôi	Lan truyền ngược
linear	nhiều-nhiều	$\mathbf{z}_{i+1} = W \mathbf{z}_i$	${}^1 \mathbf{g}_i = W^\top \mathbf{g}_{i+1}$
sigmoid	một-một	$\mathbf{z}_{i+1} = \text{sigmoid}(\mathbf{z}_i)$	$\mathbf{g}_i = \mathbf{g}_{i+1} \odot \mathbf{z}_{i+1} \odot (\mathbf{1} - \mathbf{z}_{i+1})$
tanh	một-một	$\mathbf{z}_{i+1} = \tanh(\mathbf{z}_i)$	$\mathbf{g}_i = \mathbf{g}_{i+1} \odot (\mathbf{1} - \mathbf{z}_{i+1} \odot \mathbf{z}_{i+1})$
ReLU	một-một	$\mathbf{z}_{i+1} = \mathbf{z}_i \odot I(z_i^{(r)} > 0)$	$\mathbf{g}_i = \mathbf{g}_{i+1} \odot I(z_i^{(r)} > 0)$
max	nhiều-một	$r = z_i^{(k)} \geq z_i^{(j)}, \forall j \in \{1, \dots, p\}$	${}^2 \mathbf{g}_i = [0, \dots, z_i^{(k)}, \dots, 0]^\top$
$f(\cdot)$ bất kỳ	bất kỳ	$\mathbf{z}_{i+1} = f(\mathbf{z}_i)$	${}^3 \mathbf{g}_i = J^\top \mathbf{g}_{i+1}$

<sup>1</sup> Do quy ước ma trận sắp xếp theo mẫu nên ta phải chuyển vị kết quả đạo hàm.

<sup>2</sup> Hàm max thực sự không có đạo hàm, vì sự thay đổi nhỏ của các giá trị  $z_i^{(j)}$  khác giá trị tối đa  $z_i^{(k)}$  không hề ảnh hưởng đến kết quả. Chỉ có duy nhất giá trị tối đa  $z_i^{(k)}$  là ảnh hưởng đến đầu ra cho nên ta cho tất cả giá trị  $g_i^{(j)} (j \neq k) = 0$ .

<sup>3</sup> Với  $J$  là ma trận Jacobi  $J_{kr} = \frac{\partial f_k(\mathbf{z}_i)}{\partial z_i^{(r)}}$

Sau khi lan truyền xuôi để tính toàn bộ  $\mathbf{z}_i$ , lan truyền ngược để tính toàn bộ  $\mathbf{g}_i$ , ta phải tính đạo hàm của hàm mất mát trên trọng số cạnh. Gọi  $p$  là nút ở lớp  $i - 1$ , giá trị nút này sẽ là  $z_{i-1 p}$ . Gọi  $q$  là nút ở lớp  $i$ , giá trị nút này sẽ là  $z_{iq}$ . Ta có

$$\frac{\partial L}{\partial w_{pq}} = \frac{\partial L}{\partial z_{iq}} \cdot \frac{\partial z_{iq}}{\partial w_{pq}} = \underbrace{g_{iq} \cdot z_{i-1 p}}_{1 \times 1} \quad (2.53)$$

Với quy ước ma trận sắp xếp theo mẫu số, ma trận đạo hàm của hàm mất mát  $L$  đối với ma trận tham số cạnh  $W$  được tính bằng tích ngoài (*outer product*) [11]

$$M = \mathbf{g}_i \cdot \mathbf{z}_{i-1}^\top \quad (2.54)$$

### 2.2.1.8 Các vấn đề gặp phải khi huấn luyện mạng thần kinh

Mạng thần kinh có hàng triệu tham số  $w_{ij}$  trong đó một số cạnh có giá trị đạo hàm  $\partial L / \partial w_{ij}$  rất lớn so với các thành phần khác, dẫn đến thuật toán xuống đồi cập nhật một phần bộ tham số quá lệch so với phần còn lại. Điều đó tương đương với việc một số lớp ảnh hưởng trực tiếp đến lớp đầu ra, còn một số lớp thì hầu như chẳng có ảnh hưởng. Điều này xuất hiện chủ yếu là do giá trị đạo hàm được tính toán bằng quy tắc xích và nhân liên tục với nhau. Xét một mạng thần kinh với  $m$  lớp, để đơn giản ta xem mỗi lớp chỉ có duy nhất 1 nút, giá trị của nút ở lớp ẩn thứ  $t + 1$  là

$$h_{t+1} = \Phi(w_{t+1}h_t)$$

Đạo hàm từ nút  $t + 1$  đối với nút  $t$  là

$$\frac{\partial h_{t+1}}{\partial h_t} = w_{t+1} \cdot \Phi'(w_{t+1}h_t)$$

Đạo hàm từ hàm mất mát đối với nút  $t$  sẽ có dạng

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial h_t} = w_{t+1} \cdot \Phi'(w_{t+1}h_t) \cdot \frac{\partial L}{\partial h_{t+1}}$$

Nếu ta khởi tạo các giá trị  $w_k$  dựa theo phân phối chuẩn, thì giá trị trung bình của  $w = 1$ . Nếu ta chọn hàm kích hoạt là hàm sigmoid, đạo hàm của nó sẽ là  $\Phi' = \sigma(1 - \sigma)$ , giá trị của hàm sigmoid nằm trong khoảng  $(0, 1)$  nên đạo hàm này có giá trị lớn nhất là  $\Phi' = 0.25 < 1$  khi  $\sigma = 0.5$ . Nghĩa là đạo hàm của lớp sau luôn bé hơn hoặc bằng 0.25 giá trị của lớp trước nó. Nếu ta đi ngược  $r$  lớp thì đạo hàm lớp  $t - r$  sẽ bé hơn hoặc bằng  $0.25^r$  giá trị lớp  $t$ . Do đó khi ta cập nhật bộ tham số bằng thuật toán lan truyền ngược thì tham số cạnh  $w$  của các lớp trước sẽ được điều chỉnh ít hơn nhiều so với các lớp sau, vấn đề này gọi là độ dốc biến mất (*vanishing gradient problem*). Nếu ta chọn một hàm kích hoạt cho ra giá trị  $\Phi' > 1$  thì trường hợp ngược lại sẽ xảy ra: tham số cạnh  $w$  của các lớp trước sẽ được điều chỉnh nhiều hơn so với các lớp sau, vấn đề này gọi là độ dốc bùng nổ (*exploding gradient problem*).

Tổng quát trong mạng thần kinh có nhiều lớp, mỗi lớp có nhiều nút. Gọi  $A$  là ma trận đạo hàm của lan truyền ngược  $\mathbf{g}_i = A\mathbf{g}_{i+1}$  đối với lớp kích hoạt. Vì là ma trận đạo hàm của lớp kích hoạt nên nó thường là ma trận chéo (bảng 2.1) nên ta có thể chéo hoá ma trận này thành  $A = PDP^{-1}$ . Quy tắc xích trong lan truyền ngược là tích liên tục của  $A$ , tương đương với  $A^t = PD^tP^{-1}$ . Nghĩa là độ dốc biến mất sẽ xảy ra khi các giá trị riêng  $\lambda$  của  $A$  nhỏ hơn 1, ngược lại sẽ xảy ra độ dốc bùng nổ. [11]

Có nhiều cách để hạn chế vấn đề này, cách đơn giản là ta phải chọn hàm kích hoạt sao cho đạo hàm của nó luôn ở lân cận 1, và đạo hàm này phải cho cả giá trị  $1^-$  và  $1^+$ . Ví

dù ta chọn các hàm kích hoạt từng phần như ReLU sẽ giá trị đạo hàm là 0 hoặc 1, nghĩa là giờ đây độ dốc bùng nổ và độ dốc biến mất sẽ ít xảy ra hơn. Nhưng đối với các giá trị âm, nó sẽ không cập nhật lại trọng số vì đạo hàm của nó bằng 0 tại các điểm này. Điều đó dẫn đến trọng số nối với nút này không được học nữa, đầu ra của nó luôn bằng 0 nên nút này không có ý nghĩa, vấn đề này gọi là nút chết (*dead neurons*). Một cách để khắc phục vấn đề này là dùng hàm ReLU bị rò rỉ (*leaky ReLU*). Trong thực tế thì không tồn tại hàm kích hoạt nào để giải quyết triệt để vấn đề này mà ta chỉ có thể làm giảm đi ảnh hưởng của nó lên mạng.

### 2.2.2 Mạng thần kinh tích chập - CNN (*Convolutional Neural Network*)

#### 2.2.2.1 Giới thiệu

Mạng thần kinh tích chập được phát triển dựa trên thí nghiệm của Hubel và Wiesel về vỏ não thị giác của mèo. Thí nghiệm này đã chỉ ra trong vỏ não thị giác của mèo có các vùng tế bào tương ứng với các vùng nhìn thấy của mắt, nghĩa là các vùng nhìn thấy khác nhau sẽ kích hoạt các vùng tế bào tương ứng của chúng. Ngoài ra thí nghiệm cũng cho thấy có một vài tế bào thì nhạy cảm hơn với các cạnh thẳng, trong khi các tế bào khác thì không. Các tế bào thần kinh này nối với nhau thành một hệ thống thần kinh sinh học hoàn chỉnh. Chính điều này đã khiến các nhà khoa học thiết kế ra mạng thần kinh tích chập với những đặc điểm giống hệ thần kinh thị giác, với hình ảnh thực được mắt nhìn thấy được thay bằng dữ liệu đầu vào có cấu trúc lưới (*grid-structured*), nghĩa là dữ liệu mà vị trí tương đối của nó so với các dữ liệu khác trong không gian là quan trọng. Ví dụ dễ thấy về dữ liệu có cấu trúc lưới là hình ảnh hai chiều, vì các pixel kề nhau thường mang màu sắc gần giống nhau, và thường nó biểu thị cho một vật thể nào đó. Bởi vì đặc trưng xử lý dữ liệu có cấu trúc lưới, do đó các dạng dữ liệu tuần tự như văn bản, chuỗi thời gian, ... cũng có thể được xử lý bằng mạng thần kinh tích chập.[11]

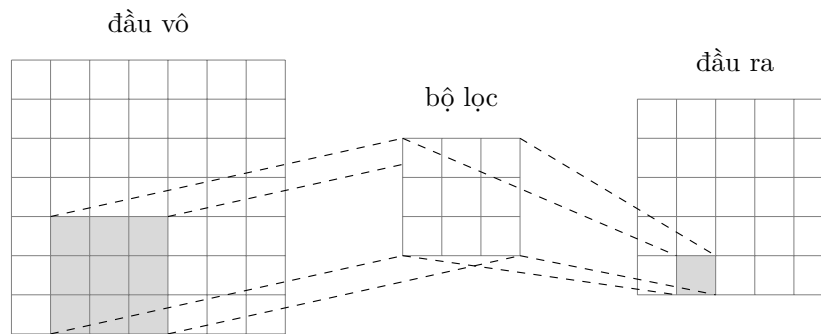
Một đặc điểm quan trọng định nghĩa mạng tích chập là phép tích chập (*convolution operation*), được biểu thị dưới dạng các lớp tích chập (*convolution layer*) trong mạng thần kinh. Phép tích chập là một phép nhân ma trận giữa một ma trận trọng số vuông có cấu trúc lưới, với một phần nhỏ đầu vào có cấu trúc lưới được lấy từ các phần khác nhau trong đầu vào hoàn chỉnh. Do đó, các mạng tích chập được định nghĩa là mạng thần kinh nhân tạo có sử dụng ít nhất một lớp tích chập.

Mạng tích chập hoạt động khá giống với một mạng thần kinh cơ bản. Khác biệt ở chỗ kết nối giữa các lớp trong mạng này được thiết kế để bảo toàn vị trí tương đối của dữ liệu (tính không gian của dữ liệu). Các kết nối giữa các lớp này thường không phải loại kết nối đầy đủ (*fully connected*) mà là kết nối thưa (*sparsely connected*). Trong mạng tích chập có ba loại lớp thường thấy: lớp tích chập (*convolution layer*), lớp gộp (*pooling layer*) và lớp ReLU (*ReLU layer*). Lớp ReLU hoạt động như một lớp ẩn trong mạng thần kinh truyền thống.

### 2.2.2.2 Kiến trúc cơ bản

Trong mạng tích chập, trạng thái của các lớp được sắp xếp dựa trên cấu trúc lưới trong không gian. Vị trí dữ liệu trong cấu trúc lưới này là quan trọng, vì một nhóm cục bộ các điểm dữ liệu gần nhau trong không gian, sẽ được ánh xạ qua phép tích chập hay phép biến đổi (*transformation*) để thành dữ liệu mới. Đối với mạng thần kinh nhân tạo, giá trị của mỗi lớp ẩn  $\mathbf{h}$  là một vector hay một tensor 1 chiều (vector), thì giá trị của mỗi lớp trong mạng tích chập là một tensor 3 chiều, bao gồm chiều rộng (*width*), chiều cao (*height*) và chiều sâu (*depth*). Với chiều rộng và chiều cao thường được xem là toạ độ trong cấu trúc lưới (toạ độ trong không gian), còn chiều sâu thể hiện cho một thuộc tính khác độc lập với không gian. Ví dụ một tấm hình  $32 \times 32$  được lưu với bảng màu RGB, thì biểu diễn của nó trong mạng tích chập là một tensor 3 chiều  $32 \times 32 \times 3$ , với mỗi lớp  $32 \times 32$  là một tensor 2 chiều tượng trưng cho ma trận vị trí, còn chiều thứ 3 biểu diễn một kênh màu căn bản (*color channel*) trong RGB. Ở đây ta ký hiệu trạng thái  $\mathbf{h}_p$  của lớp  $p$  sẽ có chiều là  $L_p \times B_p \times d_p$ , với  $L$  là chiều dài (*length*),  $B$  là chiều rộng (*breath*),  $d$  là độ sâu (*depth*). Với mỗi lớp tương ứng với  $d = 1 \dots n$  ta gọi nó là bản đồ đặc trưng (*feature maps*).

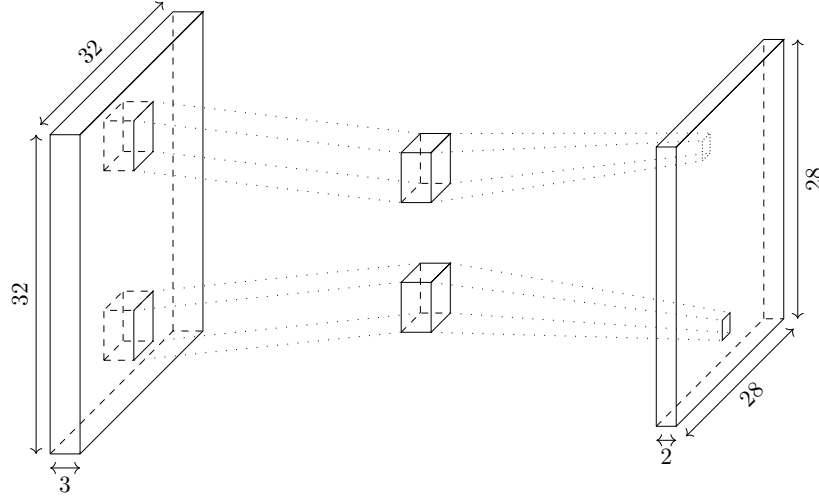
Đối với mạng thần kinh truyền thống hay mạng hồi quy, các tham số cạnh được biểu diễn dưới dạng ma trận, thì trong mạng tích chập các tham số này được biểu diễn dưới dạng tensor 3 chiều, và gọi là bộ lọc (*filters*). Một bộ lọc thường có chiều dài bằng chiều rộng, và ta ký hiệu là  $F_q \times F_q \times d_q$ . Với chiều  $F_q$  có điều kiện  $F_q < L_q, F_q < B_q, F_q$  thường là một số lẻ nhỏ, ví dụ như 3 và 5. Chiều sâu  $d_q$  phải bằng chiều sâu của đầu vào (hay bằng chiều sâu của lớp mà bộ lọc áp dụng). Bộ lọc sẽ trượt (*slide*) trên tất cả các vị trí khả dĩ của đầu vào để tạo đầu ra. Kết quả đầu ra được tính bằng cách làm phẳng (*flatten*) tensor 3 chiều  $F_p \times F_p \times d_p$  của đầu vào và bộ lọc thành 2 vector có độ lớn là  $F_p \cdot F_p \cdot d_p$ , rồi tính tích chập của hai vector này, nên kết quả đầu ra sẽ là một con số (hình 2.23).



**Hình 2.23:** Lớp tích chập với  $d_p = 1$

Có thể thấy số vị trí mà bộ lọc có thể trượt trên đầu vào chính là số chiều của đầu ra. Như trong ví dụ trên ta có chiều dài và rộng của đầu vào là  $L_p = B_p = 7$ , còn của bộ lọc

là  $F_p = 3$ , ta có thể tính được chiều dài mới sẽ là  $L_{p+1} = L_p - F_p + 1 = 5$  và chiều rộng mới sẽ là  $B_{p+1} = B_p - F_p + 1 = 5$ . Cần lưu ý vì mỗi bộ lọc sẽ làm phẳng tensor 3 chiều và tính tích chập, nên kết quả khi bộ lọc trượt trên toàn bộ dữ liệu đầu vào sẽ là một ma trận. Trong nhiều trường hợp chúng ta muốn tạo ra nhiều bản đồ đặc trưng hơn, ta phải áp dụng nhiều bộ lọc hơn cho dữ liệu đó (hình 2.24). Độ sâu ở lớp  $p + 1$  sẽ là số lượng bộ lọc chúng ta áp dụng lên lớp  $p$ , nên tổng tham số của bộ lọc tác động lên lớp  $p$  sẽ là  $F_q^2 \cdot d_q \cdot d_{q+1}$ .



**Hình 2.24:** Lớp tích chập sử dụng nhiều bộ lọc

Trong thực tế số lượng bản đồ đặc trưng trong một lớp có thể rất lớn. Lý do là vì người ta muốn áp dụng nhiều bộ lọc khác nhau để có thể xác định các mẫu không gian thuộc về một phần nhỏ của dữ liệu. Một vài bộ lọc thường dùng trong bảng 2.2.

Gọi  $W^{(p,q)} = [w_{ijk}^{(p,q)}]$  là bộ lọc thứ  $p$  của lớp  $q$ , với  $i, j, k$  lần lượt tương ứng với chiều dài, rộng và sâu. Gọi  $H^{(q)}$  là các bản đồ đặc trưng (hay trạng thái ẩn). Phép tích chập biến đổi các bản đồ đặc trưng từ lớp  $q$  đến lớp  $q + 1$  được định nghĩa như sau: [11]

$$h_{ijp}^{(q+1)} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w_{rsk}^{(p,q)} \cdot h_{i+r-1, j+s-1, k}^{(q)} \quad \forall i \in \{1, \dots, L_q - F_q + 1\} \quad (2.55)$$

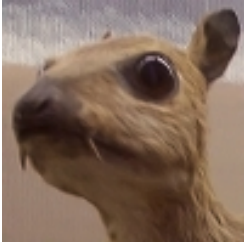
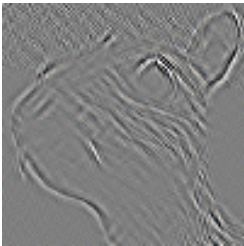
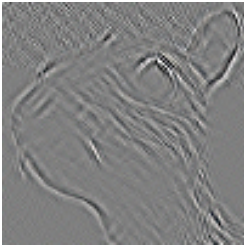
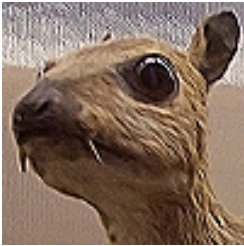

$$\forall i \in \{1, \dots, B_q - F_q + 1\}$$

$$\forall i \in \{1, \dots, d_{q+1}\}$$

Một tính chất của phép tích chập là khi ta dịch chuyển (*shift*) các điểm theo một hướng nào đó 1 đơn vị, rồi áp dụng phép tích chập, thì kết quả của phép tích chập cũng bị dịch chuyển. Nghĩa là vị trí tương đối của dữ liệu đối với nhau là không thay đổi.

Nếu ta chỉ dùng các bộ lọc  $3 \times 3$ , thì số điểm dữ liệu đầu vào qua lớp tích chập đầu tiên là  $3 \times 3$ , qua lớp tích chập thứ hai là  $5 \times 5$ , qua lớp tích chập thứ ba là  $7 \times 7$ , ... Qua càng nhiều lớp tích chập thì độ lớn từng chiều của dữ liệu ngày càng nhỏ lại. Các lớp

**Bảng 2.2:** Một số bộ lọc thường dùng [14]

Bộ lọc	Giá trị	Kết quả
Xác định ( <i>identity</i> )	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Cạnh ( <i>ridge</i> )	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Cạnh ( <i>ridge</i> )	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sắc nét ( <i>sharpen</i> )	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Mờ ( <i>box blur</i> )	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

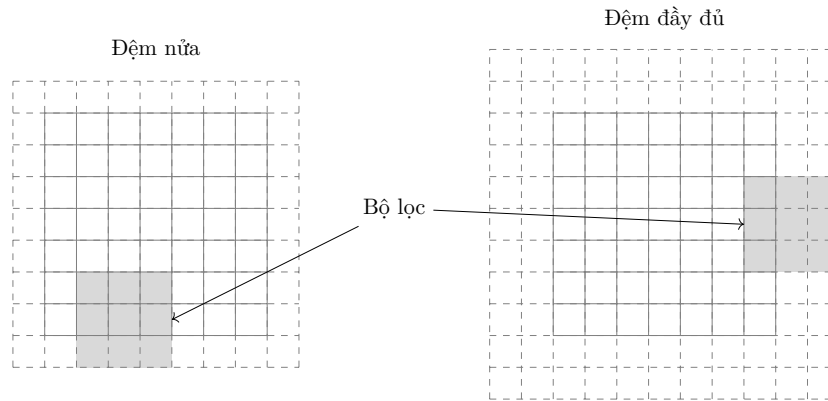


phía sau tuy sẽ mất dần dữ liệu nhưng những dữ liệu còn lại là sự tổ hợp của các hình mẫu đơn giản và quan trọng được trích xuất ở các lớp phía trước, quá trình này gọi là trích xuất thuộc tính phân cấp (*hierarchical feature engineering*).

### 2.2.2.3 Đệm - (Padding)

Xét một điểm có tọa độ không gian ở giữa dữ liệu  $(L_q/2, B_q/2)$ , một bộ lọc sẽ đi qua điểm này  $F_q \times F_q$  lần. Nhưng đối với các điểm ở cạnh hay gần cạnh của dữ liệu, bộ lọc sẽ đi qua với số lần ít hơn, như đối với các điểm góc thì bộ lọc chỉ đi qua duy nhất 1 lần. Điều này nói lên rằng phép tích chập truyền thống sẽ xem nhẹ các điểm ở gần các cạnh so với các điểm ở trung tâm. Vấn đề này được giải quyết bằng cách thêm các điểm dữ liệu rỗng bao quanh bên ngoài bộ dữ liệu cũ, và gọi các điểm dữ liệu mới này là phần đệm (*padding*). Phần đệm được đánh số là 0 nên nó không tham gia vào biến đổi dữ liệu. Có ba loại đệm thường thấy là

1. Đệm hợp lệ (*valid padding*): không thêm phần đệm, giống như cách xử lý ở phần trước. Kết quả của phép tích chập sẽ giảm số chiều còn  $(L_q - F_q + 1, B_q - F_q + 1)$ .
2. Đệm nửa (*half-padding*): phần đệm có kích thước ở mỗi cạnh là  $(F_q - 1)/2$  (vì  $F_q$  thường là lẻ). Khi một bộ lọc đi đến viền cạnh thì gần một nửa bộ lọc sẽ đi ra ngoài phần đệm nên gọi là đệm nửa. Kết quả của phép tích chập sẽ giữ nguyên số chiều là  $(L_q, B_q)$ .
3. Đệm đầy đủ (*full-padding*): phần đệm có kích thước ở mỗi cạnh là  $F_q - 1$ . Khi một bộ lọc đi đến viền cạnh thì gần hết bộ lọc sẽ đi ra ngoài phần đệm nên gọi là đệm đầy đủ. Kết quả của phép tích chập sẽ tăng số chiều lên  $(L_q + F_q - 1, B_q + F_q - 1)$



**Hình 2.25:** Đệm nửa và đệm đầy đủ

### 2.2.2.4 Bước tiến - (Strides)

Trong tất cả các ví dụ trên đều sử dụng bước tiến (*stride*) là 1. Trong nhiều trường hợp ta không cần phải áp dụng bộ lọc cho tất cả các điểm dữ liệu, ví dụ trong trường hợp chúng ta muốn giảm độ chi tiết của dữ liệu sau khi đi qua bộ lọc, ta có thể tinh chỉnh bước tiến lớn hơn 1. Khi ta sử dụng bước tiến là  $S_q$ , bộ lọc sẽ nhận dữ liệu

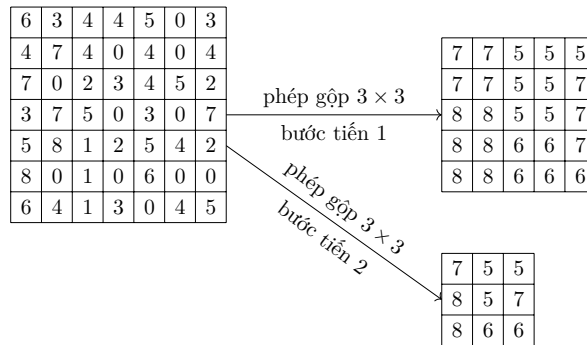
ở các điểm  $1, S_q + 1, 2S_q + 1, \dots$ . Kích thước của đầu ra bây giờ sẽ xấp xỉ bằng  $((L_q - F_q + 1)/S_q, (B_q - F_q + 1)/S_q)$ . Ý nghĩa của điều này là đầu ra sẽ có diện tích giảm đi khoảng  $S_q^2$  lần so với khi sử dụng bước tiến là 1. Thông thường người ta chỉ sử dụng bước tiến 1, trong một vài trường hợp đặc biệt là 2, và rất ít khi xài bước tiến lớn hơn.

#### 2.2.2.5 Lớp ReLU

Lớp ReLU hoạt động giống lớp kích hoạt trong mạng thần kinh truyền thống khi ta biểu diễn mạng truyền thần kinh truyền thống dưới dạng ma trận. Trong mạng truyền thống, các lớp kích hoạt và lớp tuyến tính nằm xen kẽ nhau; thì trong mạng tích chập, lớp ReLU sẽ nằm xen kẽ với các lớp tích chập và lớp gộp (*pooling layer*). Việc sử dụng hàm ReLU làm hàm mặc định cho lớp này thay vì các hàm phi tuyến như sigmoid và tanh, vì hàm ReLU đã được chứng minh là tính nhanh hơn và chính xác hơn. Việc tính nhanh và chính xác hơn giúp cho việc thiết kế mạng được sâu hơn, và có thể huấn luyện mô hình trong thời gian dài.

#### 2.2.2.6 Gộp - (Pooling)

Phép gộp là một kỹ thuật biến một diện tích  $P_q \times P_q$  ở mỗi lớp thành một số. Số chiều của kết quả sau khi gộp sẽ là  $(L_q - P_q + 1) \times (B_q - P_q + 1) \times d_q$ . Khi sử dụng bước tiến là  $S_q$  với phép gộp thì số chiều sẽ khoảng  $(L_q - P_q + 1)/S_q \times (B_q - P_q + 1)/S_q \times d_q$ . Phép gộp sẽ luôn làm giảm chiều dài và rộng của dữ liệu, nhưng không bao giờ làm giảm độ sâu của nó, vì mỗi phép gộp chỉ tác động lên một lớp duy nhất một lần. Tổng quá phép gộp sẽ biến mỗi bản đồ đặc trưng cha thành bản đồ đặc trưng con nhỏ hơn mang những thuộc tính của cha.



Hình 2.26: Gộp tối đa [11]

Thông thường trong mạng tích chập, lớp gộp sẽ nằm sau lớp tích chập và lớp ReLU. Có hai loại phép gộp thường thấy là gộp tối đa (*max pooling*) và gộp trung bình (*average pooling*). Mục đích của phép gộp tối đa là nó sẽ giảm chiều nhưng vẫn giữ lại các đặc điểm nổi trội (ví dụ như cách cạnh, các điểm sáng), điều này làm cho phép gộp tối đa đảm bảo được tính bất biến khi tịnh tiến (*translation invariance*). Trong ví dụ nhận dạng quả chuối, dù cho quả chuối có nằm góc trên của hình hay góc dưới của hình, kết quả

của phép gộp tối đa luôn giữ lại cách cạnh đặc trưng của quả chuối. Phép gộp trung bình thường ít được sử dụng hơn, và thường được sử dụng trong việc làm mượt (*smooth*) đầu ra bằng cách tính trung bình của các đầu vào. [11]

Mục đích của phép gộp khi làm giảm số chiều là để lớp đằng sau dù nhỏ hơn nhưng mang những thông tin quan trọng hơn. Phép gộp thường đi với bước tiến  $> 1$ . Như trong hình 2.26, khi sử dụng phép gộp  $3 \times 3$  với bước tiến 1, kết quả đạt được là một bản đồ đặc trưng với rất nhiều các điểm dữ liệu trùng nhau, và khi dùng phép gộp  $3 \times 3$  với bước tiến là 2, số điểm dữ liệu trùng nhau đã giảm đi đáng kể.

### 2.2.2.7 Cách sắp xếp các lớp trong mạng thần kinh tích chập

Trong mạng thần kinh tích chập thông thường bao gồm 3 loại lớp tích chập C, lớp ReLU R và lớp gộp P xếp xen kẽ nhau; dữ liệu sau khi đi qua các lớp này sẽ làm phẳng và đưa vào mạng thần kinh truyền thống F. Lớp ReLU thường đi ngay sau lớp tích chập nên hai lớp này là CR. Sau nhiều nhóm lớp tích chập - ReLU sẽ là lớp gộp, ví dụ như CR-CR-CR-P. Ví dụ mạng AlexNet sẽ có dạng CR-P-CR-P-CR-CR-CR-P-F.

### 2.2.2.8 Lan truyền ngược trong mạng thần kinh tích chập

Đối với lớp ReLU ta có thể tính đạo hàm như trong mạng thần kinh truyền thống.

Đối với lớp gộp tối đa, trong trường hợp không có bất kỳ sự chồng chéo giữa các lớp gộp, nghĩa là mỗi điểm chỉ đi qua duy nhất 1 lớp gộp, thì lan truyền ngược chỉ phụ thuộc vào vị trí của phần tử tối đa (hàm max được ghi trong bảng 2.1). Trong trường hợp có sự chồng chéo giữa các lớp gộp, gọi  $P_1, \dots, P_r$  là tất cả các lớp gộp có đi qua điểm  $h$  và tạo ra các giá trị  $h_1, \dots, h_r$  ở lớp kế tiếp. Ta tính đạo hàm của tất cả  $h_1, \dots, h_r$  tương ứng với  $h$ , rồi lấy tổng.

Đối với lớp tích chập, xét một điểm  $c$  ở lớp  $i$ , điểm này sẽ đi qua nhiều bộ lọc, mỗi bộ lọc sẽ lướt qua điểm này nhiều lần và phụ thuộc vào số bước tiến. Gọi  $S_c$  là tập hợp tất cả các điểm ở lớp  $i + 1$  được sinh ra mà có sự đóng góp của điểm  $c$ . Vì lan truyền xuôi từ  $c$  đến  $S_c$  là phép tích chập của hai vector đơn giản, nếu điểm  $r \in S_c$  thì đạo hàm  $\partial r / \partial c = w_r$ , với  $w_r$  là trọng số trong lớp lọc kết nối giữa  $c$  và  $d$ . Nếu gọi  $\delta_c$  là đạo hàm của hàm mất mát đối với điểm  $c$ , dựa trên thuật toán lan truyền ngược ta có

$$\delta_c = \sum_{r \in S_c} w_r \cdot \delta_r \quad (2.56)$$

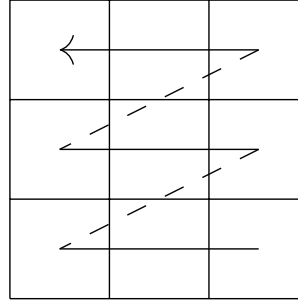
Sau khi tính đạo hàm mất mát đối với điểm, ta cần tính đạo hàm mất mát đối với trọng số  $\partial L / \partial \mathbf{w}$  trong bộ lọc. Cần lưu ý là các điểm trong dữ liệu sẽ dùng chung một bộ lọc để tạo ra một lớp bản đồ đặc trưng mới, nên ta phải chú ý các trọng số được chia sẻ và lấy tổng chúng để được kết quả cuối cùng.

Xét một phép tích chập với đầu vào  $d_q = 1$  và đầu ra  $d_{q+1} = 1$ , với bộ lọc duy nhất  $3 \times 3 \times 1$  như hình 2.27a, không sử dụng đệm và bước tiến mặc định là 1. Chọn một điểm  $c$  bất kỳ trong dữ liệu đầu vào, khi bộ lọc lần đầu tiên đi qua nút  $c$  thì vị trí tương

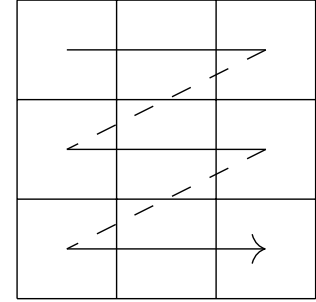
ứng của nó trong bộ lọc là  $i$ , nghĩa là trọng số nối giữa điểm  $c$  ở lớp  $q$  và một điểm  $r$  ở lớp  $q + 1$  là  $w = i$ , và theo như 2.56 thì đạo hàm tương ứng  $\partial r / \partial c = i$ . Bộ lọc này sẽ tiếp tục di chuyển và khi bộ lọc gặp điểm  $c$  thì điểm này sẽ có vị trí tương ứng với bộ lọc được biểu diễn trong hình 2.27b, và kết quả của ở lớp  $q + 1$  sẽ có vị trí tương ứng trong hình 2.27c. Nghĩa là ta có thể tạo một bộ lọc mới  $3 \times 3 \times 1$  với các giá trị  $a, b, \dots, i$  có vị

a	b	c
d	e	f
g	h	i

(a) Bộ lọc  $3 \times 3 \times 1$



(b) Vị trí tương ứng giữa điểm  $c$  và bộ lọc cùng với thứ tự xuất hiện của nó



(c) Vị trí của  $r$  và thứ tự xuất hiện của nó

**Hình 2.27:** Lan truyền ngược của bộ lọc là bộ lọc

trí tương ứng khi biến đổi hình 2.27b thành hình 2.27c. Bộ lọc này tạo thành một phép tích chập ngược từ lớp  $q + 1$  đến lớp  $q$  và gọi là bộ lọc nghịch đảo (*inverted filter*).

Xét một mạng tích chập với  $d_q \neq 1$  và  $d_{q+1} \neq 1$ . Gọi tensor 5 chiều  $\mathcal{W} = [w_{ijk}^{(p,q)}]$  là bộ lọc thứ  $p$  ở lớp thứ  $q$ , có các trọng số  $w_{ijk}$  với  $i \in \{1, \dots, L_q\}$ ,  $j \in \{1, \dots, B_q\}$ ,  $k \in \{1, \dots, d_q\}$ . Gọi một tensor 5 chiều  $\mathcal{U}$  chứa tất cả bộ lọc nghịch đảo của  $\mathcal{W}$  sẽ có thứ tự bộ lọc tương ứng là  $k$ , độ sâu của một bộ lọc tương ứng là  $p$ . Như trong hình 2.27 thì ta phải đảo chiều bộ lọc, nên tọa độ chiều dài và rộng tương ứng là  $r = F_q - i + 1$  và  $s = F_q - j + 1$ . Mỗi phần tử trong tensor  $\mathcal{U}$  sẽ có giá trị [11]

$$u_{rsp}^{(k,q+1)} = w_{ijk}^{(p,q)} \quad (2.57)$$

### 2.2.3 Mạng thần kinh hồi quy - RNN (*Recurrent Neural Network*)

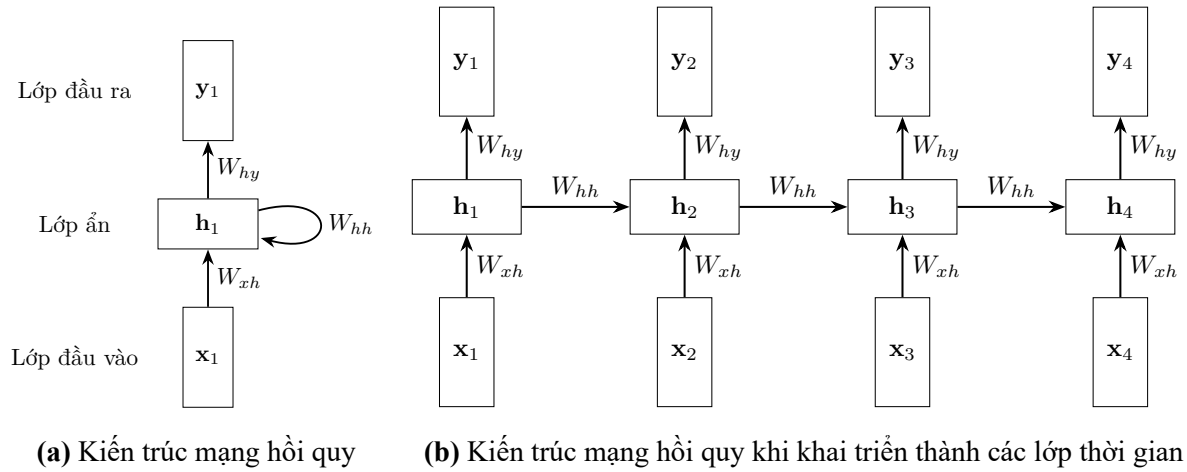
#### 2.2.3.1 Giới thiệu

Kiến trúc mạng thần kinh căn bản được thiết kế để nhận các điểm dữ liệu đầu vào có số chiều cố định, và thứ tự các điểm này không quan trọng. Tuy nhiên trong thực tế có nhiều loại dữ liệu có số chiều không cố định, hay có những dữ liệu mà thứ tự của nó quan trọng. Ví dụ dữ liệu là chuỗi các giá trị thực (*real-valued*) như chuỗi thời gian (*time-series*), hay dữ liệu là chuỗi các ký hiệu có thứ tự (*symbolic*) như văn bản, dữ liệu sinh học. Những dữ liệu này mang đặc tính thứ tự [11]:

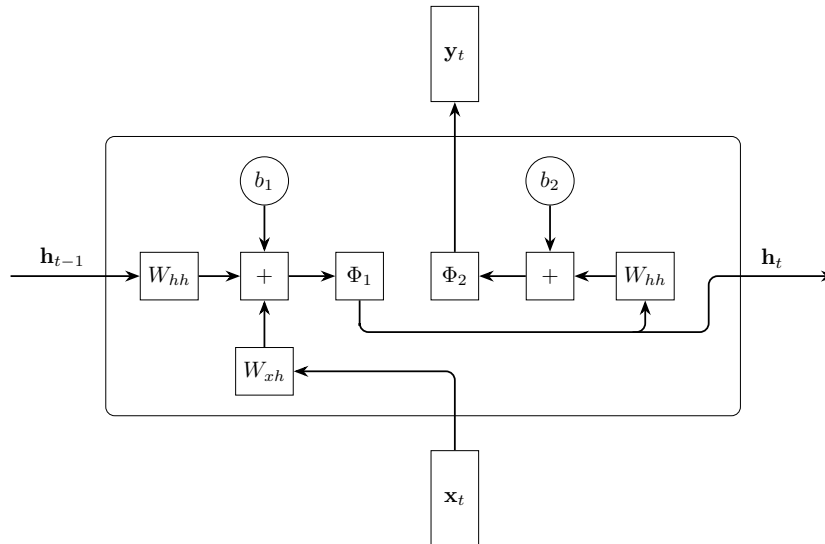
- Trong chuỗi thời gian, các điểm dữ liệu được sắp xếp có quy luật. Nếu ta hoán đổi thứ tự các điểm sẽ làm mất đi tín hiệu trong chuỗi thời gian. Ngoài ra, giá trị tại

thời điểm  $t$  của chuỗi thời gian có mối liên quan chặt chẽ đến các giá trị trong cửa sổ (*window*) trước đó.

- Khi ta thêm/bớt/đảo thứ tự một vài từ trong câu thì độ dài dữ liệu, ngữ nghĩa của câu có thể bị thay đổi hoặc không.
- Dữ liệu sinh học luôn tồn tại dưới dạng chuỗi, trong đó mỗi điểm dữ liệu là một axit amin, hoặc chi tiết hơn thì các điểm dữ liệu biểu thị cho một nucleôtit.



**Hình 2.28:** Mạng hồi quy [11]



**Hình 2.29:** Kiến trúc của lớp ẩn trong mạng hồi quy

Khi ta dùng dữ liệu ngắn hơn mạng thần kinh yêu cầu ở lớp đầu vào, ta phải thêm các điểm vô nghĩa vào; hoặc khi ta dùng dữ liệu có độ dài dài hơn, thì ta phải cắt bớt dữ liệu đi; hoặc khi ta dùng dữ liệu có tính tuần tự thì mạng thần kinh không thể xử lý tốt vì nó không quan tâm đến thứ tự các điểm nó được học. Để khắc phục điều này, người ta đã tạo ra một cấu trúc khác cho mạng thần kinh nhân tạo, đó là mạng thần kinh hồi quy (*recurrent neural network - RNN*)

Mạng thần kinh hồi quy là các mạng thần kinh nhân tạo kết nối với nhau để tạo thành đồ thị có hướng dọc theo một trình tự thời gian. Ý tưởng căn bản khi tạo ra mạng thần kinh hồi quy là xem mạng như một vòng lặp, các vòng lặp liên tiếp nhau cho phép mạng sử dụng thông tin từ các dữ liệu trước đó, và ta có thể xem nó như một loại bộ nhớ. Mạng có thể tăng giảm số lượng thần kinh tùy ý, điều này cho phép mạng tiếp nhận dữ liệu với bất kỳ chiều dài nào nên nó có thể áp dụng cho các tác vụ như nhận dạng chữ viết tay hay nhận dạng tiếng nói, là những tác vụ có tính chất kết nối, không phân đoạn.

Mạng thần kinh hồi quy mô phỏng hoạt động của não bộ con người, cho phép máy tính có thể nhận diện các khuôn mẫu sẵn có để xử lý các vấn đề thông thường. Mạng được tạo thành từ nhiều lớp thời gian (*temporal layer*) (hình 2.28b) tiếp nối nhau và có những hoạt động tương tự như hoạt động của các nơron trong não người. Từ đó, mạng thần kinh hồi quy có thể dự đoán các dữ liệu chuỗi theo một cách mà mạng thần kinh thường không thể.

### 2.2.3.2 Phân loại

Kiến trúc của RNN có thể thay đổi dựa trên bài toán cần xử lý, bài toán có thể có một hay nhiều input và output. Dưới đây là một số kiến trúc RNN [1]:

- **Một-một (one-to-one)** Chỉ có một cặp input-output ở kiến trúc này. Kiến trúc này thường được sử dụng trong các mạng neuron truyền thống.
- **Một-nhiều (one-to-many)** Một input duy nhất trong kiến trúc one-to-many có thể tạo ra rất nhiều output khác nhau. Kiến trúc này thường được sử dụng cho quá trình sản xuất nhạc.
- **Nhiều-một (many-to-one)** Ở kiến trúc này, một output duy nhất được tạo ra bằng cách kết hợp nhiều input từ các mốc thời gian khác nhau. Kiến trúc này thường được sử dụng trong các bài toán phân tích và nhận dạng cảm xúc, nơi các nhãn được định nghĩa bởi các chuỗi từ.
- **Nhiều-nhiều (many-to-many)** Kiến trúc này sử dụng một chuỗi nhiều input để sinh ra một chuỗi nhiều output. Ví dụ điển hình cho kiến trúc này là các hệ thống dịch thuật ngôn ngữ.

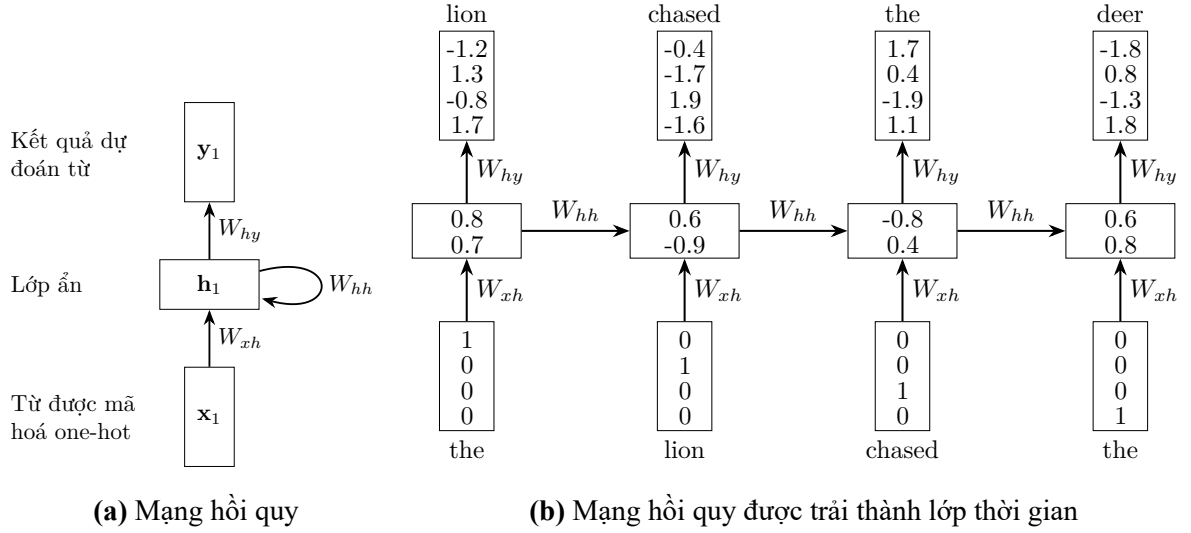
### 2.2.3.3 Kiến trúc

Dù mạng hồi quy được dùng trong hầu hết các lĩnh vực có tồn dữ liệu chuỗi, ứng dụng của nó trong lĩnh vực xử lý văn bản lại phổ biến và tự nhiên hơn cả. Dưới đây là hoạt động của mạng hồi quy trong ứng dụng dự đoán từ ngữ.

Một mạng hồi quy được biểu diễn trên hình 2.30a. Điểm mấu chốt ở đây chính là vòng lặp tại chính giữa hình, thứ thay đổi trạng thái ẩn (*hidden state*) của cả mạng sau mỗi từ trong chuỗi. Kiến trúc này trở nên rõ ràng hơn khi chúng ta “mở” vòng lặp đó ra trên một trục thời gian và biểu diễn giống như các mạng thần kinh nối tiếp nhau (hình 2.30b). Khi này chúng ta sẽ có những nút khác nhau đại diện cho lớp ẩn  $\mathbf{h}_t$  trên mỗi điểm

**Bảng 2.3:** Một số loại RNN [1]

Loại RNN	Hình minh hoạ	Ứng dụng
Một-một $T_x = T_y = 1$		Mạng thần kinh truyền thống
Một-nhiều $T_x = 1, T_y > 1$		Sinh nhạc
Nhiều-một $T_x > 1, T_y = 1$		Phân loại cảm xúc, ý kiến
Nhiều-Nhiều $T_x = T_y$		Nhận dạng tên của thực thể
Nhiều-nhiều $T_x \neq T_y$		Dịch



**Hình 2.30:** Mạng hồi quy trong dự đoán từ [11]

thời gian  $t$ . Dữ liệu đầu vào của nút thứ  $n$  là từ thứ  $n$ . Cách trình bày này tương tự hình 2.30a nhưng dễ nắm bắt hơn do có sự tương đồng với các mạng thần kinh truyền thống. Ma trận  $W$  trong các tầng khác nhau được sử dụng chung nhằm đảm bảo một hàm số duy nhất được sử dụng ở mỗi mốc thời gian. Dựa vào hình có thể thấy mỗi tầng đều có  $W_{hh}$ ,  $W_{hy}$ ,  $W_{xh}$  giống nhau.

Trong bài toán dự đoán các từ tiếp theo khi đã biết các từ trước đó, ví dụ chúng ta có một câu:

“The lion chased the deer” [11]

Khi từ “The” đi vào tầng đầu tiên, đầu ra sẽ là một vector chứa các xác suất của các từ bao gồm cả từ “lion”, và khi từ “lion” là input của tầng tiếp theo, đầu ra vẫn là một vector như vậy. Đây là một phân loại của mô hình ngôn ngữ khi mà xác suất của một từ được ước lượng dựa trên lịch sử của các từ trước đó.

Với mốc thời gian  $t$ , vector đầu vào là  $\mathbf{x}_t$ , trạng thái ẩn là  $\mathbf{h}_t$  và vector đầu ra (trong ví dụ này là xác suất các từ có khả năng xuất hiện tại thời gian mốc thời gian  $t + 1$ ) là  $\mathbf{y}_t$ . Hai vector  $\mathbf{x}_t$  và  $\mathbf{y}_t$  sẽ có số chiều giống nhau là  $d$ , với  $d$  là số từ khác nhau của câu (khi ta dùng mã hoá one-hot). Trạng thái ẩn  $\mathbf{h}$  tại  $t$  được tính từ  $\mathbf{x}_t$  và  $\mathbf{h}_{t-1}$  thể hiện dưới dạng

$$\mathbf{h}_t = W_{xh}\mathbf{x}_t + W_{hh}\mathbf{h}_{t-1} \quad (2.58)$$

$$\mathbf{y}_t = W_{hy}\mathbf{h}_{t-1} \quad (2.59)$$

Ban đầu, vector  $\mathbf{h}_0$  được khởi tạo là vector  $\mathbf{0}$  hoặc là một vector bất kỳ. Bởi vì mỗi



giá trị của  $\mathbf{h}_t$  được tính bằng  $\mathbf{h}_{t-1}$  trước đó, nên ta có thể viết

$$\begin{aligned}\mathbf{h}_t &= f(\mathbf{h}_{t-1}, \mathbf{x}_t) \\ &= \dots \\ &= f(\dots f(f(\mathbf{h}_1, \mathbf{x}_1), \mathbf{x}_2) \dots \mathbf{x}_t)\end{aligned}\tag{2.60}$$

$$= F_t(\mathbf{x}_1, \dots, \mathbf{x}_t)\tag{2.61}$$

nghĩa là mạng sẽ cho phép nhận đầu vào với độ lớn thay đổi  $\mathbf{x}_1, \dots, \mathbf{x}_t$  (2.61), và thứ tự của các đầu vào này sẽ ảnh hưởng đến kết quả cuối cùng (2.60).

#### 2.2.3.4 Lan truyền ngược theo thời gian - BPTT (Backpropagation Through Time)

Bởi vì mạng hồi quy là các mạng thần kinh truyền thông nằm kế tiếp nhau, nên bộ trọng số cạnh  $W$  sẽ là giống nhau giữa các lớp, và gọi là trọng số được chia sẻ (*shared weights*). Xét một lớp thần kinh trong mạng hồi quy, gọi  $w_{ij}^{(t)}$  là trọng số giữa hai nút  $i$  và  $j$  thuộc về lớp thời gian  $t$ , hiển nhiên ta có  $w_{ij}^{(1)} = w_{ij}^{(2)} = \dots = w_{ij}^{(T)}$ . Nếu ta xem các trọng số  $w_{ij}^{(t)}$  này không liên quan với nhau, như trong mạng thần kinh truyền thông ta có

$$\frac{\partial L}{\partial w_{ij}} = \sum_{t=1}^T \frac{\partial L}{\partial w_{ij}^{(t)}} \underbrace{\frac{\partial w_{ij}^{(t)}}{\partial w_{ij}}}_{=1} = \sum_{t=1}^T \frac{\partial L}{\partial w_{ij}^{(t)}}\tag{2.62}$$

Nghĩa là ta có thể xem trọng số được chia sẻ này là độc lập với nhau, tính toán lan truyền xuôi, sau đó tính lan truyền ngược bình thường như bình thường, cuối cùng chỉ cần lấy tổng chúng lại.

$$\frac{\partial L}{\partial W_{xh}} = \sum_{t=1}^T \frac{\partial L}{\partial W_{xh}^{(t)}}\tag{2.63}$$

$$\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^T \frac{\partial L}{\partial W_{hh}^{(t)}}\tag{2.64}$$

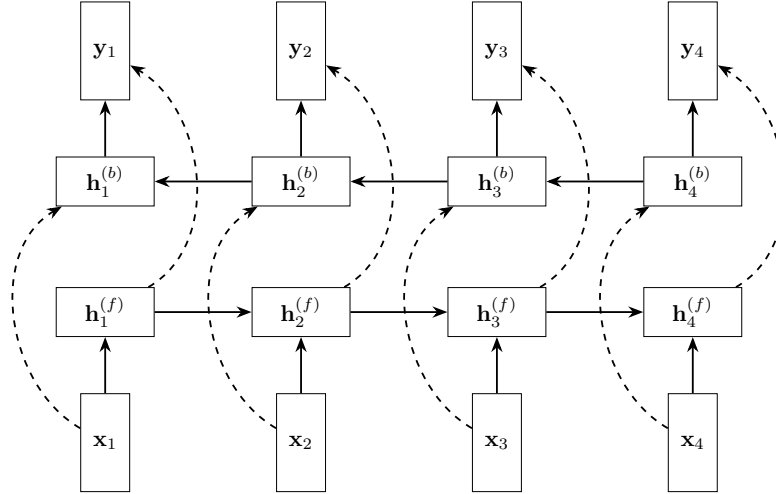
$$\frac{\partial L}{\partial W_{hy}} = \sum_{t=1}^T \frac{\partial L}{\partial W_{hy}^{(t)}}\tag{2.65}$$

Một trong những vấn đề tính toán khi huấn luyện mạng hồi quy là đầu vào có thể rất dài, do đó số lượng lớp thời gian trong mạng cũng có thể rất lớn. Điều này có thể dẫn đến các vấn đề khi phải tính toán và sử dụng bộ nhớ quá nhiều, cùng với hội tụ quá chậm. Vấn đề này được giải quyết thông qua phương pháp lan truyền ngược một phần theo thời gian (*truncated backpropagation through time*). Kỹ thuật này có thể được coi là biến thể của thuật toán xuống đồi ngẫu nhiên (*stochastic gradient descent*) cho mạng hồi quy. Trong phương pháp này, ban đầu tính lan truyền xuôi một cách bình thường, tới

bước lan truyền ngược thì ta chỉ tính đạo hàm của mất mát đối với  $k$  lớp thời gian trước đó. Nghĩa là nếu lan truyền xuôi được thực hiện qua 1000 lớp thì lan truyền chỉ cần tính từ lớp 1000 đến lớp 900 nếu  $k = 100$ , điều này làm giảm đi đáng kể khối lượng tính toán cũng như bộ nhớ cần phải lưu trữ. Trong các ứng dụng về học máy trong thời gian thực như dự báo thời tiết, bước lan truyền xuôi sẽ không được tính bằng toàn bộ đầu vào, mà ta chia đầu vào thành từng nhóm dữ liệu nhỏ, lan truyền xuôi đối với nhóm dữ liệu đó, rồi lan truyền ngược về  $k$  bước trước đó. Giá trị lan truyền xuôi cuối cùng của nhóm  $n$  là  $\mathbf{h}_{end, group=n}$  sẽ được dữ lại để đưa vào đầu vào của nhóm lan truyền xuôi kế tiếp  $n + 1$  là  $\mathbf{h}_{1, group=n+1}$ . [11]

### 2.2.3.5 Mạng thần kinh hồi quy hai chiều (Bidirectional Recurrent Neural Network)

Một nhược điểm của mạng hồi quy là trạng thái ẩn  $\mathbf{h}_t$  chỉ biết về các đầu vào trước đó đến thời điểm  $\mathbf{x}_1, \dots, \mathbf{x}_t$ , nhưng nó không biết về các trạng thái tương lai  $\mathbf{x}_i, i > t$ . Trong một số ứng dụng như suy luận về ngữ nghĩa của từ, kết quả của mạng sẽ được cải thiện đáng kể nếu biết cả thông tin về quá khứ và tương lai. Ví dụ trong cụm từ “con chó”, ta khó có thể dự đoán từ “chó” khi ta chỉ biết từ “con”, nhưng ta có thể dễ dàng dự đoán từ “con” khi đã biết từ “chó”. Mạng hồi quy không thể giải quyết vấn đề này vì nó chỉ quan tâm đến các giá trị quá khứ, trong khi trong một số ứng dụng ta cần phải có cái nhìn tổng quát xung quanh thời điểm hiện tại, nghĩa là cả quá khứ lẫn tương lai.



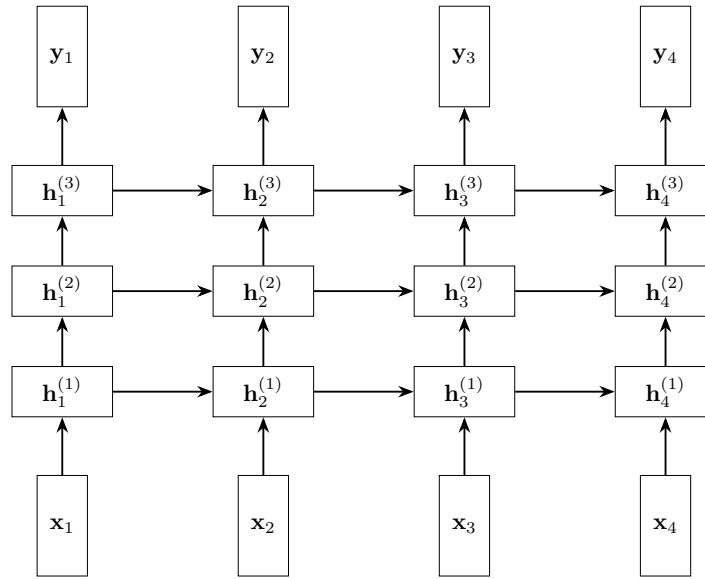
**Hình 2.31:** Kiến trúc mạng hồi quy hai chiều

Mạng hồi quy hai chiều được chứng minh là xử lý tốt đối với các ứng dụng cần biết cả giá trị quá khứ lẫn tương lai. Nhưng đối với các ứng dụng không cần biết giá trị tương lai, mạng hồi quy hai chiều nhiều trường hợp vẫn cho ra độ chính xác cao hơn. Trong thực tế, các ứng dụng cần phải quan tâm cả quá khứ lẫn tương lai của dữ liệu có thể kể đến là nhận diện chữ viết tay (*handwriting recognition*), ta dễ dự đoán nét chữ ở giữa hơn khi ta có thông tin về hai phía của nó. Mạng hồi quy hai chiều về cơ bản hoạt động như sau:

1. Tính giá trị trạng thái ẩn tiến (*forward hidden state*)  $\mathbf{h}^{(f)}$  từ  $t = 0$  đến  $t = T$  và tính giá trị trạng thái ẩn lùi (*backward hidden state*)  $\mathbf{h}^{(b)}$  từ  $t = T$  đến  $t = 0$ .
2. Tính giá trị lớp đầu ra  $\mathbf{y}_i = f(\mathbf{h}^{(f)}, \mathbf{h}^{(b)})$ .
3. Tính giá trị đạo hàm hàm mất mát đối với mỗi lớp đầu ra.
4. Tính giá trị đạo hàm hàm mất mát đối với từng trạng thái ẩn  $(\mathbf{h}^{(f)}, \mathbf{h}^{(b)})$  dùng thuật toán lan truyền ngược.
5. Tính tổng các giá trị đạo hàm hàm mất mát tương ứng với các trọng số được chia sẻ  $w$ .

### 2.2.3.6 Mạng thần kinh hồi quy nhiều lớp (Multilayer Recurrent Network)

Trong các ứng dụng thực tế, kiến trúc đa lớp mạng thần kinh được sử dụng để xây dựng các mô hình phức tạp hơn. Đối với mạng thần kinh truyền thống, lớp phía sau  $\mathbf{h}_{k+1}$  sẽ được tính bằng lớp trước nó  $\mathbf{h}_k$ . Nhưng đối với mạng hồi quy đa lớp giá trị của một lớp  $\mathbf{h}_t^{(k)}$  sẽ được tính bằng giá trị của lớp trước nó  $\mathbf{h}_t^{(k-1)}$  và giá trị của lớp thời gian trước nó  $\mathbf{h}_{t-1}^{(k)}$  (hình 2.32).



**Hình 2.32:** Kiến trúc mạng hồi quy nhiều lớp

Đầu tiên, chúng ta viết lại phương trình tính lớp ẩn của mạng hồi quy một lớp, gọi  $W_{t-1,t}^{(k)}$  là ma trận giữa lớp thời gian  $t - 1$  và  $t$ , cùng nằm trên lớp  $k$ , từ 2.58 ta có phương trình của lớp ẩn đầu tiên là

$$\mathbf{h}_t^{(1)} = [W_{xh}, W_{t-1,t}^{(1)}] \begin{bmatrix} \mathbf{x}_t \\ \mathbf{h}_{t-1}^{(1)} \end{bmatrix} \quad (2.66)$$

Đối với giá trị lớp ẩn đầu tiên, công thức sẽ khá giống bên trên. Gọi  $W_t^{(k-1,k)}$  là ma trận

trong cùng lớp thời gian  $t$ , giữa lớp  $k - 1$  và  $k$ . Gọi  $W_t^{(k)}$  là ma trận biến đổi, ta có

$$\begin{aligned} \mathbf{h}_t^{(k)} &= \begin{bmatrix} W_t^{(k-1,k)}, W_{t-1,t}^{(k)} \end{bmatrix} \begin{bmatrix} \mathbf{h}_t^{(k-1)} \\ \mathbf{h}_{t-1}^{(k)} \end{bmatrix} \\ &= W_t^{(k)} \begin{bmatrix} \mathbf{h}_t^{(k-1)} \\ \mathbf{h}_{t-1}^{(k)} \end{bmatrix} \end{aligned} \quad (2.67)$$

Thông thường số lượng lớp ẩn tầm hai hoặc ba là đủ. Đối với số lớp ẩn lớn rất dễ bị quá khớp, nên số lượng lớp ẩn tỉ lệ thuận với số lượng dữ liệu cho mô hình học.

#### 2.2.4 Bộ nhớ dài-ngắn hạn - LSTM (*Long Short-Term Memory*)

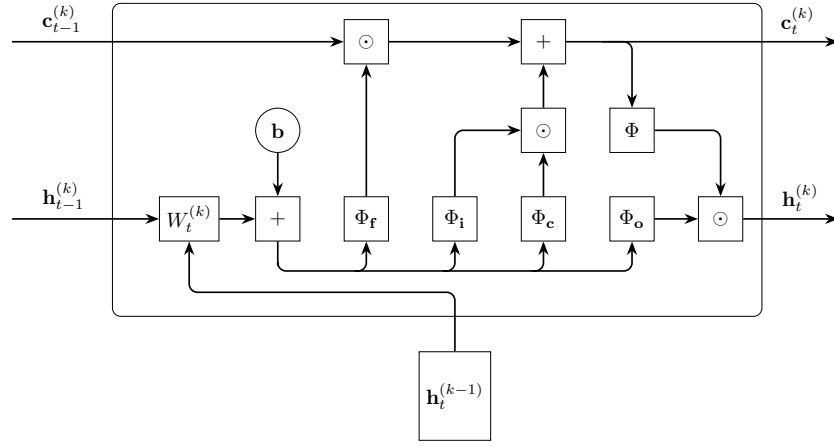
Như đã thảo luận ở phần mạng hồi quy, vấn đề đầu tiên mà chúng ta gặp phải là độ dốc biến mất hoặc độ dốc bùng nổ luôn xảy ra, và ta không thể giải quyết triệt để vấn đề này. Ngoài ra trong mạng hồi quy, như đã nói thì kiến trúc của nó làm cho nó hoạt động giống như một bộ nhớ (vì nó có ghi nhớ thông tin của dữ liệu trước đó), nhưng bộ nhớ này không phải là vô hạn. Và khi chiều dài dữ liệu tăng lên tương đương với việc số lớp tăng lên, thông tin từ dữ liệu cũ ngày càng mất giá trị so với thông tin mới nhận được. Nghĩa là ta có thể nói bộ nhớ này bị quên đi ký ức cũ khi lượng ký ức mới nạp vô càng nhiều. Ví dụ xét một mạng hồi quy khi ta cho học câu dữ liệu sau:

*“Nhà em có nuôi một con chó cỏ. Con chó có đôi mắt to, đôi tai vểnh lên và cái đuôi đen tuyền. Năm nay nó đã 10 tuổi”*

Mạng hồi quy bây giờ sẽ không biết được “nó” ở câu thứ ba đang nói về “con chó cỏ” ở câu đầu tiên, vì dữ liệu đủ dài sẽ ghi đè và làm xáo trộn dữ liệu, kết quả là làm dữ liệu cũ bị yếu đi đáng kể. Từ đó có thể nói rằng mạng thần kinh hồi quy học tốt khi sử dụng dữ liệu chuỗi ngắn, mỗi lần chỉ xử lý duy nhất một dữ liệu, và ta gọi mạng hồi quy có đặc tính của bộ nhớ ngắn hạn (*short-term memory*). Và ngược lại để sử dụng dữ liệu chuỗi dài, hay ta muốn tạo ra một mạng thần kinh có đặc tính của bộ nhớ dài hạn (*long-term memory*), ta phải giải quyết được vấn đề về việc ký ức cũ của bộ nhớ không bị mất đi khi có nhiều ký ức mới. Bộ nhớ dài-ngắn hạn được sinh ra với đặc tính lưu trữ các bộ nhớ ngắn hạn, nhưng trong khoảng thời gian dài hơn.

Một bộ nhớ dài-ngắn hạn là một phiên bản nâng cấp của mạng hồi quy nhiều lớp với sự tinh chỉnh các bước tính toán ở trong trạng thái ẩn khi nó được lan truyền. Để đạt được điều đó ta thêm vô một trạng thái ẩn khác và gọi trạng thái ẩn mới này là một ô trạng thái (*cell state*)  $\mathbf{c}_t$ . Ta có thể xem ô trạng thái này hoạt động như một bộ nhớ thông thường với hai chức năng chính là “quên” và “nhớ”.

Xét một nút ẩn bất kỳ trong mạng thần kinh nhiều lớp, phương trình để tính một trạng thái ẩn bất kỳ được đề cập ở 2.67 chỉ dùng một phép nhân ma trận, với bộ nhớ dài-ngắn hạn thì trạng thái ẩn được tính phức tạp hơn. Gọi  $\mathbf{i}$ ,  $\mathbf{f}$ ,  $\mathbf{o}$ ,  $\mathbf{c}$  là bốn vector tức thời khác nhau thể hiện cho đầu vào (*input*), quên (*forget*), đầu ra (*output*), ô trạng thái (*cell state*). Ta



Hình 2.33: Kiến trúc của LSTM

có thể xem ba vector  $\mathbf{i}$ ,  $\mathbf{f}$ ,  $\mathbf{o}$  là ba cổng logic (*logic gate*). Ba cổng này không mang thông tin mà chỉ hoạt động với mục đích là có cho phép nhận đầu vào hay không (*input gate*), có cho phép quên ký ức cũ hay không (*forget gate*), và có cho phép kết hợp với đầu ra hay không (*output gate*). Với vector  $\mathbf{c}$  là bộ nhớ tức thời của lớp này, ta có phương trình tính toán trong lớp

$$\begin{bmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{c} \end{bmatrix} = \begin{pmatrix} \text{sigmoid} \\ \text{sigmoid} \\ \text{sigmoid} \\ \Phi \end{pmatrix} \left( W_t^{(k)} \begin{bmatrix} \mathbf{h}_t^{(k-1)} \\ \mathbf{h}_{t-1}^{(k)} \end{bmatrix} + \mathbf{b} \right) \quad [\text{Tính các vector trung gian}] \quad (2.68)$$

$$\mathbf{c}_t^{(k)} = \mathbf{f} \odot \mathbf{c}_{t-1}^{(k)} + \mathbf{i} \odot \mathbf{c} \quad [\text{Tính chỉnh bộ nhớ}] \quad (2.69)$$

$$\mathbf{h}_t^{(k)} = \mathbf{o} \odot \Phi(\mathbf{c}_t^{(k)}) \quad [\text{Tính trạng thái ẩn}] \quad (2.70)$$

Nếu trạng thái ẩn có số chiều là  $p$ , thì ma trận  $W_t^{(k)}$  bây giờ sẽ có số chiều là  $4p \times 2p$ , vì nó cần tạo ra bốn vector trung gian ở đầu ra. Bộ nhớ dài ngắn hoạt động qua ba bước như sau (chi tiết trong hình 2.33):

1. Tính các vector trung gian  $\mathbf{i}$ ,  $\mathbf{f}$ ,  $\mathbf{o}$ ,  $\mathbf{c}$ .
2. Tại bước tính chỉnh bộ nhớ  $\mathbf{c}_t^{(k)}$ , có hai sự kiện sẽ xảy ra
  - $\mathbf{f} \odot \mathbf{c}_{t-1}^{(k)}$  quyết định có quên bộ nhớ cũ  $\mathbf{c}_{t-1}^{(k)}$  hay không.
  - $\mathbf{i} \odot \mathbf{c}$  quyết định có thêm dữ liệu mới  $\mathbf{c}$  vào bộ nhớ hay không.
 tổng của hai sự kiện “quên” và “nhớ” này tạo nên bộ nhớ mới.
3. Tính trạng thái ẩn sẽ là sự “rò rỉ” từ bộ nhớ  $\mathbf{c}_t^{(k)}$  qua trạng thái ẩn  $\mathbf{h}_t^{(k)}$  thông qua cổng đầu ra  $\mathbf{o}$ .

Hàm sigmoid khi tính giá trị các vector trung gian có ý nghĩa là tính ra giá trị xác suất để ta quyết định có nên thực hiện các lệnh như thêm đầu vào (*input*), quên giá trị cũ (*forget*) hay thêm vào đầu ra (*output*). Ta có thể xem các vector trung gian là các cổng

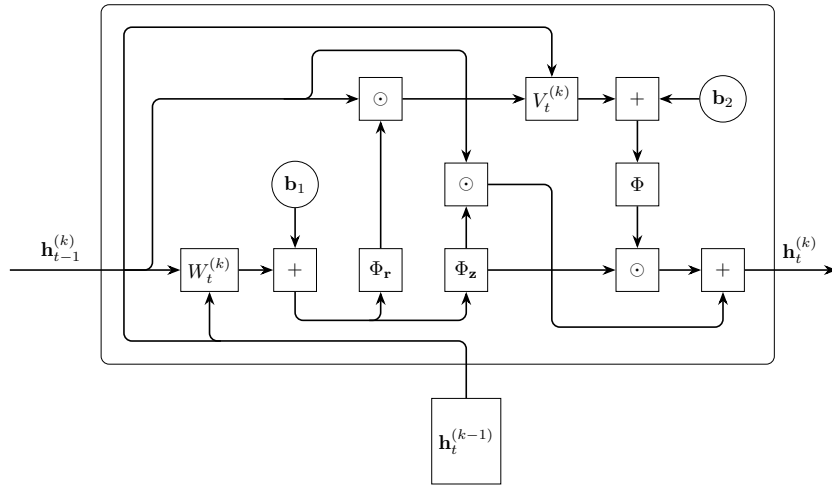
logic (*logic gate*) AND thay vì các giá trị thực từ sigmoid để đơn giản hoá sự tính toán. Nhưng trong thực tế ta vẫn phải dùng các hàm tương tự sigmoid vì nó có đạo hàm dễ tính trong lan truyền ngược. Xét một bộ nhớ dài-ngắn hạn chỉ có một lớp, ta có

$$c_t = f \cdot c_{t-1} + i \cdot c \quad (2.71)$$

Đạo hàm của bộ nhớ ở lớp thời gian  $t + 1$  so với lớp thời gian  $t$  sẽ là  $f$ , nghĩa là dòng lan truyền ngược đối với bộ nhớ  $c_t$  sẽ phụ thuộc hoàn toàn vào  $f$ . Ta chỉ cần đặt hệ số  $\mathbf{b}_f$  tại bước tính các vector trung gian đủ cao để giá trị của nó sau khi qua hàm sigmoid sẽ gần bằng 1, điều này làm cho vấn đề độ dốc biến mất xảy ra rất chậm.

### 2.2.5 Bộ nhớ tái phát - GRU (*Gated Recurrent Unit*)

Bộ nhớ tái phát có thể được xem xét là một phiên bản đơn giản của bộ nhớ dài-ngắn hạn. Đối với bộ nhớ dài-ngắn hạn, nó điều chỉnh trực tiếp lượng thông tin của trạng thái ẩn bằng cổng quên và cổng đầu ra, còn bộ nhớ tái phát chỉ sử dụng một cổng đặt lại (*reset gate*). Điểm khác biệt lớn là bộ nhớ tái phát không dùng ô trạng thái làm bộ nhớ.



Hình 2.34: Kiến trúc của GRU

Xét một nút ẩn bất kỳ trong mạng thần kinh nhiều lớp, thay vì phải tính đến bốn vector trung gian thì bộ nhớ tái phát chỉ tính hai vector trung gian là cổng đặt lại và cổng cập nhật (*update gate*)

$$\begin{bmatrix} \mathbf{z} \\ \mathbf{r} \end{bmatrix} = \begin{pmatrix} \text{sigmoid} \\ \text{sigmoid} \end{pmatrix} \left( W_t^{(k)} \begin{bmatrix} \mathbf{h}_t^{(k-1)} \\ \mathbf{h}_{t-1}^{(k)} \end{bmatrix} + \mathbf{b}_1 \right) \quad [\text{Tính các vector trung gian}] \quad (2.72)$$

$$\mathbf{h}_t^{(k)} = \mathbf{z} \odot \mathbf{h}_{t-1}^{(k)} + (\mathbf{1} - \mathbf{z}) \odot \Phi \left( V_t^{(k)} \begin{bmatrix} \mathbf{h}_t^{(k-1)} \\ \mathbf{r} \odot \mathbf{h}_{t-1}^{(k)} \end{bmatrix} + \mathbf{b}_2 \right) \quad [\text{Tính trạng thái ẩn}] \quad (2.73)$$

Ma trận  $W_t^{(k)}$  bây giờ chỉ có số chiều là  $2p \times 2p$  vì nó chỉ cần tạo ra hai vector trung gian ở đầu ra, còn ma trận  $V_t^{(k)}$  sẽ có số chiều là  $p \times 2p$ . Vì bộ nhớ tái phát không sử dụng ô trạng thái để làm bộ nhớ, nên hai cổng của nó phải đảm nhiệm nhiệm vụ này. Bộ nhớ tái phát hoạt động qua hai bước như sau:

1. Tính các vector trung gian  $\mathbf{z}, \mathbf{r}$
2. Tại bước tính trạng thái ẩn  $\mathbf{h}_t^{(k)}$  sẽ có hai việc xảy ra
  - Cổng đặt lại  $\mathbf{r}$  sẽ quyết định có bao nhiêu dữ liệu cũ được truyền vô trạng thái ẩn  $\mathbf{r} \odot \mathbf{h}_{t-1}^{(k)}$ , và nó sẽ thông qua một phép nhân ma trận  $V$  để kết hợp trạng thái ẩn từ lớp thời gian trước đó  $\mathbf{h}_{t-1}^{(k)}$  với trạng thái ẩn từ lớp trước đó  $\mathbf{h}_t^{(k-1)}$  với nhau.
  - Cổng cập nhật  $\mathbf{z}$  xem tổng đóng góp của hai trạng thái ẩn (trạng thái ẩn từ lớp thời gian trước đó  $\mathbf{h}_{t-1}^{(k)}$  và kết quả vừa tính ở bước trên) là bằng 1. Cổng cập nhật sẽ chia ra hai phần  $\mathbf{z}$  và  $1 - \mathbf{z}$  như là phần bù của nhau để quyết định sẽ lấy dữ liệu của mỗi bên đóng góp như thế nào.

Như có thể thấy cổng cập nhật làm hai nhiệm vụ, đầu tiên là hoạt động giống như một cổng đầu vào  $\mathbf{z}$ , thứ hai là hoạt động giống như một cổng quên  $1 - \mathbf{z}$ . Xét một bộ nhớ tái phát với duy nhất một lớp với  $v_1, v_2$  là hai trọng số của ma trận  $V$ , ta có

$$h_t = z \cdot h_{t-1} + (1 - z) \cdot \Phi(v_1 x_t + v_2 r h_{t-1} + b) \quad (2.74)$$

Đạo hàm giữa hai nút trong bộ nhớ tái phát sẽ là

$$\frac{\partial h_t}{\partial h_{t-1}} = z + v_2 r (1 - z) \cdot \Phi' \quad (2.75)$$

Như đã nói dòng chảy của đạo hàm trong lan truyền ngược sẽ phụ thuộc vào đạo hàm giữa hai nút kề nhau. Thành phần  $z \in (0, 1)$  sẽ làm cho lan truyền này ổn định hơn vì nó làm cho đạo hàm khó bằng 0. Ngoài ra khi  $z$  quá nhỏ thì thành phần  $(1 - z)$  sẽ đóng vai trò của  $z$  trong trường hợp trên, và nó là thành phần làm cho dòng chảy này ổn định, nên tổng của hai thành phần này sẽ làm cho đạo hàm xấp xỉ 1. Điều đó giải quyết vấn đề độ dốc biến mất trong mạng thần kinh.

## 2.3 XỬ LÝ NGÔN NGỮ TỰ NHIÊN

### 2.3.1 Xử lý ngôn ngữ tự nhiên

#### 2.3.1.1 Định nghĩa

Xử lý ngôn ngữ tự nhiên (Natural language processing - NLP) là một lĩnh vực phụ liên ngành của khoa học máy tính và tìm kiếm thông tin. Mục tiêu chính của nó là cung cấp cho máy tính khả năng hỗ trợ và thao tác với ngôn ngữ của con người. NLP liên quan đến việc xử lý các tập dữ liệu ngôn ngữ tự nhiên, chẳng hạn như kho văn bản hoặc kho dữ liệu giọng nói, bằng cách sử dụng các phương pháp học máy dựa trên quy tắc hoặc

xác suất (tức là thống kê và gần đây nhất là dựa trên mạng nơ-ron). Mục tiêu là giúp máy tính có khả năng “hiểu” nội dung của các tài liệu, bao gồm cả sắc thái theo ngữ cảnh của ngôn ngữ trong đó. Vì mục đích này, xử lý ngôn ngữ tự nhiên thường vay mượn các ý tưởng từ ngôn học lý thuyết. Sau đó, công nghệ có thể trích xuất chính xác thông tin và hiểu biết có trong tài liệu, cũng như phân loại và tổ chức các tài liệu đó.

Những thách thức trong xử lý ngôn ngữ tự nhiên thường liên quan đến nhận dạng giọng nói, hiểu ngôn ngữ tự nhiên và tạo ngôn ngữ tự nhiên.

NLP có nguồn gốc từ những năm 1940 khi Alan Turing đã xuất bản một bài báo có tựa đề “Computing Machinery and Intelligence” đề xuất thử hiện nay được gọi là Bài kiểm tra Turing làm tiêu chuẩn cho trí thông minh, mặc dù vào thời điểm đó nó không được coi là một vấn đề riêng biệt so với trí tuệ nhân tạo. Bài kiểm tra được đề xuất bao gồm một nhiệm vụ liên quan đến việc giải nghĩa và tạo ra ngôn ngữ tự nhiên một cách tự động. Sau đó quá trình phát triển của NLP gồm có 3 giai đoạn chính:

- Symbolic NLP (1950 - đầu 1990): Tiền đề của NLP được trình bày bởi thí nghiệm Chinese Room của John Searle: Cung cấp cho máy tính một bộ quy tắc (ví dụ: sách hội thoại tiếng Trung, với các câu hỏi và câu trả lời tương ứng), máy tính mô phỏng khả năng hiểu ngôn ngữ tự nhiên (hoặc các tác vụ NLP khác) bằng cách áp dụng các quy tắc đó vào dữ liệu nó gặp phải.
- Statistical NLP (1990s-2010s): Cho đến những năm 1980, hầu hết các hệ thống xử lý ngôn ngữ tự nhiên đều dựa trên các bộ quy tắc phức tạp được viết bằng tay. Tuy nhiên, bắt đầu từ cuối những năm 1980, đã có một cuộc cách mạng trong lĩnh vực xử lý ngôn ngữ tự nhiên với việc đưa ra các thuật toán học máy cho xử lý ngôn ngữ. Điều này là do sự gia tăng ổn định về sức mạnh tính toán (định luật Moore) và sự giảm dần vai trò thống trị của các lý thuyết ngôn ngữ học theo Chomsky, những lý thuyết này không khuyến khích ngôn ngữ học dựa trên kho dữ liệu (corpus linguistics) - nền tảng của phương pháp học máy trong xử lý ngôn ngữ ngày nay.
- Neural NLP (hiện tại): Vào năm 2003, mô hình n-gram là thuật toán thống kê tốt nhất để xử lý ngôn ngữ tự nhiên. Tuy nhiên, Yoshua Bengio cùng các cộng sự đã vượt qua mô hình này bằng cách sử dụng mạng Perceptron đa lớp (có một lớp ẩn và khả năng xử lý ngữ cảnh của nhiều từ, được huấn luyện trên 14 triệu từ). Đến năm 2010, Tomáš Mikolov (khi đó là nghiên cứu sinh tiến sĩ tại Đại học Công nghệ Brno) cùng các cộng sự đã áp dụng mạng nơ-ron hồi quy đơn giản với một lớp ẩn vào xử lý ngôn ngữ. Sau đó, ông tiếp tục phát triển Word2vec. Và như thế, trong những năm tiếp theo của thập kỷ, việc học biểu diễn và các phương pháp học máy theo phong cách mạng nơ-ron sâu (có nhiều lớp ẩn) đã trở nên phổ biến trong lĩnh vực xử lý ngôn ngữ tự nhiên.



### 2.3.1.2 Các bước xử lý trong xử lý ngôn ngữ tự nhiên

**Phân tích hình thái:** Đây là bước đầu tiên trong xử lý văn bản, nhằm nhận biết, phân tích và miêu tả cấu trúc của từng đơn vị ngôn ngữ, chẳng hạn như từ gốc, biên từ, phụ tố, từ loại, v.v. Các bài toán điển hình trong phần này là tách từ (word segmentation) và gán nhãn từ loại (part-of-speech tagging) trong xử lý tiếng Việt.

**Phân tích cú pháp:** Phân tích cú pháp là quá trình phân tích một chuỗi các biểu tượng ngôn ngữ theo cú pháp của ngôn ngữ. Đầu vào là một câu văn và đầu ra là một cây phân tích thể hiện cấu trúc cú pháp của câu đó. Các phương pháp thường dùng trong phân tích cú pháp là văn phạm phi ngữ cảnh (context-free grammar - CFG), văn phạm danh mục kết nối (combinatory categorial grammar - CCG), và văn phạm phụ thuộc (dependency grammar - DG).

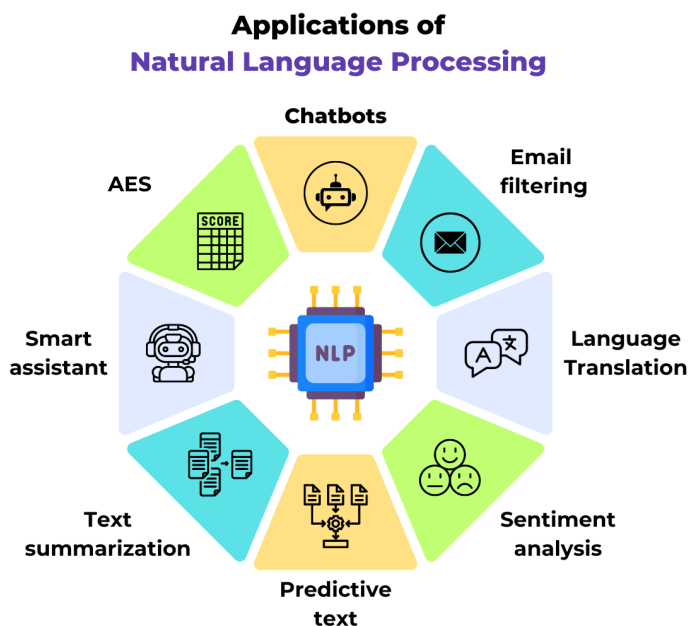
**Phân tích ngữ nghĩa:** Phân tích ngữ nghĩa nhằm tìm ra ý nghĩa của các đơn vị ngôn ngữ và xác định mối quan hệ ngữ nghĩa giữa chúng, từ cấp độ từ vựng đến cấp độ cú pháp. Điều này liên quan đến việc hiểu được các từ và câu trong bối cảnh ngữ nghĩa của chúng.

**Phân tích diễn ngôn:** Phân tích diễn ngôn xem xét mối quan hệ giữa ngôn ngữ và ngữ cảnh sử dụng, thực hiện ở mức đoạn văn hoặc toàn bộ văn bản thay vì chỉ ở mức câu. Điều này giúp hiểu rõ hơn ý nghĩa và tầm quan trọng của một đoạn văn trong ngữ cảnh tổng thể của văn bản.

### 2.3.1.3 Một vài ứng dụng của xử lý ngôn ngữ tự nhiên

Xử lý ngôn ngữ tự nhiên được ứng dụng rộng rãi trong nhiều khía cạnh của cuộc sống và công việc. Trong suốt hơn nửa thế kỷ, NLP đã phát triển đáng kể và đem lại nhiều ứng dụng đột phá.

- Nhận dạng giọng nói (Automatic Speech Recognition - ASR, hoặc Speech To Text - STT) là một trong những ứng dụng tiêu biểu của xử lý ngôn ngữ tự nhiên. Với sự tiến bộ của các thuật toán và công nghệ, việc chuyển đổi từ tiếng nói sang văn bản đã trở nên chính xác hơn. Hệ thống nhận dạng giọng nói ngày càng phổ biến trong các ứng dụng hỗ trợ khách hàng, điều khiển thiết bị qua giọng nói và ghi chú cuộc họp.
- Tổng hợp giọng nói (Speech synthesis hoặc Text to Speech – TTS) là một ứng dụng ngược lại, cho phép máy tính đọc văn bản một cách tự nhiên và sinh động. Công nghệ tổng hợp giọng nói đang ngày càng tiến xa, giúp giọng nói của các chatbot và trợ lý ảo trở nên sống động và giống thực tế hơn, tạo sự tương tác tự nhiên và gần gũi hơn với con người.
- Trả lời câu hỏi (Question Answering - QA) là một bài toán phức tạp trong xử lý ngôn ngữ tự nhiên, yêu cầu máy tính hiểu và trả lời câu hỏi dưới dạng ngôn ngữ tự nhiên. Một hệ thống QA hiệu quả cần kết hợp các phương pháp như trích xuất



**Hình 2.35:** Minh họa một số ứng dụng của xử lý ngôn ngữ tự nhiên

thông tin, phân tích ngữ pháp và xử lý ngữ nghĩa. Đối với những câu hỏi đơn giản, hệ thống trả lời câu hỏi có thể đưa ra câu trả lời chính xác. Tuy nhiên, với những câu hỏi phức tạp hơn hoặc yêu cầu hiểu sâu hơn, các thách thức vẫn còn tồn tại và đòi hỏi sự phát triển thêm của xử lý ngôn ngữ tự nhiên.

- Tóm tắt văn bản tự động (Automatic Text Summarization) là một ứng dụng quan trọng trong việc thu gọn và tóm tắt thông tin quan trọng từ văn bản gốc. Có hai phương pháp chính trong tóm tắt, là trích xuất và tóm lược ý. Trong trích xuất, hệ thống sẽ chọn các câu hoặc đoạn văn từ văn bản gốc, giữ lại những thông tin quan trọng nhất. Trong tóm lược ý, hệ thống sẽ tạo ra những câu tóm tắt mới dựa trên ngữ cảnh của văn bản.
- Chatbot là một trong những ứng dụng phổ biến nhất của xử lý ngôn ngữ tự nhiên. Nhờ vào sự tiến bộ của các mô hình học máy và xử lý ngôn ngữ tự nhiên, chatbot ngày càng trở nên thông minh và có khả năng giao tiếp một cách tự nhiên. Chatbot được sử dụng trong nhiều lĩnh vực như hỗ trợ khách hàng, dự đoán thời tiết, đặt lịch hẹn và nhiều ứng dụng thú vị khác.
- Dịch máy (Machine Translation - MT) là một trong những ứng dụng cổ điển nhất và đồng thời cũng là một trong những thách thức lớn trong xử lý ngôn ngữ tự nhiên. Với sự ra đời của mô hình dịch máy sử dụng mạng nơ-ron (neural machine translation), chất lượng dịch ngày càng được cải thiện và có thể đáp ứng nhu cầu của người dùng trong nhiều tình huống giao tiếp quốc tế.
- Tạo hình ảnh từ văn bản (Text-to-image generation): Mô hình này có khả năng nhận

dạng mô tả bằng ngôn ngữ tự nhiên và tạo ra hình ảnh tương ứng với mô tả đó. Sự phát triển của các ứng dụng này bắt đầu vào giữa những năm 2010, cùng với sự bùng nổ của Trí tuệ Nhân tạo (AI) nhờ những tiến bộ trong lĩnh vực mạng nơ-ron sâu. Đến năm 2022, chất lượng đầu ra của các mô hình tiên tiến như DALL-E 2 của OpenAI, Imagen của Google Brain, Stable Diffusion của Stability AI và Midjourney đã được đánh giá là gần tương đương với ảnh chụp thực tế và tranh vẽ của con người.

- **Phân tích cảm xúc:** Phân tích cảm xúc được sử dụng để xác định và phân loại ý định cảm xúc ẩn chứa trong văn bản. Kỹ thuật này bao gồm việc phân tích văn bản để xác định xem cảm xúc được thể hiện là tích cực, tiêu cực hay trung lập. Các mô hình phân loại cảm xúc thường sử dụng các đầu vào như chuỗi n-gram của từ, đặc trưng do con người tạo hoặc sử dụng các mô hình học sâu được thiết kế để nhận dạng cả các phụ thuộc dài hạn và ngắn hạn trong chuỗi văn bản. Ứng dụng của phân tích cảm xúc rất đa dạng, mở rộng sang các nhiệm vụ như phân loại đánh giá của khách hàng trên các nền tảng trực tuyến khác nhau.

### 2.3.2 Kỹ thuật nhúng từ (*Word embedding*)

#### 2.3.2.1 Vấn đề đặt ra

Máy móc vốn không thể hiểu được ngôn ngữ của con người, do vậy mà để máy móc xử lý ngôn ngữ thì chúng ta cần phải xử lý dữ liệu trước. Thông thường chúng ta có một số cách biểu diễn từ như sau: [3]

- Biểu diễn từ dưới dạng một con số.
- Sử dụng one-hot encoding.
- Sử dụng word embedding.

Thông thường, trong học máy, các từ sẽ được biểu diễn dưới dạng one-hot encoding, dữ liệu của các từ sẽ được chuyển về dạng các vector với số chiều tương ứng với tổng số từ. Tuy nhiên cách thể hiện one-hot encoding có một số vấn đề: [4]

- Chi phí lớn: Do số chiều của vector tương ứng với tổng số từ trong dữ liệu, độ phức tạp của vector có thể rất lớn, kéo theo đó là chi phí tính toán và lưu trữ.
- Không chứa thông tin: Do chỉ được biểu diễn dưới dạng vector mang hầu như chỉ số 0, các từ sau khi đưa về dạng one-hot encode hầu như không mang thông tin giá trị hay sự liên kết giữa các từ với nhau.
- Thiếu đi sự khái quát: Trong ngôn ngữ thì có rất nhiều từ có nghĩa giống nhau, và khi biểu diễn bằng one-hot encoding sẽ không thể thể hiện sự giống nhau này.

Đối với cách biểu diễn với những con số, đây tuy là một cách đơn giản và tiết kiệm chi phí, tuy nhiên lại có thể dẫn tới sai lệch về sự liên quan giữa các từ. Ví dụ từ “mèo” được thể hiện với số 1 và “chó” với số 2, chúng ta sẽ nhận được kết quả: “mèo” + “mèo” = “chó”. Đây là một ví dụ nhỏ về sự sai lệch khi biểu diễn dưới dạng số. [3]

Ở word embedding, kỹ thuật này gán mỗi từ với một vector, và các từ có quan hệ với

nhau cũng được tính toán để các vector cũng có thể thể hiện được quan hệ tương đồng của các từ.

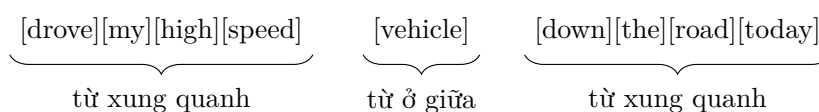
### 2.3.2.2 Giới thiệu

Word Embedding là một không gian vector dùng để biểu diễn dữ liệu có khả năng miêu tả được mối liên hệ, sự tương đồng về mặt ngữ nghĩa, văn cảnh(*context*) của dữ liệu. Không gian này bao gồm nhiều chiều và các từ trong không gian đó mà có cùng văn cảnh hoặc ngữ nghĩa sẽ có vị trí gần nhau. Ví dụ như ta có hai câu : “Hôm nay ăn táo” và “Hôm nay ăn xoài”. Khi ta thực hiện Word Embedding, “táo” và “xoài” sẽ có vị trí gần nhau trong không gian chúng ta biểu diễn do chúng có vị trí giống nhau trong một câu. [4]

### 2.3.2.3 Một số phương pháp

Có 2 phương pháp chủ yếu được sử dụng trong word embedding là Count based method và Predictive method. Các phương pháp này đều dựa trên một giả thuyết đó là những từ nào xuất hiện cùng nhau trong một ngữ cảnh sẽ có vị trí gần nhau trong không gian vector. [4]

1. Count-based method: Phương pháp này tính toán mức liên quan về mặt ngữ nghĩa giữa các từ bằng cách thống kê số lần đồng xuất hiện của một từ so với các từ khác. Ví dụ chúng ta có 2 câu: “Data is next oil” và “Data is future”, dựa trên 2 câu này chúng ta có thể tạo nên một ma trận tần suất đồng thời ở bảng 2.4. Từ bảng có thể thấy hai từ “oil” và “future” có vị trí khá gần nhau trên không gian vector.
2. Predictive method: Khác so với Count-based method, Predictive method tính toán sự tương đồng ngữ nghĩa giữa các từ để dự đoán từ tiếp theo bằng cách đưa qua một mạng neural network có một hoặc vài layer dựa trên input là các từ xung quanh (*context word*). Một context word có thể là một hoặc nhiều từ khác nhau. Chúng ta sẽ đi sâu hơn khi tìm hiểu phần tiếp theo, Word2vec.



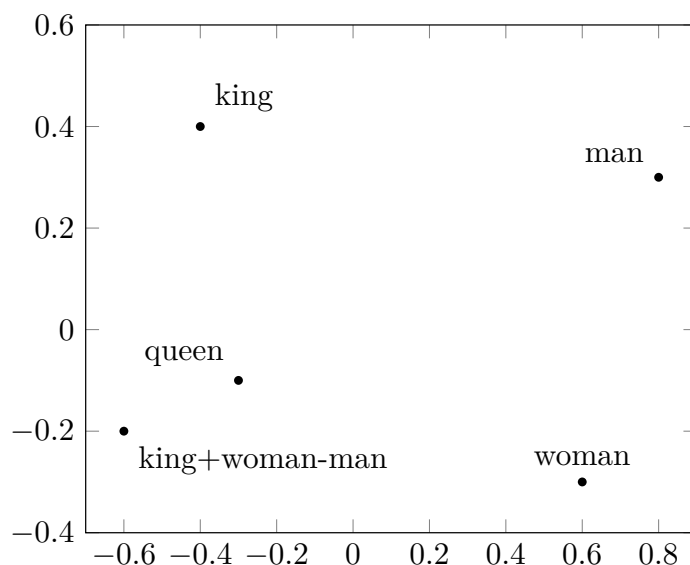
**Hình 2.36:** Từ được dự đoán và các từ xung quanh

### 2.3.2.4 Mô hình Word2vec

Trong nhiều ứng dụng NLP (*natural language processing*), các từ thường chỉ được biểu diễn bởi one-hot encoding và không giữ được quan hệ giữa các từ. Lí do để lựa chọn one-hot encoding là sự đơn giản và hiệu quả của chúng. [9] Tuy nhiên với sự cải thiện của lĩnh vực học máy, các thuật toán phức tạp được huấn luyện trên các tập dữ liệu lớn đã hoạt động hiệu quả hơn các mô hình đơn giản. Tiêu biểu là thuật toán Word2vec có thể nắm bắt được cả sự tương đồng giữa ngữ pháp và ngữ nghĩa của các từ ngữ. Và

một ví dụ được biết đến rộng rãi của Word2vec là:

$$\overline{King} - \overline{Man} = \overline{Queen} - \overline{Woman}$$



**Hình 2.37:** Ví dụ về Word2vec

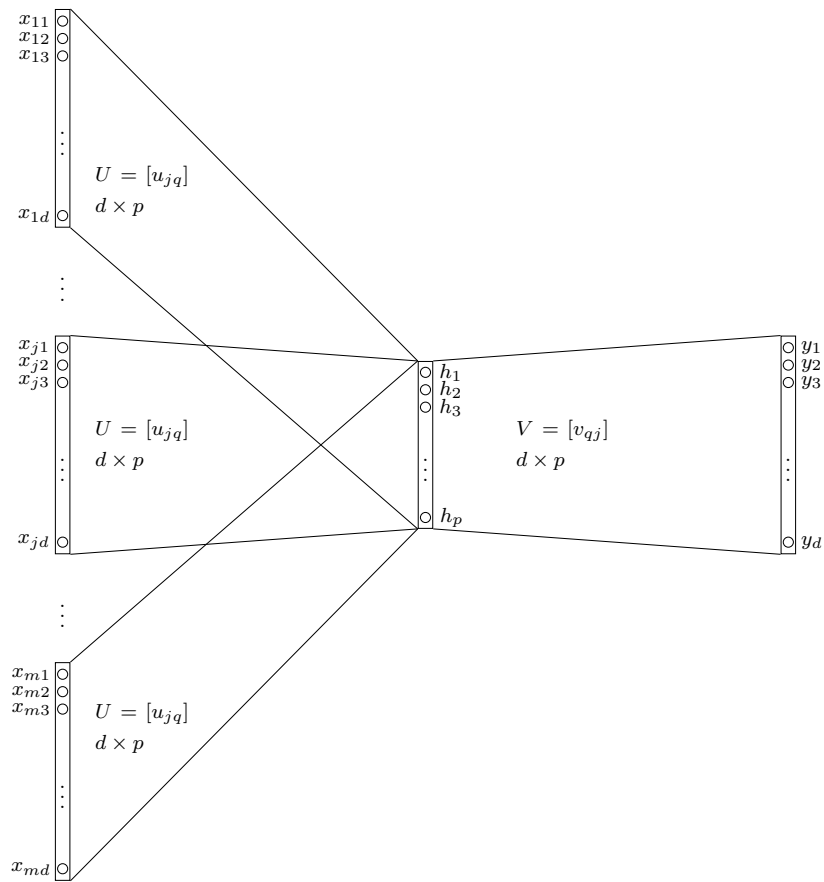
Word2vec có 2 biến thể: dự đoán từ dựa trên các từ xung quanh - Continuous Bag Of Words (CBOW) và dự đoán các từ xung quanh skip-gram.

Trong mô hình CBOW, các tập huấn luyện là các cặp từ có liên quan đến nhau, khi một tập các từ được nhập vào và một từ sẽ được đoán ra. Các từ nhập vào bao gồm  $2 \cdot t$  từ tương ứng với  $t$  từ đứng trước và  $t$  từ phía sau từ đó. Từ đó, tổng số các từ nhập vào sẽ là  $m = 2 \cdot t$  từ. Để đơn giản, các từ đầu vào sẽ được đánh là  $w_1, w_2, \dots, w_m$  và từ được dự đoán là  $w$ . Từ  $w$  có thể có  $d$  giá trị khả thi với  $d$  là tổng số từ trong toàn tập huấn luyện. Mục đích của neural embedding là tính xác suất  $P(w|w_1, w_2, \dots, w_d)$  với  $d$  là số từ trên toàn tập huấn luyện và tối ưu hóa xác suất đó trên toàn tập dữ liệu.

Kiến trúc mỗi tầng được mô tả như sau: [9]

- Lớp đầu vào (*input layer*): Kiến trúc tổng quát được mô tả ở hình trên. Đầu vào có  $m$  từ tương ứng với  $m$  vector, mỗi vector có  $d$  chiều tương ứng với  $d$  từ trong tập huấn luyện, từ đó ta có tổng cộng  $m \cdot d$  nút đầu vào. Mỗi vector đầu vào sẽ được biểu diễn dưới dạng mã hoá one-hot. Chỉ một trong số  $d$  của vector có giá trị là 1 và còn lại sẽ có giá trị là 0.
- Lớp ẩn (*hidden layer*): Lớp ẩn chứa  $p$  thành phần với  $p$  là số chiều của Word2vec. Đầu ra của lớp ẩn là  $h_1, h_2, \dots, h_p$ . Lớp đầu vào được kết nối tới lớp ẩn bởi ma trận trọng số  $U$  kích thước  $d \cdot p$  như đã minh họa ở hình

Mỗi vector  $\mathbf{u}_j = [u_{j1}, u_{j2}, \dots, u_{jp}]$  là một nhúng  $p$  chiều cho từ thứ  $j$  trong toàn bộ từ đầu vào. vector  $\mathbf{h} = [h_1, h_2, \dots, h_p]$  cung cấp một nhúng cho đầu vào. Từ đó đầu ra



**Hình 2.38:** Cách hoạt động của CBOW [9]

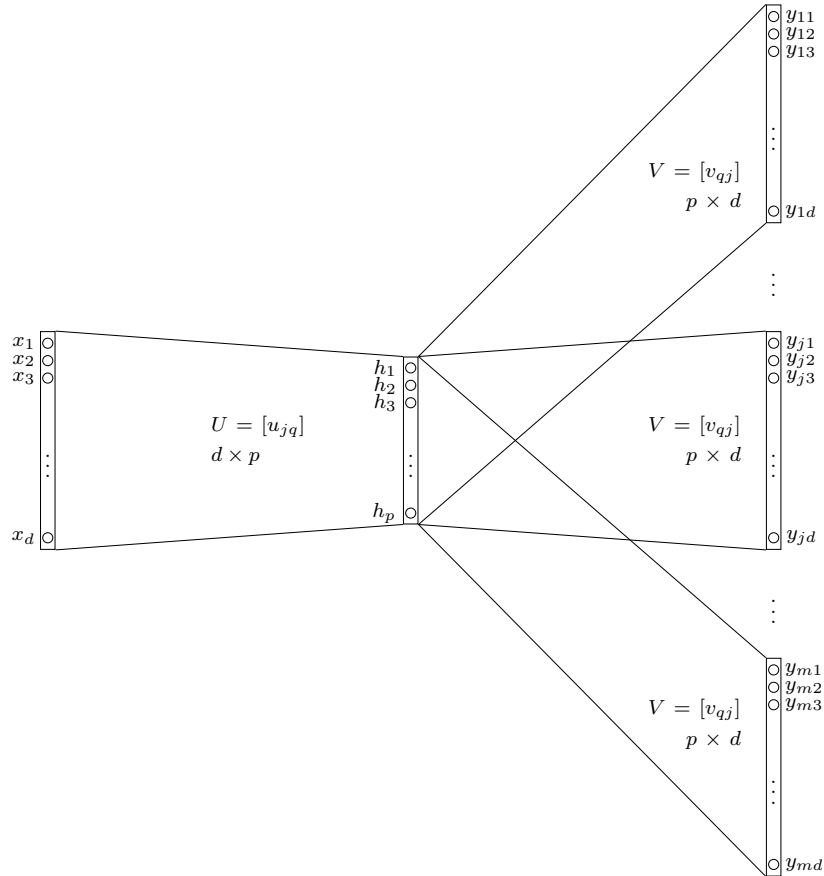
của lớp ẩn là trung bình các nhúng các từ nhập vào và được thể hiện dưới công thức:

$$\mathbf{h} = \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^d \mathbf{u}_j \cdot x_{ij} \quad (2.76)$$

Và kết quả  $\mathbf{h}$  được dùng để dự đoán kết quả đầu ra là một trong  $d$  từ trong tập từ huấn luyện. Trọng số của đầu ra là ma trận  $V = [v_{jq}]$  kích thước  $d \times p$ . Với hàng thứ  $j$  trong ma trận được biểu thị dưới dạng  $\mathbf{v}_j$ . Nhưng do ma trận kết nối tầng ẩn với đầu ra (*hidden-to-output*) là một ma trận dạng  $p \times d$ , nó sẽ được biểu diễn dưới dạng  $V^T$ . kết quả của việc nhân  $\mathbf{h}$  với ma trận  $V^T$  tạo ra một vector  $[\mathbf{h} \cdot \mathbf{v}_1, \dots, \mathbf{h} \cdot \mathbf{v}_d]$ . Kết quả cho ra là kết quả của hàm softmax để tính xác suất mỗi nút của  $y_1, \dots, y_d$ .

$$\hat{y}_j = P(y_j = 1 | w_1 \dots w_m) = \frac{\exp(\mathbf{h} \mathbf{v}_j)}{\sum_{k=1}^d \exp(\mathbf{h} \mathbf{v}_k)} \quad (2.77)$$

Trong mô hình Skip-gram, từ đầu vào được dùng để dự đoán  $m$  từ trước và sau. Từ đó chúng ta có 1 input và  $m$  output.



**Hình 2.39:** Cách hoạt động của skip-gram

Mô hình Skip-gram sử dụng một từ đầu vào  $w$  và đầu ra là  $m$  từ xung quanh được đánh là  $w_1, \dots, w_m$ . Do đó mục tiêu của skip-gram là tìm ra  $P(w_1, \dots, w_m | w)$  khác

với  $P(w|w_1, \dots, w_m)$  của mô hình CBOW. Giống như trong CBOW, skip-gram cũng sử dụng mã hoá one-hot cho đầu vào và đầu ra của mô hình. Sau khi mã hoá chúng ta sẽ có vector  $d$  chiều  $[x_1, \dots, x_d]$  tương ứng với  $d$  giá trị khả thi cho một giá trị đầu vào. Tương tự như vậy, có  $m$  vector đầu ra. Lớp ẩn có  $p$  thành phần và có kết quả  $[h_1 \dots h_p]$ . Lớp đầu vào kết nối với lớp ẩn bởi một ma trận  $U$  kích thước  $d \times p$ .

Lớp ẩn được tính bởi ma trận  $U = [u_{jq}]$  với kích thước  $d \times p$ :

$$\mathbf{h} = \sum_{j=1}^d \mathbf{u}_j \cdot x_j \quad (2.78)$$

Lớp ẩn kết nối tới lớp đầu ra bởi một ma trận  $V = [v_{jq}]$  kích thước  $d \times p$ . Nhưng do ma trận kết nối từ lớp ẩn tới lớp đầu ra là dạng  $p \times d$  nên ở đây sử dụng ma trận  $V^T$ .

$$\hat{y}_{ij} = P(y_{ij} = 1|w) = \frac{\exp(\mathbf{h} \cdot \mathbf{v}_j)}{\sum_{k=1}^d \exp(\mathbf{h} \cdot \mathbf{v}_k)} \quad \forall i \in \{1 \dots m\} \quad (2.79)$$

### 2.3.2.5 Mô hình GloVe

Ở GloVe, chúng ta có thể trích xuất được mối quan hệ giữa các từ bằng cách sử dụng ma trận tần suất đồng thời (*co-occurrence matrix*). Với một câu có  $U$  từ thì ma trận tần suất đồng thời  $X$  sẽ có dạng  $U \cdot X$ . Trong ma trận này hàng thứ  $i$  và cột thứ  $j$  sẽ biểu diễn giá trị  $X_{ij}$  thể hiện sự cùng xuất hiện của hai từ tại vị trí cột  $i$  và hàng  $j$ .

Ví dụ chúng ta có 2 câu: “Data is next oil” và “Data is future”. Dựa trên 2 câu này chúng ta có thể tạo nên một ma trận tần suất đồng thời.

**Bảng 2.4:** Ma trận tần suất đồng thời [6]

	Data	is	next	oil	future
Data	0	2	0	0	0
is	2	0	1	0	1
next	0	1	0	0	0
oil	0	1	0	1	0
future	0	0	1	0	0

Xác suất  $P(j|i) = X_{ij}/X_i$  là xác suất mà từ tại hàng  $j$  xuất hiện đồng thời với cột  $i$ . Như có thể thấy, xác suất để từ “Data” xuất hiện khi có từ “is” là  $P(\text{Data}|\text{is}) = 2/4$ . Với một ví dụ khác

Với bảng trên, chúng ta có thể thấy tỉ lệ chất rắn (*solid*) xuất hiện với băng đá (*ice*) cao hơn hẳn tỉ lệ chất khí (*gas*) với băng đá (*ice*). Như vậy, khi các từ có liên quan đến nhau, tỉ lệ đồng xuất hiện sẽ cao, ngược lại khi không liên quan, tỉ lệ đồng xuất hiện của



**Bảng 2.5:** Xác suất các từ cùng xuất hiện [6]

	k			
	solid	gas	water	fashion
$P(k ice)$	$1.9 \times 10^{-4}$	$6.6 \times 10^{-5}$	$3.0 \times 10^{-3}$	$1.7 \times 10^{-5}$
$P(k steam)$	$2.2 \times 10^{-5}$	$7.8 \times 10^{-4}$	$2.2 \times 10^{-3}$	$1.8 \times 10^{-5}$
$P(k ice)/P(k steam)$	8.9	$8.5 \times 10^{-2}$	1.36	0.96

các từ sẽ thấp. Nhưng khi một từ cùng liên quan tới nhiều từ, ví dụ nước (*water*) đều liên quan tới băng đá (*ice*) và hơi nước (*steam*), do vậy trong những trường hợp này tỉ lệ nhận được sẽ khá cao. Để xử lý vấn đề này, chúng ta sử dụng  $P_{ik}/P_{jk}$ .

Tuy nhiên khi ứng dụng ma trận này trực tiếp, chúng ta sẽ gặp vấn đề nếu dữ liệu có hàng triệu chiều. Do đó, GloVe sử dụng ma trận tần suất đồng thời cùng với Word2vec để xử lý vấn đề trên.

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \quad (2.80)$$

Ở công thức trên có 3 vector từ  $i, j, k$  với  $k$  là vector của các từ được nhập vào. Và hàm  $F$  tính xác suất các từ cùng xuất hiện với nhau.

Ở đây chúng ta có một vấn đề, đó là ở phương trình trên, phía bên trái chứa các vector trong khi số bên phải là một con số tỉ lệ. Do đó chúng ta cần chuyển đổi các vector thành các đại lượng vô hướng.

$$F((w_i - w_j)^\top, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \quad (2.81)$$

Để chuyển đổi vector thành các đại lượng vô hướng, chúng ta thường sử dụng điểm tích của hai vector. Tuy nhiên ở đây chúng ta có 3 vector, do đó chúng ta lấy hiệu của hai vector  $i$  và  $j$ . Từ phương trình 2.81, với  $F(w_i^\top \tilde{w}_k) = P_{ik} = X_{ik}/X_i$ , ta có

$$F((w_i - w_j)^\top, \tilde{w}_k) = \frac{F(w_i^\top \tilde{w}_k)}{F(w_j^\top \tilde{w}_k)} \quad (2.82)$$

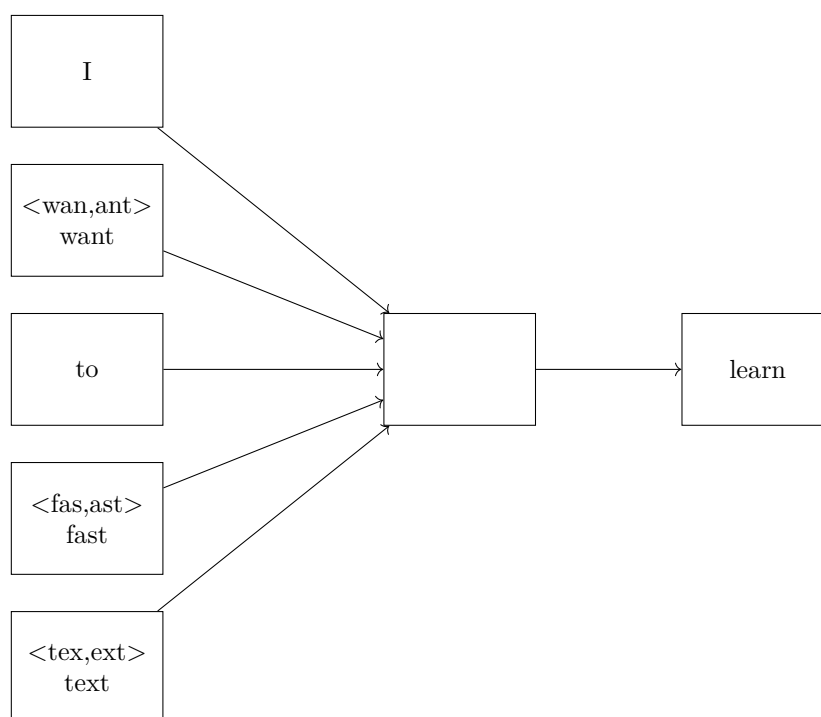
### 2.3.2.6 Mô hình Fasttext

Các phương pháp như Word2vec và GloVe tạo ra các vector khác nhau đại diện cho từ trong kho các từ huấn luyện. Tuy nhiên các phương pháp này lại không thể tìm được sự liên kết giữa các từ dựa trên sự giống nhau giữa các từ. Có nhiều ngôn ngữ mà các từ ngữ đồng nghĩa được biến đổi từ một từ ban đầu, từ đó chúng ta có thể cải thiện được mô hình vector của những ngôn ngữ đó bằng thông tin ở mức kí tự.

Fasttext có thể được coi là một sự mở rộng của Word2vec, tuy nhiên khác với Word2vec hoạt động ở mức độ các từ riêng biệt, Fasttext hoạt động ở mức n-gram kí tự đối với mỗi từ. Ví dụ chúng ta có từ “apple” và  $n = 3$  thì n-gram của từ này là: <ap, app, ppl, ple, le> và <apple>. Và vector đại diện cho từ “apple” có thể được tính bằng tổng các vector n-gram của chính nó.

Giống với Word2vec, Fasttext cũng hoạt động dựa trên 2 phương pháp:

- Continuous Bag Of Words (CBOW): Cách thức hoạt động tương tự như Word2vec, nhưng với mỗi từ được tính bằng n-gram. Ví dụ chúng ta có câu: “I want to learn fast text”, các từ “I”, “want”, “to”, “fast”, “text” được cung cấp để mô hình dự đoán từ “learn” (hình 2.40).
- Skip-gram: Với ví dụ tương tự phía trên, chúng ta áp dụng skip-gram sẽ như hình 2.41.

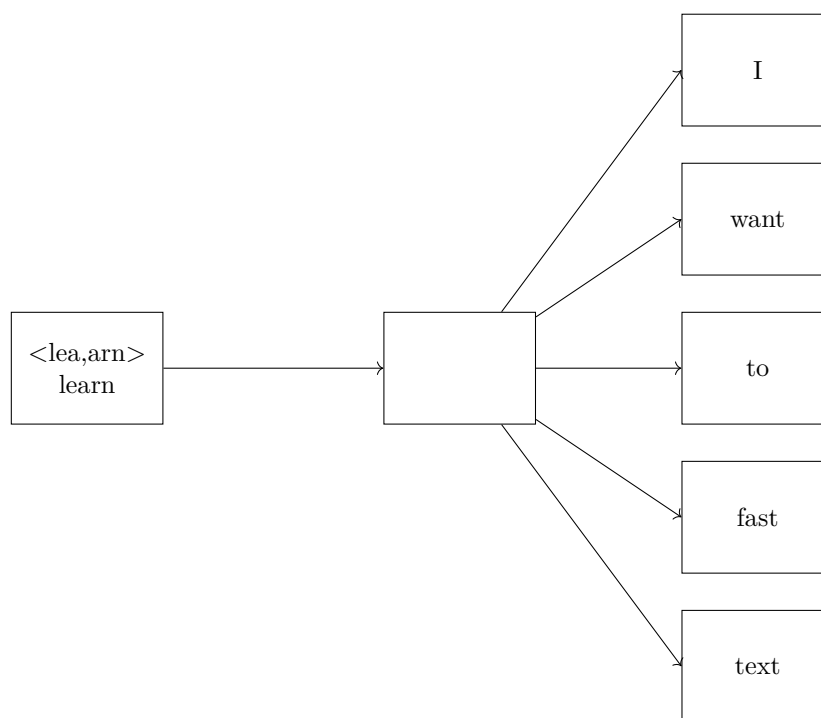


**Hình 2.40:** Mô hình fasttext đối với CBOW [15]

### 2.3.3 Ngữ cảnh (*Contextual*) và vai trò trong NLP

Bản chất của ngôn ngữ là âm thanh được phát ra để diễn giải dòng suy nghĩ của con người. Trong giao tiếp, các từ thường không đứng độc lập mà chúng sẽ đi kèm với các từ khác để liên kết mạch lạc thành một câu. Hiệu quả biểu thị nội dung và truyền đạt ý nghĩa sẽ lớn hơn so với từng từ đứng độc lập.

Ngữ cảnh trong câu có một sự ảnh hưởng rất lớn trong việc giải thích ý nghĩa của từ. Hiểu được vai trò mấu chốt đó, các thuật toán NLP hiện đại đều cố gắng đưa ngữ cảnh vào mô hình nhằm tạo ra sự đột phá và cải tiến.



**Hình 2.41:** Mô hình fasttext đối với skip-gram [15]

Phân cấp mức độ phát triển của các phương pháp embedding từ trong NLP có thể bao gồm các nhóm:

- **Non-context (không bối cảnh):** Là các thuật toán không tồn tại bối cảnh trong biểu diễn từ. Đó là các thuật toán NLP đời đầu như ‘word2vec, GLoVe, fasttext’ đã trình bày ở trên. Chúng ta chỉ có duy nhất một biểu diễn vector cho mỗi một từ mà không thay đổi theo bối cảnh. Ví dụ:

Câu A: Đơn vị tiền tệ của Việt Nam là [đồng].

Câu B: Vợ [đồng] ý với ý kiến của chồng là tăng thêm mỗi tháng 500k tiền tiêu vặt. Thì từ “đồng” sẽ mang 2 ý nghĩa khác nhau nên phải có hai biểu diễn từ riêng biệt. Các thuật toán non-context đã không đáp ứng được sự đa dạng về ngữ nghĩa của từ trong NLP.

- **Uni-directional (một chiều):** Là các thuật toán đã bắt đầu xuất hiện bối cảnh của từ. Các phương pháp nhúng từ dựa trên RNN là những phương pháp nhúng từ một chiều. Các kết quả biểu diễn từ đã có bối cảnh nhưng chỉ được giải thích bởi một chiều từ trái qua phải hoặc từ phải qua trái. Ví dụ:

Câu C: Hôm nay tôi mang 200 tỷ [gửi] ở ngân hàng.

Câu D: Hôm nay tôi mang 200 tỷ [gửi] ....

Như vậy vector biểu diễn của từ “gửi” được xác định thông qua các từ liền trước với nó. Nếu chỉ dựa vào các từ liền trước “Hôm nay tôi mang 200 tỷ” thì ta có thể nghĩ từ phù hợp ở vị trí hiện tại là cho “vay”, “mua”, “thanh toán”, ...

Ví dụ đơn giản trên đã cho thấy các thuật toán biểu diễn từ có bối cảnh tuân theo theo một chiều sẽ gặp hạn chế lớn trong biểu diễn từ hơn so với biểu diễn 2 chiều. Mô hình ELMo[x] là một ví dụ cho phương pháp một chiều. Mặc dù ELMo có kiến trúc dựa trên một mạng BiLSTM xem xét bối cảnh theo hai chiều từ trái sang phải và từ phải sang trái nhưng những chiều này là độc lập nhau nên ta coi như đó là biểu diễn một chiều.

Thuật toán ELMo đã cải tiến hơn so với word2vec và fasttext đó là tạo ra nghĩa của từ theo bối cảnh. Trong ví dụ về từ đồng thì ở mỗi câu A và B chúng ta sẽ có một biểu diễn từ khác biệt.

- **Bi-directional (hai chiều):** Ngữ nghĩa của một từ không chỉ được biểu diễn bởi những từ liền trước mà còn được giải thích bởi toàn bộ các từ xung quanh. Luồng giải thích tuân theo đồng thời từ trái qua phải và từ phải qua trái cùng một lúc. Đại diện cho các phép biểu diễn từ này là những mô hình sử dụng kỹ thuật transformer mà chúng ta sẽ tìm hiểu bên dưới. Gần đây, những thuật toán NLP theo trường phái bidirectional như BERT, ULMFit, OpenAI GPT đã đạt được những kết quả State-Of-The-Art trên hầu hết các tác vụ của GLUE benchmark.

### 2.3.4 Mô hình Transformer

#### 2.3.4.1 Giới thiệu

Mô hình Sequence-to-Sequence (Seq2Seq) nhận đầu vào là một chuỗi và trả lại đầu ra cũng là một chuỗi. Ví dụ bài toán QA, đầu vào là câu hỏi “how are you?” và đầu ra là câu trả lời “I am good”. Phương pháp truyền thống sử dụng RNN cho cả encoder (phần mã hóa đầu vào) và decoder (phần giải mã đầu vào và trả đầu ra tương ứng).

Nhưng mô hình này vẫn có nhiều điểm yếu như đã trình bày, điểm yếu đầu tiên là tốc độ huấn luyện rất chậm, do bản chất tuần hoàn của mạng, việc tính toán gradient cho các bước thời gian trước đó cần thực hiện theo trình tự, dẫn đến việc sử dụng CPU thay vì GPU, vốn có khả năng tính toán song song hiệu quả hơn. Việc sử dụng Truncated Backpropagation để cải thiện tốc độ huấn luyện vẫn còn hạn chế. Điểm yếu thứ hai là nó xử lý không tốt với những câu dài do hiện tượng Gradient Vanishing/Exploding như đã đề cập.

Ra đời năm 1997, LSTM và theo sau đó là GRU (2000) có vẻ đã giải quyết phần nào vấn đề Gradient Vanishing, nhưng cả hai kỹ thuật này lại phức tạp hơn RNN rất nhiều, và hiển nhiên nó cũng train chậm hơn RNN đáng kể.

Vậy có cách nào tận dụng khả năng tính toán song song của GPU để tăng tốc độ học hỏi cho các mô hình ngôn ngữ, đồng thời khắc phục điểm yếu xử lý câu dài không? Transformers chính là câu trả lời. Ý tưởng chính của phần này được lấy từ bài báo Attention is all you need[xxx] xuất bản vào năm 2017 của tác giả Ashish Vaswani tại hội nghị quốc tế hằng năm về hệ thống xử lý thông tin mạng thần kinh nhân tạo (NeurIPS).

Tính đến năm 2024, bài báo đã được trích dẫn 100.000 lần, trở thành một trong những bài báo có ảnh hưởng nhất trong lĩnh vực học máy. Bài báo giới thiệu kiến trúc Transformer, một mô hình học máy mới dựa hoàn toàn trên cơ chế chú ý (attention). Transformer đã thay đổi cách thức xây dựng các mô hình học máy cho các nhiệm vụ xử lý ngôn ngữ tự nhiên, đặc biệt là dịch máy.

Điều đặc biệt về Transformer so với mô hình RNN là nó không yêu cầu xử lý các phần tử trong chuỗi một cách tuần tự. Trong khi RNN phải xử lý từng phần tử theo thứ tự từ đầu đến cuối, Transformer có thể xử lý toàn bộ câu ngôn ngữ tự nhiên cùng một lúc. Điều này cho phép Transformer tận dụng tối đa khả năng tính toán song song của GPU và giảm thời gian xử lý đáng kể. Thay vì phải chờ đợi kết quả từ các phần tử trước đó trong chuỗi, Transformer có thể đồng thời xem xét và ánh xạ mối quan hệ giữa tất cả các từ trong câu. Điều này giúp mô hình hiểu được mối quan hệ toàn diện trong câu và tạo ra các biểu diễn ngữ nghĩa phong phú hơn.

### 2.3.4.2 Kiến trúc Transformer

Kiến trúc của Transformer gồm 2 phần là Encoder bên trái và Decoder bên phải.

**Encoder:** là tổng hợp xếp chồng lên nhau của 6 layers xác định. Mỗi layer bao gồm 2 layer con (sub-layer) trong nó. Sub-layer đầu tiên là multi-head self-attention mà lát nữa chúng ta sẽ tìm hiểu. Layer thứ 2 đơn thuần chỉ là các fully-connected feed-forward layer. Một lưu ý là chúng ta sẽ sử dụng một kết nối residual ở mỗi sub-layer ngay sau layer normalization. Kiến trúc này có ý tưởng tương tự như mạng resnet trong CNN. Đầu ra của mỗi sub-layer là  $LayerNorm(x + Sublayer(x))$  có số chiều là 512 theo như bài viết.

**Decoder:** Decoder cũng là tổng hợp xếp chồng của 6 layers. Kiến trúc tương tự như các sub-layer ở Encoder ngoại trừ thêm 1 sub-layer thể hiện phân phối attention ở vị trí đầu tiên. Layer này không gì khác so với multi-head self-attention layer ngoại trừ được điều chỉnh để không đưa các từ trong tương lai vào attention. Tại bước thứ  $i$  của decoder chúng ta chỉ biết được các từ ở vị trí nhỏ hơn  $i$  nên việc điều chỉnh đảm bảo attention chỉ áp dụng cho những từ nhỏ hơn vị trí thứ  $i$ . Cơ chế residual cũng được áp dụng tương tự như trong Encoder.

Ngoài ra luôn có một bước cộng thêm Positional Encoding vào các đầu vào của encoder và decoder nhằm đưa thêm yếu tố thời gian vào mô hình làm tăng độ chuẩn xác. Đây chỉ đơn thuần là phép cộng vector mã hóa vị trí của từ trong câu với vector biểu diễn từ. Chúng ta có thể mã hóa dưới dạng  $[0, 1]$  vector vị trí hoặc sử dụng hàm  $\sin, \cos$  như trong bài báo.

### 2.3.4.3 Cơ chế Attention

#### 1 Scaled Dot-Product Attention:

Đây chính là một cơ chế self-attention khi mỗi từ có thể điều chỉnh trọng số của nó

cho các từ khác trong câu sao cho từ ở vị trí càng gần nó nhất thì trọng số càng lớn và càng xa thì càng nhỏ dần. Sau bước nhúng từ (đi qua embedding layer) ta có đầu vào của encoder và decoder là ma trận  $\mathbf{X}$  kích thước  $m \times n$ ,  $m, n$  lần lượt là độ dài câu và số chiều của một vector nhúng từ.

Sau đó, ma trận  $\mathbf{X}$  sẽ được nhân với 3 ma trận trọng số tương ứng  $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$  chính là những hệ số mà model cần huấn luyện, ta thu được ma trận  $\mathbf{Q}, \mathbf{K}, \mathbf{V}$  (Query, Key và Value). Ma trận Query và Key có tác dụng tính toán ra phân phối score cho các cặp từ. Ma trận Value sẽ dựa trên phân phối score để tính ra vector phân phối xác suất đầu ra. Ba ma trận  $\mathbf{Q}, \mathbf{K}, \mathbf{V}$  chính là 3 mũi tên đầu vào của các Multi-head Attention (bản chất là Self-Attention) trong kiến trúc tổng quát hình x.

Như đã đề cập, Đầu vào để tính attention sẽ bao gồm ma trận  $\mathbf{Q}$  (mỗi dòng của nó là một vector query đại diện cho các từ trong đầu vào), ma trận  $\mathbf{K}$  (tương tự như ma trận  $\mathbf{Q}$ , mỗi dòng là vector key đại diện cho các từ trong đầu vào). Hai ma trận  $\mathbf{Q}, \mathbf{K}$  được sử dụng để tính attention mà các từ trong câu trả về cho 1 từ cụ thể trong câu. Attention vector sẽ được tính dựa trên trung bình có trọng số của các vector value trong ma trận  $\mathbf{V}$  với trọng số attention (được tính từ  $\mathbf{Q}, \mathbf{K}$ ).

Phương trình Attention như sau:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V} \quad (\text{xxx})$$

Việc chia cho  $d_k$  là số dimension của vector key nhằm mục đích tránh tràn luồng nếu số mũ là quá lớn. Trong thực hành chúng ta tính toán hàm attention trên toàn bộ tập các câu truy vấn một cách đồng thời được đóng gói thông qua ma trận  $\mathbf{Q}$ . Keys và Values cũng được đóng gói cùng nhau thông qua ma trận  $\mathbf{K}, \mathbf{V}$ .

## 2 Multi-head Attention:

Như vậy sau quá trình Scale dot production chúng ta sẽ thu được 1 ma trận attention. Các tham số mà model cần tinh chỉnh chính là các ma trận  $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$ . Mỗi quá trình như vậy được gọi là một head của attention. Khi lặp lại quá trình này nhiều lần (trong bài báo là ba heads) ta sẽ thu được quá trình Multi-head Attention.

Sau khi thu được ba ma trận attention ở đầu ra chúng ta sẽ concatenate các matrix này theo các cột để thu được ma trận tổng hợp multi-head matrix có chiều cao trùng với chiều cao của ma trận đầu vào.

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h) \mathbf{W}^O$$

với  $\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$

Cuối cùng, để trả đầu ra có cùng kích thước với ma trận ban đầu chúng ta chỉ cần nhân với ma trận  $\mathbf{W}_0$  có chiều rộng bằng với chiều rộng của ma trận đầu vào.

### 2.3.5 Tiếp cận nông và học sâu trong ứng dụng pre-training NLP

#### 2.3.5.1 Tiếp cận nông (Shallow approach)

Trong xử lý ảnh chúng ta đều biết tới những pretrained models nổi tiếng trên bộ dữ liệu Imagenet với 1000 classes. Nhờ số lượng classes lớn nên hầu hết các nhãn trong phân loại ảnh thông thường đều xuất hiện trong Imagenet và chúng ta có thể học chuyển giao lại các tác vụ xử lý ảnh rất nhanh và tiện lợi. Chúng ta cũng kỳ vọng NLP có một tập hợp các pretrained models như vậy, tri thức từ model được huấn luyện trên các nguồn tài nguyên văn bản không nhãn (unlabeled text) rất dồi dào và sẵn có.

Tuy nhiên trong NLP việc học chuyển giao là không hề đơn giản như Computer Vision. Các kiến trúc mạng deep CNN của Computer Vision cho phép học chuyển giao trên đồng thời cả low-level và high-level features thông qua việc tận dụng lại các tham số từ những layers của mô hình pretrained.

Nhưng trong NLP, các thuật toán cũ hơn như GLoVe, word2vec, fasttext chỉ cho phép sử dụng các biểu diễn vector nhúng của từ là các low-level features như là đầu vào cho layer đầu tiên của mô hình. Các layers còn lại giúp tạo ra high-level features thì dường như được huấn luyện lại từ đầu.

Như vậy chúng ta chỉ chuyển giao được các đặc trưng ở mức độ rất nông nên phương pháp này còn được gọi là tiếp cận nông (shallow approach). Việc tiếp cận với các layers sâu hơn là không thể. Điều này tạo ra một hạn chế rất lớn đối với NLP so với Computer Vision trong việc học chuyển giao. Cách tiếp cận nông trong học chuyển giao còn được xem như là feature-based.

Khi áp dụng feature-based, chúng ta sẽ tận dụng lại các biểu diễn từ được huấn luyện trước trên những kiến trúc mô hình cố định và những bộ văn bản có kích thước rất lớn để nâng cao khả năng biểu diễn từ trong không gian đa chiều. Một số pretrained feature-based bạn có thể áp dụng trong tiếng anh đã được huấn luyện sẵn đó là GloVe, word2vec, fasttext, ELMo.

#### 2.3.5.2 Học sâu (Deep learning)

Ngoài ra, Một trong những thách thức lớn nhất của NLP là thiếu hụt dữ liệu huấn luyện. Mặc dù tổng thể có một lượng dữ liệu văn bản khổng lồ, nhưng để tạo ra các bộ dữ liệu cho các nhiệm vụ cụ thể, chúng ta cần phân chia chúng thành nhiều lĩnh vực rất đa dạng. Điều này dẫn đến việc chỉ có vài nghìn hoặc vài trăm nghìn mẫu huấn luyện được dán nhãn thủ công. Nếu muốn đạt được những cải tiến đáng kể, các mô hình NLP dựa trên học sâu đòi hỏi lượng dữ liệu lớn hơn nhiều - hàng triệu hoặc hàng tỷ mẫu huấn luyện được dán nhãn.

Để giúp bù đắp khoảng trống về dữ liệu này, các nhà nghiên cứu đã phát triển các kỹ

thuật khác nhau để huấn luyện sử dụng khối lượng văn bản khổng lồ không dán nhãn trên web (được gọi là pre-training). Các mô hình này được tiền huấn luyện này sau đó có thể được tinh chỉnh (fine-tuned) trên các bộ dữ liệu nhỏ hơn cho các nhiệm vụ cụ thể, chẳng hạn như giải quyết các vấn đề như trả lời câu hỏi, phân tích cảm xúc. Cách tiếp cận này đã mang lại những cải thiện đáng kể so với việc huấn luyện từ đầu trên các bộ dữ liệu nhỏ hơn cho các nhiệm vụ cụ thể.

Các mô hình NLP đột phá trong hai năm trở lại đây như BERT, ELMo, ULMFit, OpenAI GPT đã cho phép việc chuyển giao layers trong NLP khả thi hơn. Chúng ta không chỉ học chuyển giao được các đặc trưng mà còn chuyển giao được kiến trúc của mô hình nhờ số lượng layers nhiều hơn, chiều sâu của mô hình sâu hơn trước đó.

Các kiến trúc mới phân cấp theo level có khả năng chuyển giao được những cấp độ khác nhau của đặc trưng từ low-level tới high-level. Trong khi học nông chỉ chuyển giao được low-level tại layer đầu tiên. Tất nhiên low-level cũng đóng vai trò quan trọng trong các tác vụ NLP. Nhưng high-level là những đặc trưng có ý nghĩa hơn vì đó là những đặc trưng đã được tinh luyện.

Người ta kỳ vọng rằng ULMFit, OpenAI GPT, BERT sẽ là những mô hình pretrained giúp tiến gần hơn tới việc xây dựng một lớp các pretrained models ImageNet for NLP. Khi học chuyển giao theo phương pháp học sâu chúng ta sẽ tận dụng lại kiến trúc từ mô hình pretrained và bổ sung một số layers phía sau để phù hợp với nhiệm vụ huấn luyện. Các tham số của các layers gốc sẽ được fine-tuning lại. Chỉ một số ít các tham số ở layers bổ sung được huấn luyện lại từ đầu.

### 2.3.6 Mô hình BERT (*Bidirectional Encoder Representations from Transformers*)

#### 2.3.6.1 Giới thiệu BERT

Vào cuối năm 2018, các nhà nghiên cứu tại Google AI Language đã công bố một kỹ thuật mới được gọi là BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) - một bước đột phá lớn đã gây tiếng vang trong cộng đồng Học sâu bởi hiệu suất đáng kinh ngạc của nó. Trong bài báo BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[xxx] các tác giả đã nêu ra những cải tiến của model BERT trong các tác vụ:

- Tăng GLUE score (General Language Understanding Evaluation score), một chỉ số tổng quát đánh giá mức độ hiểu ngôn ngữ lên 80.5%.
- Tăng accuracy trên bộ dữ liệu MultiNLI đánh giá tác vụ quan hệ văn bản (text entailment) lên 86.7%.
- Tăng accuracy F1 score trên bộ dữ liệu SQuAD v1.1 đánh giá tác vụ question and answering lên 93.2%.

Về cơ bản, nhiệm vụ mà các mô hình ngôn ngữ cố gắng giải quyết là "điền vào chỗ trống" dựa trên ngữ cảnh. Ví dụ, với câu: "Tôi đi chợ và mua một \_\_\_\_ giày." Một mô



hình ngôn ngữ có thể hoàn thành câu bằng cách dự đoán từ điền vào chỗ trống:

- “đôi” với tỉ lệ 80% vì “đôi” là cách đếm thông dụng cho giày.
- “chiếc” với tỉ lệ 20% vì đôi khi người ta cũng một chiếc giày.

Trước khi BERT được công bố, những mô hình ngôn ngữ sẽ xem xét chuỗi văn bản này trong quá trình huấn luyện theo hướng từ trái sang phải hoặc kết hợp cả hai hướng. Cách tiếp cận đơn hướng này hoạt động tốt cho việc tạo câu, chúng ta có thể dự đoán từ tiếp theo, thêm nó vào chuỗi, sau đó dự đoán từ tiếp theo sau đó cho đến khi có một câu hoàn chỉnh.

Tuy nhiên, với sự xuất hiện của BERT, mô hình ngôn ngữ giờ đây được huấn luyện theo hai hướng, đã làm thay đổi cách tiếp cận ngữ cảnh của từ. Điều này có nghĩa là bây giờ chúng ta có thể có được hiểu biết sâu sắc hơn về bối cảnh so với các mô hình ngôn ngữ một hướng.

Thay vì dự đoán từ tiếp theo trong chuỗi, BERT sử dụng một kỹ thuật mới gọi là Masked ML (MLM). Kỹ thuật này hoạt động bằng cách ngẫu nhiên che khuất một số từ trong câu và sau đó BERT cố gắng dự đoán các từ đó. Che khuất ở đây có nghĩa là mô hình sẽ xem xét cả hai hướng của câu, sử dụng toàn bộ ngữ cảnh của câu, bao gồm cả các từ trước và sau từ bị che khuất, để dự đoán từ đó. Không giống như các mô hình ngôn ngữ trước đây, BERT có thể tính đến cả token trước và token sau cùng một lúc. Các mô hình LSTM kết hợp trái-sang-phải và phải-sang-trái hiện nay không thể làm được điều này mặc dù trên thực tế, chính xác hơn chúng ta có thể nói rằng đó là huấn luyện không hướng (non-directional).

### 2.3.6.2 Pre-training BERT

BERT được huấn luyện dựa trên Transformer, mục tiêu là tạo mô hình biểu diễn ngôn ngữ, vì vậy BERT chỉ cần phần Encoder. Đầu vào cho bộ mã hóa của BERT là một chuỗi các token, được chuyển đổi thành các vector và sau đó được xử lý trong mạng nơ-ron. Tuy nhiên, trước khi quá trình xử lý bắt đầu, BERT cần một số dữ liệu bổ sung vào input:

- **Nhúng từ (token embedding):** Ký hiệu [CLS] được thêm vào các token từ đầu vào ở đầu câu đầu tiên và ký hiệu [SEP] được chèn vào cuối mỗi câu.
- **Nhúng câu (segment embedding):** Thêm một ký hiệu chỉ thị Câu A hoặc Câu B cho mỗi token. Điều này cho phép bộ mã hóa phân biệt giữa các câu.
- **Nhúng vị trí (position embedding):** Thêm một vector nhúng vị trí cho mỗi token trong chuỗi. Điều này cho phép mô hình biết vị trí của mỗi token trong chuỗi.

Về bản chất, Transformer sẽ ánh xạ chuỗi ký tự đầu vào thành chuỗi ký tự đầu ra, do đó đầu ra cũng là một chuỗi các vector tương ứng giữa các token đầu vào và đầu ra ở cùng chỉ số. Như đã đề cập trước đó, BERT sẽ không dự đoán từ tiếp theo trong câu. Quá trình huấn luyện sử dụng hai chiến lược sau:

1. **Masked ML (MLM):** Masked ML là một tác vụ cho phép chúng ta fine-tuning lại

các biểu diễn từ trên các bộ dữ liệu unsupervised-text bất kỳ. Chúng ta có thể áp dụng Masked ML cho những ngôn ngữ khác nhau để tạo ra biểu diễn embedding cho chúng. Các bộ dữ liệu của tiếng anh có kích thước lên tới vài vài trăm tới vài nghìn GB được huấn luyện trên BERT đã tạo ra những kết quả khá ấn tượng.

Theo đó:

- Khoảng 15% các token của câu input được thay thế bởi token [MASK] trước khi truyền vào model đại diện cho những từ bị che dấu (masked). Mô hình sẽ dựa trên các từ không được che (non-masked) xung quanh và đồng thời là bối cảnh của [MASK] để dự báo giá trị gốc của từ được che dấu. Số lượng từ được che dấu được lựa chọn là một số ít (15%) để tỷ lệ bối cảnh chiếm nhiều hơn (85%).
- Bản chất của kiến trúc BERT vẫn là một mô hình seq2seq gồm 2 phase encoder giúp embedding các từ input và decoder giúp tìm ra phân phối xác suất của các từ ở output. Kiến trúc Transformer encoder được giữ lại trong tác vụ Masked ML. Sau khi thực hiện self-attention và feed forward ta sẽ thu được các vector embedding ở output là  $O_1, O_2, \dots, O_5$
- Để tính toán phân phối xác suất cho từ output, chúng ta thêm một Fully connect layer ngay sau Transformer Encoder. Hàm softmax có tác dụng tính toán phân phối xác suất. Số lượng units của fully connected layer phải bằng với kích thước của từ điển.
- Cuối cùng ta thu được vector nhúng của mỗi một từ tại vị trí [MASK] sẽ là embedding vector giảm chiều của vector  $O_i$  sau khi đi qua fully connected layer như mô tả trên hình vẽ bên phải.

Hàm loss function của BERT sẽ bỏ qua mất mát của những từ không bị che và chỉ nhận của những từ bị che dấu. Do đó mô hình sẽ hội tụ lâu hơn nhưng đây là đặc tính bù trừ cho sự gia tăng ý thức về bối cảnh. Việc lựa chọn ngẫu nhiên 15% số lượng các từ bị che dấu cũng tạo ra vô số các kịch bản input cho mô hình huấn luyện nên mô hình sẽ cần phải huấn luyện rất lâu mới học được toàn diện các khả năng.

### 2. Next Sentence Prediction (NSP):

Đây là một bài toán phân loại học có giám sát với 2 nhãn (hay còn gọi là phân loại nhị phân). Input đầu vào của mô hình là một cặp câu (pair-sequence) sao cho 50% câu thứ 2 được lựa chọn là câu tiếp theo của câu thứ nhất và 50% được lựa chọn một cách ngẫu nhiên từ bộ văn bản mà không có mối liên hệ gì với câu thứ nhất. Nhãn của mô hình sẽ tương ứng với IsNext khi cặp câu là liên tiếp hoặc NotNext nếu cặp câu không liên tiếp.

Cũng tương tự như trên, chúng ta cần đánh dấu các vị trí đầu câu thứ nhất bằng token [CLS] và vị trí cuối các câu bằng token [SEP]. Các token này có tác dụng

nhận biết các vị trí bắt đầu và kết thúc của từng câu thứ nhất và thứ hai.

Mô hình sẽ dựa vào các vector embedding của token [CLS] để dự đoán xem cặp câu đó có phải là liên tiếp hay không. Để dự đoán, chúng ta sẽ thêm một fully connected layer với 2 units (tương ứng với 2 nhãn) ngay sau vector embedding của token [CLS]. Hàm softmax sẽ giúp tính toán phân phối xác suất của 2 nhãn. Cuối cùng, mô hình sẽ dự đoán nhãn có xác suất cao nhất là nhãn cuối cùng của cặp câu.

### 2.3.6.3 Tinh chỉnh (fine-tuning) BERT

BERT đã vượt trội so với các phương pháp tối ưu (state-of-the-art) trước đây trên nhiều tác vụ khác nhau trong lĩnh vực hiểu ngôn ngữ tổng quát, chẳng hạn như suy luận ngôn ngữ tự nhiên, phân tích tình cảm và trả lời câu hỏi. Nếu muốn tinh chỉnh BERT cho một nhiệm vụ cụ thể dựa trên tập dữ liệu, việc cần làm là thêm một layer phía sau phần Encoder của BERT.

Ví dụ, giả sử chúng ta đang tạo một ứng dụng trả lời câu hỏi (question answering). Về bản chất, đây là một tác vụ dự đoán - khi nhận được một câu hỏi làm đầu vào, mục tiêu của ứng dụng là xác định câu trả lời đúng có trong một ngữ liệu nào đó. Vì vậy, với một câu hỏi và một đoạn ngữ cảnh, mô hình sẽ dự đoán một token bắt đầu và một token kết thúc trong đoạn văn đó, đoạn văn được lấy từ 2 token này có khả năng là trả lời chính xác.

Ví dụ:

- Input Question: Where do water droplets collide with ice crystals to form precipitation?
- Input Paragraph: ... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. ...
- Output Answer: within a cloud

Tương tự như các tác vụ liên quan đến cặp câu được đề cập ở phần dự đoán câu tiếp theo, ở phần này câu hỏi sẽ trở thành câu đầu tiên và đoạn văn là câu thứ hai trong chuỗi đầu vào. Tuy nhiên, lần này có thêm hai tham số mới được học trong quá trình tinh chỉnh: một vector bắt đầu và một vector kết thúc.

### 2.3.6.4 Những kiến trúc của BERT

Hiện tại có nhiều phiên bản khác nhau của model BERT. Các phiên bản đều dựa trên việc thay đổi kiến trúc của Transformer tập trung ở 3 tham số:  $L$ : số lượng các block sub-layers trong transformer,  $H$ : kích thước của embedding vector (hay còn gọi là hidden size),  $A$ : Số lượng head trong multi-head layer, mỗi một head sẽ thực hiện một self-attention. Tên gọi của 2 kiến trúc bao gồm:

- $BERT_{BASE}$  ( $L=12$ ,  $H=768$ ,  $A=12$ ): Tổng tham số 110 triệu.
- $BERT_{LARGE}$  ( $L=24$ ,  $H=1024$ ,  $A=16$ ): Tổng tham số 340 triệu.

Như vậy ở kiến trúc BERT Large chúng ta tăng gấp đôi số layer, tăng kích thước hidden size của embedding vector gấp 1.33 lần và tăng số lượng head trong multi-head layer gấp

1.33 lần.

## 2.4 XÂY DỰNG MÔ HÌNH PHÁT HIỆN TỪ NGỮ ĐỘC HẠI

### 2.4.1 Môi trường cài đặt và các thư viện sử dụng

#### 2.4.1.1 Môi trường cài đặt

Bài toán được thực hiện hoàn toàn trên hai môi trường tính toán đám mây là Google Colaboratory hay còn gọi là Google Colab, là một sản phẩm từ Google Research, và Kaggle Notebooks được cung cấp bởi Kaggle, một nền tảng cộng đồng trực tuyến cho các nhà khoa học dữ liệu và học máy. Cả hai môi trường này đều miễn phí và cho phép thực thi Python trên nền tảng đám mây, đặc biệt phù hợp với Data analysis, Machine Learning và giáo dục.

Những môi trường này không cần yêu cầu cài đặt hay cấu hình máy tính, mọi thứ có thể chạy thông qua trình duyệt, bạn có thể sử dụng tài nguyên máy tính từ CPU tốc độ cao và cả GPUs và cả TPUs đều được cung cấp cho bạn.

Sử dụng chúng có những lợi ích ưu việt như: sẵn sàng chạy Python ở bất kỳ thiết bị nào có kết nối Internet mà không cần cài đặt, chia sẻ và làm việc nhóm dễ dàng, sử dụng miễn phí GPU cho các dự án về AI. [10]

Đối với bài này thì nhóm sử dụng phần cứng có tên T4, với 2vCPU Xeon 2.00GHz, 12.7GB bộ nhớ trong, 78.2GB bộ nhớ ngoài, dùng GPU Tesla T4 có 15.36 GB bộ nhớ.

#### 2.4.1.2 Thư viện

##### 1. Thư viện *Tensorflow*

TensorFlow là một thư viện mã nguồn mở end-to-end được tạo ra chủ yếu dành cho các ứng dụng Học máy. Nó là một thư viện toán học ký hiệu sử dụng luồng dữ liệu và lập trình có thể phân biệt để thực hiện các nhiệm vụ khác nhau, tập trung vào đào tạo và suy luận các mạng neuron sâu (*deep neural network*). Nó cho phép các nhà phát triển tạo các ứng dụng học máy bằng cách sử dụng các công cụ, thư viện và tài nguyên cộng đồng khác nhau. [7]

TensorFlow cho phép bạn xây dựng biểu đồ và cấu trúc luồng dữ liệu để xác định cách dữ liệu di chuyển qua biểu đồ bằng cách lấy đầu vào là một mảng đa chiều được gọi là tensor. Nó cho phép bạn xây dựng một sơ đồ các hoạt động có thể được thực hiện trên các đầu vào này, đi ở một đầu và đến ở đầu kia là đầu ra. Kiến trúc của Tensorflow hoạt động trong ba phần:

- Xử lý trước dữ liệu
- Xây dựng mô hình
- Đào tạo và ước tính mô hình

##### 2. Thư viện *Keras*

Keras là một API cấp cao được thiết kế cho Python để triển khai mạng nơ-ron để

dàng hơn. Nó được phát triển bởi Google.

Keras có thể chạy trên các thư viện và khung công tác như TensorFlow, Theano, PlaidML, MXNet, CNTK. Chúng đều là những thư viện rất mạnh nhưng cũng khó hiểu để tạo mạng nơ-ron. Mặt khác, Keras rất thân thiện với người mới bắt đầu vì cấu trúc tối thiểu của nó cung cấp cách tạo ra các mô hình học sâu một cách dễ dàng và gọn ghẽ dựa trên TensorFlow hoặc Theano.

Keras đã được TensorFlow thông qua làm API cấp cao chính thức của mình. Khi được nhúng vào TensorFlow, nó cung cấp các mô-đun có sẵn cho tất cả các tính toán mạng nơ-ron và do đó có thể thực hiện học sâu rất nhanh. TensorFlow rất linh hoạt và lợi ích chính là tính toán phân tán. Bạn có thể linh hoạt và có thể kiểm soát ứng dụng của mình, thực hiện ý tưởng của bạn trong thời gian ngắn, sử dụng Keras, trong khi tính toán liên quan đến tensors, đồ thị tính toán,...có thể được tùy chỉnh bằng cách sử dụng Tensorflow Core API.

### 3. *Thư viện WordCloud*

Word cloud (đám mây từ) là một kỹ thuật trực quan hóa dữ liệu trong NLP. Nó là một đám mây chứa rất nhiều từ với các kích thước khác nhau, chúng thể hiện tần suất hoặc tầm quan trọng của mỗi từ. WordCloud được sử dụng rộng rãi trong việc phân tích dữ liệu từ các trang mạng xã hội. [5]

### 4. *Thư viện Pickle*

Pickle được sử dụng để thực hiện chuyển đổi các cấu trúc đối tượng Python sang một dạng byte để có thể được lưu trữ trên ổ đĩa hoặc được gửi qua mạng. Sau đó, luồng ký tự này sau đó có thể được truy xuất và chuyển đổi trở lại sang dạng đối tượng ban đầu trong Python. [18]

Bạn có thể thực hiện chuyển đổi các đối tượng với các kiểu dữ liệu sau sang dạng byte bằng mô-đun Pickle.

- Kiểu Booleans
- Kiểu giá trị số nguyên
- Kiểu giá trị số thực
- Kiểu giá trị số phức
- Kiểu chuỗi ký tự
- Kiểu Tuples
- Kiểu Lists hay danh sách
- Kiểu tập hợp Set
- Kiểu Dictionaries hay từ điển

### 5. *Thư viện Underthesea*

Underthesea là một thư viện xử lý ngôn ngữ tự nhiên (NLP) cho tiếng Việt. Nó cung cấp các công cụ và chức năng để phân tích và xử lý văn bản tiếng Việt, bao gồm

tách từ, gán nhãn từ loại, phân tích cú pháp và trích xuất thông tin. Thư viện này được xây dựng trên cơ sở của thư viện Python khác là Pyvi và có thể được sử dụng để xử lý các tác vụ NLP trong các ứng dụng và dự án liên quan đến tiếng Việt.

Dưới đây là một số nhiệm vụ mà thư viện Underthesea có thể thực hiện:

- Phân đoạn Câu (Sentence Segmentation): Chia văn bản thành các câu riêng lẻ.
- Chuẩn hóa Văn bản (Text Normalization): Đưa dữ liệu văn bản về dạng chuẩn thống nhất, ví dụ như loại bỏ ký tự thừa, chuyển đổi sang dạng chữ thường,...
- Phân tách Từ (Word Segmentation): Chia văn bản thành các từ riêng lẻ.
- Gắn Thẻ từ Loại (POS Tagging): Gắn nhãn cho từ loại của từng từ (danh từ, động từ, tính từ,...).
- Ngắt Nhóm (Chunking): Nhóm các từ thành các cụm từ hoặc đơn vị có nghĩa.
- Phân tích Cấu trúc Phụ thuộc (Dependency Parsing): Phân tích cấu trúc ngữ pháp giữa các từ.
- Nhận dạng Thực thể Tên riêng (Named Entity Recognition): Xác định các thực thể được đặt tên (ví dụ: tên người, địa điểm).
- Phân loại Văn bản (Text Classification): Phân loại văn bản thành các nhóm được xác định trước (ví dụ như tin tức, email, spam,...).
- Phân tích Cảm xúc (Sentiment Analysis): Xác định sắc thái cảm xúc hoặc tình cảm của văn bản (tích cực, tiêu cực, trung lập).
- Phát hiện Ngôn ngữ (Language Detect): Xác định ngôn ngữ của văn bản.
- Chuyển đổi Văn bản thành Giọng nói (Say): Chuyển đổi văn bản viết thành âm thanh nói.

### 6. Thư viện Transformer

Transformer là một thư viện mã nguồn mở được phát triển bởi Hugging Face. Nó cung cấp các công cụ và mô hình cho xử lý ngôn ngữ tự nhiên (NLP), bao gồm cả việc dịch máy, phân loại văn bản, và sinh văn bản. Thư viện này dựa trên kiến trúc Transformer, một mô hình mạng nơ-ron sử dụng cơ chế tự chú ý để xử lý các chuỗi đầu vào.

Transformer cho phép bạn sử dụng các mô hình ngôn ngữ đã được huấn luyện trước (pre-trained models) để thực hiện các tác vụ NLP. Các mô hình này đã được huấn luyện trên các tập dữ liệu lớn và có thể được sử dụng để trích xuất thông tin, phân loại văn bản, và thậm chí tạo ra văn bản mới.

Thư viện Transformer cung cấp một API dễ sử dụng để tải và sử dụng các mô hình đã được huấn luyện trước. Ta có thể sử dụng các mô hình này để thực hiện các tác vụ NLP mà không cần phải huấn luyện lại từ đầu.

### 2.4.2 Mô tả tập dữ liệu

Dữ liệu gốc được thu thập từ Kaggle, một nền tảng trực tuyến dành cho cộng đồng khoa học dữ liệu và học máy [8]. Nguồn dữ liệu này xuất phát từ thử thách phân loại bình luận độc hại, được tổ chức và cung cấp bởi Conversation AI, một nhóm nghiên cứu do Jigsaw và Google thành lập. Theo mô tả, dữ liệu thô được trích xuất từ các bình luận trong phần thảo luận (talk page) trên Wikipedia, một bách khoa toàn thư trực tuyến đa ngôn ngữ.

Tập dữ liệu bao gồm nhiều thuộc tính được mô tả như sau:

- *id*: Là id của từng bình luận trong tập dữ liệu.
- *comment\_text*: Các bình luận dưới dạng văn bản thuần túy, sẽ được xử lý để máy có thể học được.
- Các nhãn trạng thái thể hiện sự độc hại của bình luận:
  1. *toxic*: tính khái quát chung các nhãn còn lại.
  2. *severe\_toxic*: thể hiện mức độ độc hại rất cao.
  3. *obscene*: thể hiện bình luận có sử dụng ngôn từ thô tục.
  4. *threat*: thể hiện bình luận mang tính chất đe dọa.
  5. *insult*: thể hiện bình luận mang tính chất xúc phạm.
  6. *identity\_hate*: thể hiện bình luận mang tính chất xúc phạm thân thể.

Tập dữ liệu này có khoảng 160 nghìn bản ghi và được dùng làm tập dữ liệu chính được để huấn luyện mô hình.

### 2.4.3 Tiền xử lý dữ liệu

#### 2.4.3.1 Xử lý các ký tự đặc biệt

Trong văn bản cũng có thể chứa các ký tự đặc biệt, những ký tự này đôi khi nằm ở những vị trí không thích hợp và làm ảnh hưởng tới câu. Vì lý do đó mà cần loại bỏ chúng ra khỏi tập dữ liệu.

#### 2.4.3.2 Quá trình EDA (Exploratory Data Analysis)

Số lượng bình luận mỗi nhãn được thể hiện dưới dạng barplot ở hình ???. Như có thể thấy, các đánh giá được chia ra thành 6 cột tương ứng gồm “toxic”, “severe\_toxic”, “obscene”, “threat”, “insult”, “identity\_hate”. Có thể thấy “toxic” chiếm số lượng lớn nhất do đây là tiêu chí chung để có thể đánh giá cho bình luận, các thuộc tính khác có thể xem như là mở rộng hoặc bổ sung chi tiết cho thuộc tính “toxic”.

Chúng ta có thể thấy rằng dữ liệu sạch chiếm đa số trong tập dữ liệu, trong khi dữ liệu độc hại chiếm phần nhỏ hơn. Tuy vậy tỉ lệ dữ liệu độc hại không quá thấp, hoàn toàn có thể sử dụng để huấn luyện mô hình.

Dựa vào hình ??? về tương quan giữa các nhãn với nhau, chúng ta có thể nhìn ra được sự tương quan lớn ở một số nhãn. Có 3 cặp nhãn có độ tương quan cao mà có thể được chỉ ra ở đây: cặp “toxic” - “obscene”, cặp “toxic” - “insult” và cặp “obscene” - “insult”.

Dựa vào các mối tương quan trên, có thể đưa ra một số kết luận như sau:

*Ngôn ngữ độc hại thường được dùng trong khi xúc phạm người khác, và khi làm như vậy họ có xu hướng sử dụng ngôn từ thô tục.*

Với kết luận này, có thể dự đoán rằng mô hình được huấn luyện có khả năng bắt được những từ ngữ tục tĩu với độ nhạy tốt.

Đối với những từ thường xuất hiện của mỗi nhãn, ta sử dụng thư viện WordCloud để trực quan các từ có tần suất xuất hiện nhiều trong mỗi nhãn tương ứng của dữ liệu. Kích cỡ càng lớn chứng tỏ mức độ xuất hiện của từ đó rất nhiều và ngược lại, kích cỡ nhỏ tương ứng với việc ít xuất hiện.

### 2.4.4 Thiết lập mô hình

#### 2.4.4.1 Phương pháp đánh giá

Để đánh giá tính chính xác của mô hình, có 3 phương pháp được sử dụng: Precision, CategoricalAccuracy và Recall.

- Precision là tỷ lệ đối tượng được gán các kết quả giống nhau có kết quả thực sự giống nhau.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- Recall là tỷ lệ các đối tượng giống nhau được gán các kết quả giống nhau.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- Accuracy là tỷ lệ các đối tượng được gán đúng với kết quả mẫu. Categorical Accuracy tương đối giống Accuracy nhưng kết quả khi so sánh không phải là một nhãn duy nhất mà là một vector.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

#### 2.4.4.2 Chia dữ liệu

Trước khi thực hiện huấn luyện mô hình, tập dữ liệu huấn luyện được chia thành 2 phần train và test theo tỉ lệ 9:1. Sau đó trong quá trình huấn luyện tập dữ liệu train sẽ được tách thành hai tập train và valid với tỉ lệ 9:1. Hai tập dữ liệu train và valid được sử dụng trong quá trình huấn luyện mô hình trong khi tập dữ liệu test được sử dụng để kiểm thử lại mô hình đã huấn luyện.

#### 2.4.4.3 Tokenization

Tokenization đánh số cho mỗi từ có trong tập từ. Dưới đây là kết quả sau khi thực hiện tokenize tập dữ liệu.



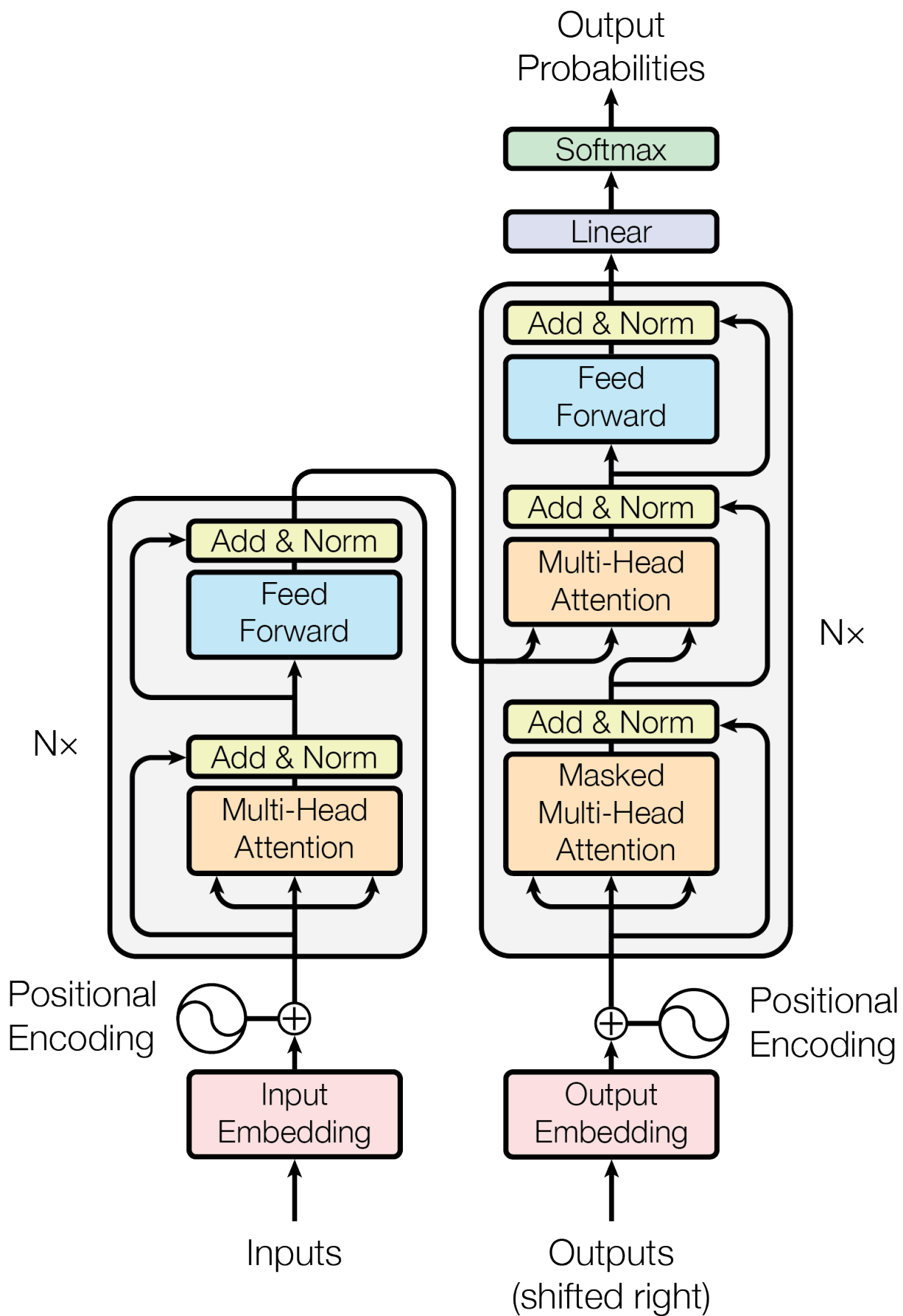
### 2.4.4.4 Word Embedding

Mô hình nhóm sử dụng là mô hình fasttext. Do mô hình này hoạt động dựa trên cách tách các từ thành nhiều phần nhỏ, sẽ dễ dàng hơn để liên hệ với các từ viết thiếu rõ ràng. Mô hình fasttext được sử dụng là một mô hình đã được huấn luyện từ trước chuyên dùng cho tiếng Việt.

$$\text{array}\left(\begin{bmatrix} -0.0203, & -0.0129, & 0.1333, & 0.0378, & -0.0195, & -0.0205, \\ & & & \dots & & \\ 0.0032, & -0.0046, & 0.0358, & -0.0872, & 0.0294, & 0.011 \end{bmatrix}, \right. \\ \left. \text{dtype} = \text{float32}\right)$$

Kết quả trên thể hiện một từ “không” dưới dạng vector từ  $1 \times 300$  trong mô hình fasttext. Tạo một ma trận nhúng từ bao gồm tất cả các từ chứa trong tập dữ liệu. Với 200000 là số từ tối đa có thể xuất hiện và 300 là số chiều của một vector từ, khởi tạo một ma trận với kích thước  $200000 \times 300$  để chứa các vector từ. Với mỗi từ đã được mô hình hóa bởi fasttext, lấy ra từ đó và index của chúng, sau đó lấy ra vector từ tương ứng. Sau cùng thay thế vector đã lấy vào vị trí tương ứng trong ma trận.

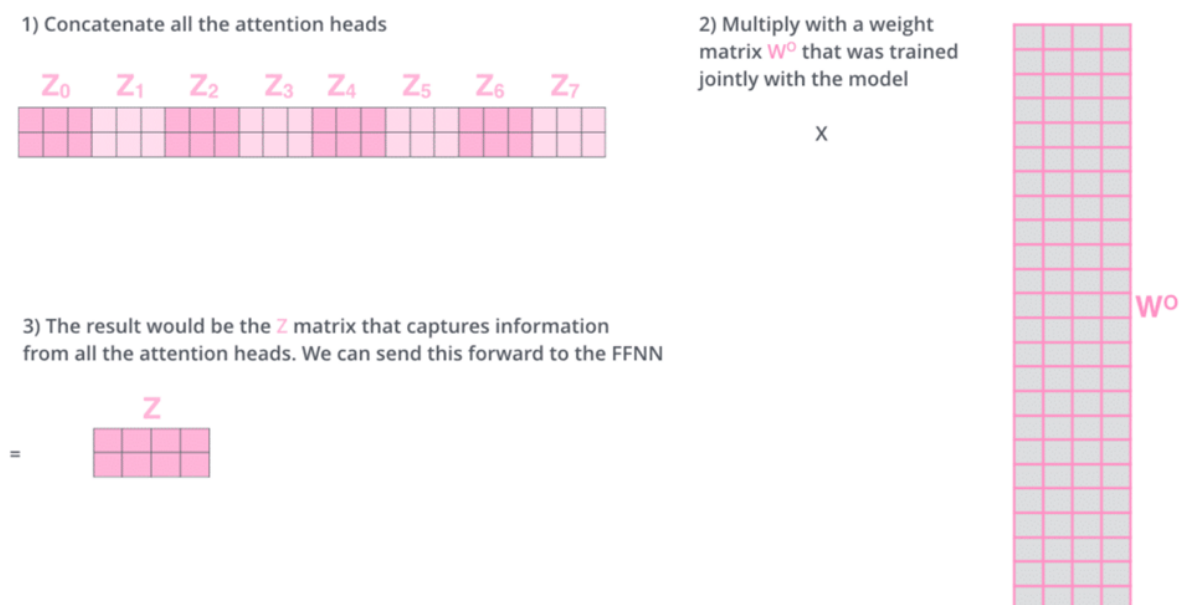
### 2.4.5 Huấn luyện mô hình và đánh giá kết quả



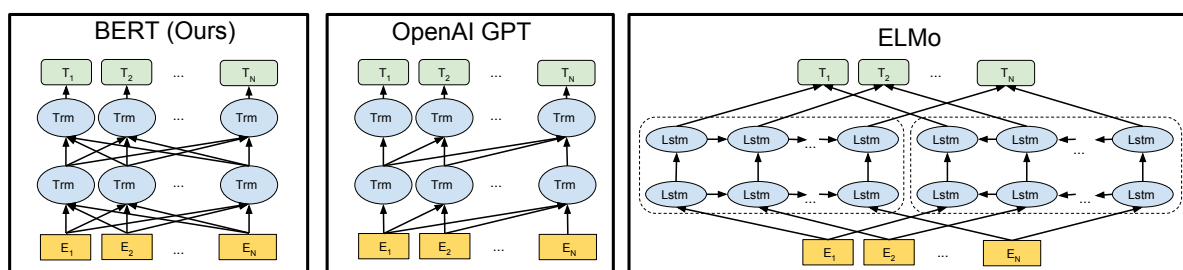
**Hình 2.42:** Kiến trúc của mô hình Tranformer

$$\text{softmax} \left( \frac{\begin{matrix} \text{Q} \\ \text{K}^T \end{matrix}}{\sqrt{d_k}} \right) \begin{matrix} \text{V} \end{matrix} = \begin{matrix} \text{Z} \end{matrix}$$

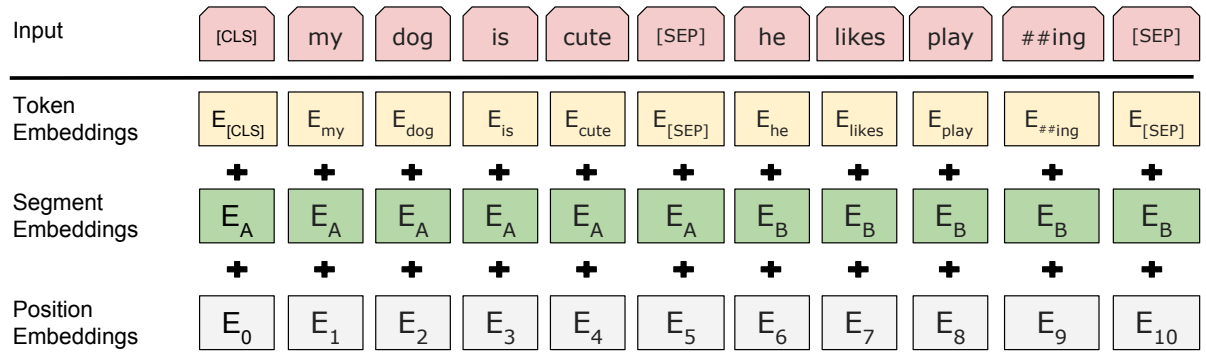
**Hình 2.43:** Phương trình Attention thể hiện dưới dạng ma trận



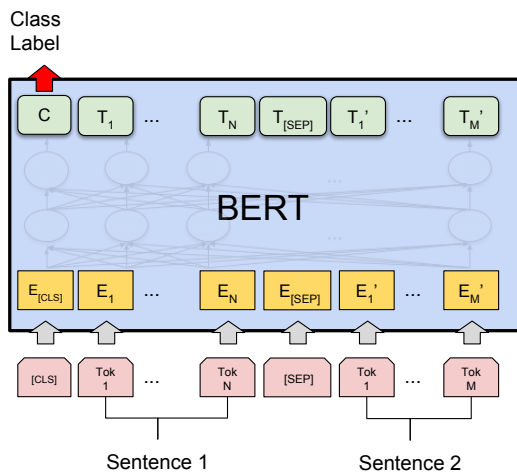
**Hình 2.44:** Multi-head Attention minh họa dưới dạng ma trận



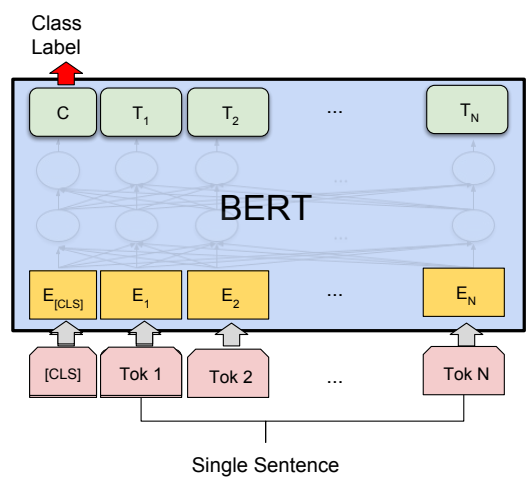
**Hình 2.45:** Sự khác nhau giữa các kiến trúc mô hình pre-trained. BERT sử dụng bidirectional Transformer. OpenAI GPT sử dụng Transformer theo hướng trái sang phải. ELMo sử dụng kết hợp đầu ra của hai LSTM được huấn luyện độc lập theo hướng trái sang phải và phải sang trái. Trong ba mô hình này, chỉ có BERT học được biểu diễn của từ ngữ phụ thuộc lẫn nhau vào cả ngữ cảnh trước và sau ở tất cả các layer.



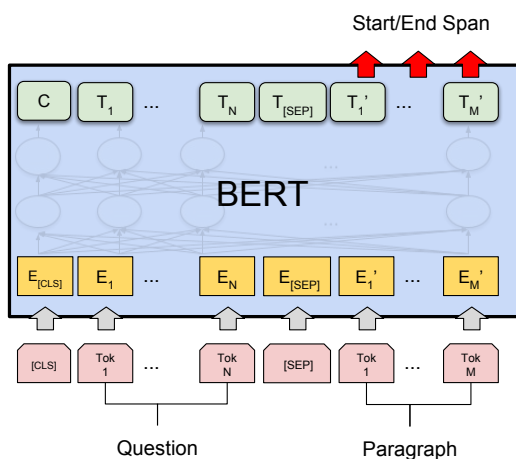
**Hình 2.46:** Biểu diễn đầu vào của BERT: Bao gồm các token embedding, token segment và token position.



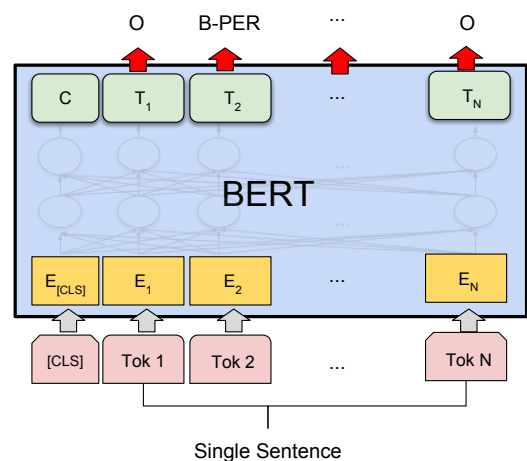
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



(b) Single Sentence Classification Tasks:  
SST-2, CoLA



(c) Question Answering Tasks:  
SQuAD v1.1



(d) Single Sentence Tagging Tasks:  
CoNLL-2003 NER

**Hình 2.47:** Minh họa việc tinh chỉnh BERT cho các tác vụ khác nhau.

**Bảng 2.6:** Tập dữ liệu Kaggle - Toxic Comment Classification Challenge

	<b>id</b>	<b>comment_text</b>	<b>toxic</b>	<b>severe_toxic</b>	<b>obscene</b>	<b>threat</b>	<b>insult</b>	<b>identity_hate</b>
1	0002b...	COCKSUCKER BE- FORE YOU PISS ...		1	1	0	1	0
2	0020e...	Stupid peace of shit stop deleting my ...	1	1	1	0	1	0
3	0020f...	=Tony Sid- away is obvi- ously a fistfuc- kee ...	1	0	1	0	1	0
4	00321...	Stop undo- ing my edits or die!	1	0	0	1	0	0
5	008e0...	Kill all niggers ...	1	0	1	0	1	1
6	00819...	I did re- search thank you very much ...	0	0	0	0	0	0

**Bảng 2.7:** Xử lý các ký tự đặc biệt

comment_text	comment_text
Giải trình\nTại sao các chỉnh sửa được thực hi	Giải trình Tại sao các chỉnh sửa được thực hiệ...
D’Aww! Anh ấy phù hợp với màu nền này màD.	Aww! Anh ấy phù hợp với màu nền này mà ...
Này anh bạn, tôi thực sự không cố gắng chỉnh	Này anh bạn, tôi thực sự không cố gắng chỉnh s...
“\nHơn\nTôi không thể đưa ra bất kỳ đề xuất th	Hơn Tôi không thể đưa ra bất kỳ đề xuất thực...
Ngài là người hùng của tôi. Bất kỳ cơ hội nào	Ngài là người hùng của tôi. Bất kỳ cơ hội nào ...

**Bảng 2.8:** Ví dụ kết quả tokenization

tôi	của	một	bạn	là	có	không	và	các	đã
1	2	3	4	5	6	7	8	9	10