LEARN AI

TTT
22TNT
HCMUS
tongtrongtam1909@gmail.com

Mục lục

Phần I

ML(Machine Learning)

Chương 1

Linear regression

- Hồi quy tuyến tính là mô hình giả định mối quan hệ giữa đầu vào và đầu ra có thể được biểu diễn dưới dạng một đường thẳng, từ đó dự đoán các giá trị đầu vào chưa có nhãn.
- Nói cách khác, bài toán đang cố tìm một hàm tuyến tính:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon$$

  Trong đó:

  - β_i, với $i = 0, 1, 2, \ldots, n$, là các trọng số (weights).
  - ϵ là nhiễu (error term).

- Phương trình hồi quy tuyến tính:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \cdots + \hat{\beta}_n x_n$$

- Biểu diễn ma trận:

$$A = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} & \cdots & x_n^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \quad W = \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \hat{\beta}_2 \\ \vdots \\ \hat{\beta}_n \end{bmatrix} \quad \hat{Y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\Rightarrow \hat{Y} \approx AW$$

- Hàm mất mát Mean-Squared Error (MSE):

$$MSE = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

$$\mathbf{W} = \arg\min_{W} (MSE)$$

- Biến đổi MSE về dạng đại số:

$$\begin{aligned} MSE &= \frac{1}{m}\|AW - Y\|^2 \\ &= \frac{1}{m}(AW - Y)^T(AW - Y) \\ &= \frac{1}{m}(W^T A^T AW - 2W^T A^T Y + Y^T Y) \end{aligned}$$

- Gradient:

$$\begin{aligned} \nabla MSE &= \frac{2}{m}(A^T AW - A^T Y) \\ \nabla^2 MSE &= \frac{2}{m}A^T A > 0 \Rightarrow MSE \text{ có cực tiểu duy nhất} \end{aligned}$$

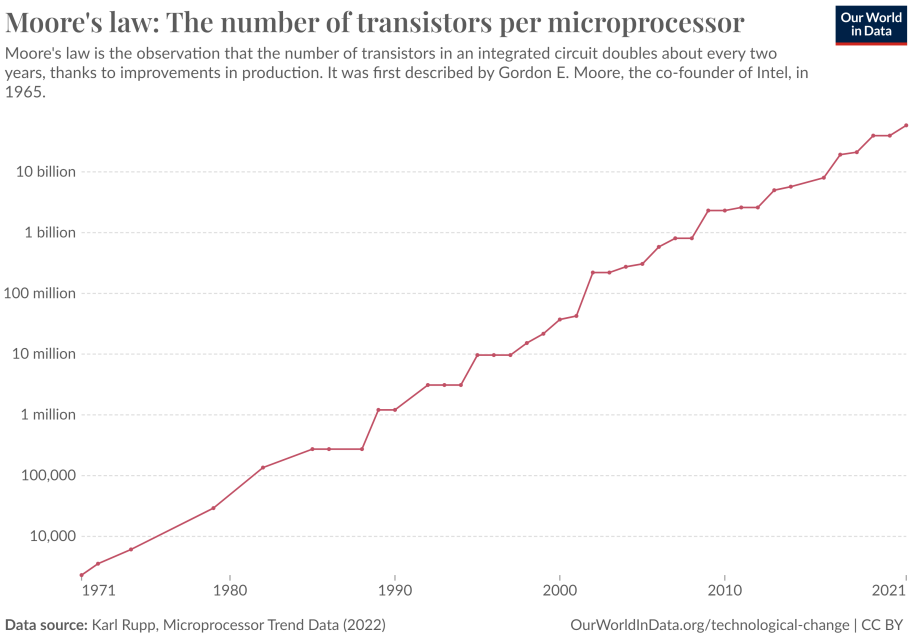
- Giải nghiệm tối ưu:

$$W = (A^T A)^{-1} A^T Y$$

- Ví dụ: Dữ liệu số bóng bán dẫn (N) theo thời gian:

Năm	**Số bóng bán dẫn**
1971	2,250
1972	2,500
1974	5,000
1978	29,000
1982	120,000
1985	275,000
1989	1,180,000
1993	3,100,000
1997	7,500,000
1999	24,000,000
2000	42,000,000
2002	220,000,000
2003	410,000,000

Bảng 1.1: Dữ liệu số bóng bán dẫn theo thời gian



Hình 1.1: Số bóng bán dẫn theo thời gian trong các bộ vi xử lý

- a) Mô hình hoá: $\log_{10}(N) \approx \theta_1 + \theta_2(t - 1970)$
- b) Ước lượng bằng bình phương tối thiểu:

$$A = \begin{bmatrix} 1 & t - 1970 \end{bmatrix}, \quad Y = \begin{bmatrix} \log_{10}(N) \end{bmatrix}, \quad W = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$

$$\Rightarrow W = (A^T A)^{-1} A^T Y \approx \begin{bmatrix} 3.126 \\ 0.154 \end{bmatrix}$$

- Dự đoán năm 2015:

$$\log_{10}(N) = 3.126 + 0.154(2015 - 1970) = 10.06$$
$$\Rightarrow N \approx 10^{10.06} = 1.137 \times 10^{10}$$

- Sai lệch so với thực tế: $|4 \times 10^9 - 1.137 \times 10^{10}| = 7.37 \times 10^9$
- **Nhận xét:** Mô hình tuyến tính trong thang đo log giúp dự đoán hợp lý, nhưng sai số lớn khi biến đổi ngược về giá trị gốc do bản chất phi tuyến trong tăng trưởng bóng bán dẫn.

Chương 2

Gradient descent

- **Gradient Descent** là một thuật toán tối ưu hóa được sử dụng để tìm giá trị tối ưu (minima hoặc maxima) của một hàm. Trong học máy, phương pháp này chủ yếu được sử dụng để tối ưu hóa các tham số của mô hình (ví dụ: trọng số trong mạng nơ-ron).
- **Định nghĩa:** Gradient Descent là một phương pháp lặp đi lặp lại, trong đó các tham số (θ) của mô hình được cập nhật theo hướng đối diện với gradient của hàm mất mát (loss function) tại mỗi bước, với mục tiêu giảm thiểu giá trị của hàm mất mát.
- **Công thức cập nhật tham số:** Ở mỗi bước i, tham số được cập nhật theo công thức:

  $$\theta = \theta - \eta \nabla_\theta J(\theta)$$

  Trong đó:

  - θ: Các tham số cần tối ưu hóa (ví dụ: trọng số của mô hình).
  - η: Tốc độ học (learning rate), điều chỉnh độ lớn của mỗi bước cập nhật.
  - $\nabla_\theta J(\theta)$: Gradient (đạo hàm) của hàm mất mát $J(\theta)$ đối với tham số θ.
- **Vấn đề hội tụ:** Gradient Descent không thể tìm ra giá trị tối ưu chính xác mà chỉ có thể đạt đến một giá trị xấp xỉ. Do đó, thuật toán dừng lại khi sự thay đổi của tham số giữa các bước lặp là rất nhỏ, và không thay đổi đáng kể nữa. Chúng ta thường đặt một ngưỡng ϵ để xác định khi nào thuật toán dừng lại. Điều này được biểu diễn bằng điều kiện:

  $$\left|\theta^{(i+1)} - \theta^{(i)}\right| < \epsilon$$

  Trong đó:

  - ϵ là ngưỡng dừng, giá trị nhỏ để đảm bảo thuật toán dừng khi sự thay đổi là không đáng kể.

- **Tóm tắt thuật toán Gradient Descent:**

Algorithm 1 Gradient Descent Algorithm

Input: Data, learning rate η, stopping threshold ϵ
Initialize: Choose initial values for the parameters $\theta^{(0)}$
repeat
    Compute the gradient: $\nabla_\theta J(\theta)$
    Update the parameters: $\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_\theta J(\theta)$
    Check stopping condition: $\left|\theta^{(i+1)} - \theta^{(i)}\right| < \epsilon$
until Stopping condition is satisfied
Output: Optimized parameters θ

Chương 3

Computational Graph

- **Computational Graph** (Đồ thị tính toán) là một công cụ được sử dụng để mô tả các phép toán toán học trong các mô hình học máy. Đồ thị này giúp mô tả cách dữ liệu và các phép toán di chuyển qua lại trong mô hình, từ đó dễ dàng tính toán gradient, tối ưu hóa tham số và gỡ lỗi mô hình.
- Computational Graph là một đồ thị có hướng, trong đó:
  - **Các đỉnh (nodes)** đại diện cho các phép toán hoặc các giá trị (biến hoặc tham số).
  - **Các cạnh (edges)** đại diện cho sự phụ thuộc giữa các phép toán hoặc giá trị. Cạnh từ một đỉnh này đến đỉnh khác thể hiện rằng kết quả của phép toán tại đỉnh đầu tiên là đầu vào cho phép toán tại đỉnh thứ hai.

- Xét hàm số $y = (c + d) \times (a + c) + \dfrac{e^b}{a + c}$

  Biểu diễn dạng đồ thị:

$a \leftarrow ?$

$b \leftarrow ?$

$c \leftarrow ?$

$d \leftarrow ?$

$t \leftarrow c + d$

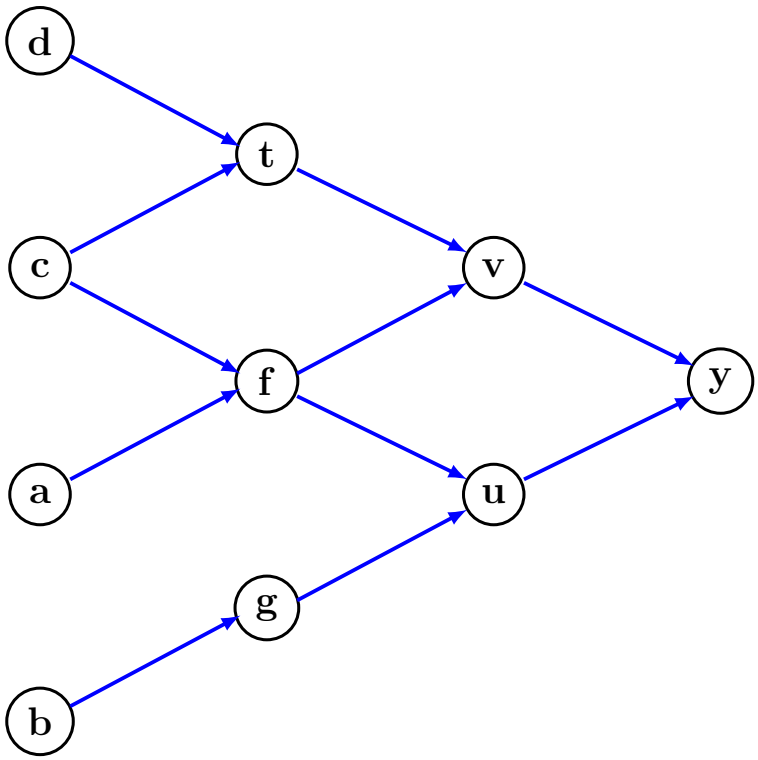
$f \leftarrow a + c$

$g \leftarrow e^b$

$v \leftarrow t \times f$

$u \leftarrow \dfrac{g}{f}$

$y \leftarrow u + v$



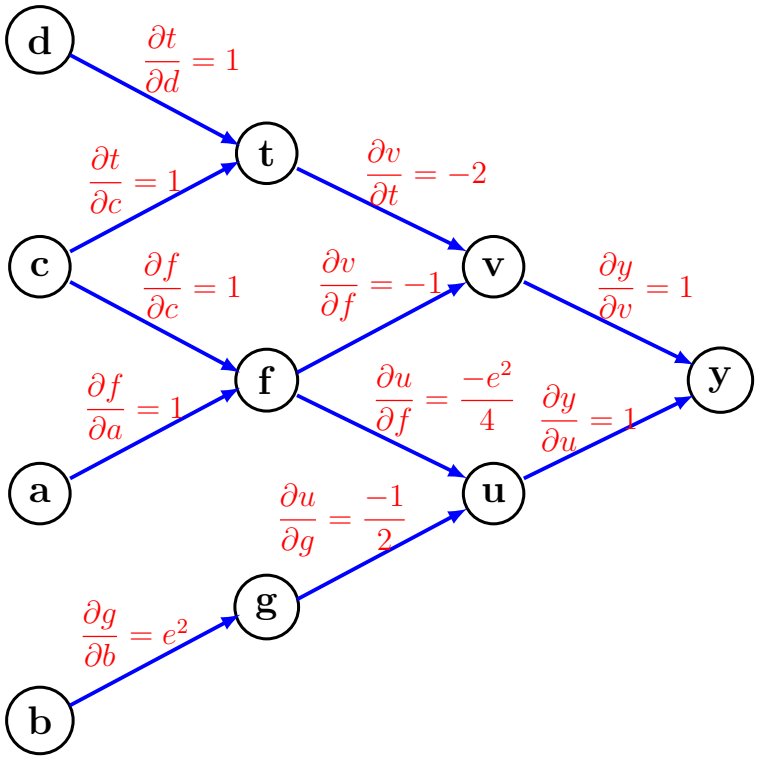
- Một số công thức đạo hàm quan trọng:

f(x)	$\dfrac{df(x)}{dx}$
$\ln x$	$\dfrac{1}{x}$
e^x	e^x
$\sigma(x) = \dfrac{1}{1 + e^{-x}}$	$\sigma(x) \times (1 - \sigma(x))$
$\tanh x = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - (\tanh x)^2$
$\text{ReLU}(x) = \max(0, x)$	$\begin{cases} 1 & \text{nếu } x > 0 \\ 0 & \text{nếu } x < 0 \\ \text{không xác định} & \text{nếu } x = 0 \end{cases}$

- Tính đạo hàm từng phần:

  Xét hàm số ở trên $y = (c+d) \times (a+c) + \dfrac{e^b}{a+c}$

  Cho $a = 1, b = 2, c = -3, d = 2$ Tính $\dfrac{\partial y}{\partial a}, \dfrac{\partial y}{\partial b}, \dfrac{\partial c}{\partial a}, \dfrac{\partial y}{\partial d}$



$$a = 1$$

$$b = 2$$

$$c = -3$$

$$d = 2$$

$$t = -1$$

$$f = -2$$

$$g = e^2$$

$$v = 2$$

$$u = \frac{-e^2}{2}$$

$$y = \frac{-e^2}{2} + 2$$

	Đạo hàm tiến	Kết quả
$\dfrac{\partial y}{\partial a}$	$\dfrac{\partial a}{\partial a} \times \dfrac{\partial f}{\partial a} \times \dfrac{\partial v}{\partial f} \times \dfrac{\partial y}{\partial v} + \dfrac{\partial a}{\partial a} \times \dfrac{\partial f}{\partial a} \times \dfrac{\partial u}{\partial f} \times \dfrac{\partial y}{\partial u}$	$\dfrac{-e^2}{4} - 1$
$\dfrac{\partial y}{\partial b}$	$\dfrac{\partial b}{\partial b} \times \dfrac{\partial g}{\partial b} \times \dfrac{\partial u}{\partial g} \times \dfrac{\partial y}{\partial u}$	$\dfrac{-e^2}{2}$
$\dfrac{\partial y}{\partial c}$	$\dfrac{\partial c}{\partial c} \times \dfrac{\partial f}{\partial c} \times \dfrac{\partial v}{\partial f} \times \dfrac{\partial y}{\partial v} + \dfrac{\partial c}{\partial c} \times \dfrac{\partial t}{\partial c} \times \dfrac{\partial v}{\partial t} \times \dfrac{\partial y}{\partial v}$	-3
$\dfrac{\partial y}{\partial d}$	$\dfrac{\partial d}{\partial d} \times \dfrac{\partial t}{\partial d} \times \dfrac{\partial v}{\partial t} \times \dfrac{\partial y}{\partial v}$	-2

Đạo hàm lùi làm tương tự, kết quả vẫn như đạo hàm tiến.

Phần II

DL(Deep Learning)

Phần III

NLP(Natural Language Processing)

Chương 1

Thuật toán tách từ

Slide: token.pdf

1.1 Byte-Pair Encoding BPE(gpt dùng)

link: here.

1.2 WordPiece(BERT dùng)

Tương tự như thuật toán trên nhưng:

BPE chỉ là đếm số lượng còn với WP thì sẽ chia số số lần xuất hiện để chuẩn hóa lại

vd: $\frac{\textbf{số lần xuất hiện am}}{\textbf{số lần xuất hiện a * só lần xuất hiện m}}$

mục đích: hạn chế việc các từ đơn lẻ xuất hiện quá nhiều

WP còn dùng cặp thăng để nối các token lại

`['[CLS]', 'i', 'am', 'study', '##ing', 'math', '[SEP]']`

$\longrightarrow$ I am studying math.

Chương 2

Chuẩn hóa văn bản

Chuẩn hóa văn bản là một chuỗi việc **chuyển văn bản sang dạng chuẩn, thuân tiện** để sử dụng trong các bài toán khác nhau

Slide: text normalization.pdf

Stanza here

2.0.1 Tách câu(Sentence segmentation)

Chia văn bản thành các câu

2.0.2 Tách token(tokenization)

• Chia văn bản thành các token

• Việc chia này dựa vào dấu câu như (.) ,(?) , (!). Vấn đề khó khăn xảy như như trong từ viết tắt của Tiếng Ánh như Mr. Mrs. thì (.) này không phải là kết của 1 câu.

2.0.3 Lemmatizaion

• 9 cách sử dụng lemmatization : here

• Lemmatization là task xác định hai từ có cùng gốc. Chúng ta sẽ xây dựng một lemmatizer mapping tất cả các từ này về nguyên bản.

vd: buys, bought, buying → buy

• Mục tiêu: thu gọn không gian phân tích, tạo ra độ chính xác cao hơn

• Nhược điểm: Nếu bộ dữ liệu có sự mập mờ lớn thì việc xử lý này sẽ đánh mất đi thông tin về ngữ nghĩa, ngữ pháp giảm độ chính xác của mô hình.

Vd: có thể là 1 câu mỉa mai bảo nhiều nma thật ra lại là ít.

• Thực hành lemma

  ◦ **Natural language toolkit:** here (dùng nhiều)

  ◦ **spaCy:** here (dùng nhiều nhất) Kiến trúc Spacy : here

- **TextBlob:** here.
- **Standford coreNLP:** here.

2.0.4 Stemming

• Cắt hậu tố khỏi từ, ít được sử dụng hơn lemmatization

Vd: buying cắt bỏ ing → buy

2.0.5 Lọc stop words

• Lọc những từ hay xuất hiện và ít ngữ nghĩa như "the", "is", "at","wich"và "on"

2.0.6 Sửa sai từ(word correction)

Cấu trúc dữ liệu trie: here

• Sai thứ tự từ trong Tiếng Anh hoặc sai dấu trong Tiếng Việt.

happpy ⟶ happy

azmazing ⟶ amazing

inteliggent ⟶ intelligent

Jaccard distance

dùng n- grams

ví dụ với bigram:xét 2 từ hello với elhlo:

- hello → {he,el,ll,lo}
- hlleo → {hl,ll,le,eo}
- union = {ll}
- intersection = {he,el,ll,lo,hl,le,eo}

$\Longrightarrow$Jaccard distance $= 1 - \dfrac{len(union)}{len(intersection)} = \dfrac{6}{7}$

Edit distance

Edit distance hay còn được gọi là Levenshtein distance đo lường **tổng số phép nhỏ nhất** cần thực hiện biến đổi chuỗi A thành chuỗi B

Số loại phép biến đổi bao gồm :

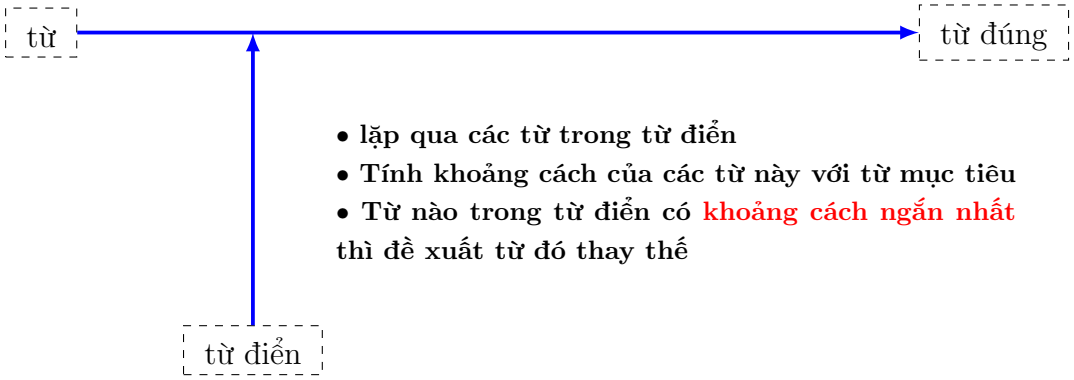
- Phép chèn (insertion)
- Phép xóa (deletion)
- Phép thay thế (Substitution)

Ví dụ: kitten → sitting

⟹ Edit distance = 3: cần ít nhất 3 phép để chuyển kitten thành sitting:

- thay k thành s (kitten → sitten)
- thay e thành i (sitten → sittin)
- thêm g ở cuối (sittin → sitting)

Quy trình sửa sai từ (word correction pipeline)



2.1 Bài toán

- Chuẩn hóa văn bản **khác nhau** giữa các bài toán.
- Tùy bài toán khác nhau thì sẽ lựa chọn bộ chuẩn hóa, không phải cái nào cũng sài.

2.1.1 Bài toán sinh từ(text Generation)

- Giữ nhiều token nhất có thể, đưa các văn bản về chung một format.

Ví dụ: lùi đầu dòng, viết hoa đầu câu,..

2.1.2 Bài toán phân loại cản xúc(Sentiment Classification)

- Loại bỏ các stop words như a, to,...
- Giữ lại biểu tượng cảm xúc như :),:D,...

Data:

Text	Sentiment
One of the other reviewers has mentioned that	positive
A wonderful little production. The	positive
Basically there's a family where a little boy	negative

Pipeline

csv file → **cleaned text** → **tokenized matrix** → **classification model**

2.2 Làm việc với Tiếng Việt

Một số lưu ý khi làm việc với Tiếng Việt: here

2.2.1 spacy cho Tiếng Việt :

2.2.2 Thư viện underthesea

• Bộ công cụ NLP tiếng Việt - Underthesea là một bộ công cụ bằng ngôn ngữ Python mã nguồn mở bao gồm các mô-đun, bộ dữ liệu và hướng dẫn hỗ trợ nghiên cứu và phát triển trong lĩnh vực Xử lý Ngôn ngữ Tự nhiên tiếng Việt. Thư viện có các API dễ sử dụng để nhanh chóng áp dụng các mô hình NLP tiền huấn luyện cho văn bản tiếng Việt của bạn, như phân đoạn từ (Word segmentation), gán phần loại từ (PoS), nhận diện thực thể đặt tên (Named entity recognition - NER), phân loại văn bản (Text Classification) và phân tích phụ thuộc (Dependency Parsing).

• Chi tiết here

• **Tách văn bản thành các câu**

• **Chuẩn hóa văn bản**

Vị trí dấu được điều chỉnh phù hợp.

• **Tách câu thành từ**

• **POS Tagging - Đánh nhãn từ**

• **Phân tích cảm xúc câu**

Chương 3

Các thuật toán traning

3.1 Stochastic gradient descent(SGD)

•Sử dụng quán tính mô hình có khả năng hội tụ nhanh hơn và vượt qua các điểm **cực tiểu cục bộ (yên ngựa)**

•Công thức quán tính (Momentum):

- $\theta_{t+1} = \theta_t - v_t$
- $v_t = \gamma v_{t-1} + \alpha \bigtriangledown J(\theta_t)$ $(v_0 = 0 \rightarrow v_1 = \alpha \bigtriangledown J(\theta_1))$

3.2 Adagrad

Tài liệu here và here

vấn đề của quán tính là càng ngày càng được tích tụ dẫn tới việc tối ưu vượt khỏi điểm cực trị.

• **Adagrad** có khả năng điều chỉnh tốc độ học dựa vào tham số

• Thực hiện cập nhật:

- **Ít** với tham số liên quan đếm features thường xuyên xuất hiện **(tốc độ học giảm)**
- **Nhiều** với tham số liên quan đếm features ít xuất hiện **(tốc độ học lớn)**

→ phù hợp với dữ liệu thưa

•**Adagrad** sử dụng tốc độ học khác nhau cho những tham số khác nhau ở một **timestep t**.

Khi Gradient lớn, tốc độ học giảm, khi Gradient nhỏ, tốc độ học tăng.

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{G_t + \epsilon}} \odot \bigtriangledown J(\theta)$$

Tổng dồn bình phương Gradient **từ đầu** đến **timestep t hiện tại**

$$G_t = \sum_{j=1}^{t} G_j \odot G_j = \sum_{j=1}^{t} \bigtriangledown J(\theta_j) \odot \bigtriangledown J(\theta_j)$$

$$\theta_t \in \mathbf{R}^{d\times d} \quad G_t \in \mathbf{R}^{d\times d}$$

• Điểm mạnh: Xóa bỏ việc điều chỉnh tốc độ học bằng tay

• Điểm yếu: G_t đặt ở dưới thương. Được cộng dồn từ đầu tới mỗi timestep t nhất định, khả năng cao mẫu rất lớn khiến tốc độ học giảm xuống xấp xỉ 0, dẫn tới mô hình **không học được thông tin mới**. Để khác phục điều này ta tới tiếp các một số thuật toán tiếp theo.

3.2.1 Adadelta

• **Adadelta** sẽ khác phục vấn đề của **Adagrad**, vấn đề tốc độ học sẽ chạm tới khi khi timestep đủ lớn.

• Thay vì tính tổng dồn bình phương gradient **từ đầu** đến **timestep t** hiện tại, Adadelta chỉ tính tổng dồn w lần gradient tới thời điểm hiện tại.

• Thay vì sử dụng w chúng ta sẽ thực hiện như sau:

$$g_t = \bigtriangledown J(\theta_t)$$

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2$$

Tương tự như momentum: $\gamma = 0.9$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

$\longrightarrow$ Thay vì sử dụng một learning rate cố định, Adadelta sử dụng một phương pháp điều chỉnh learning rate dựa trên tỷ lệ của các bước cập nhật trước đó. Điều này có nghĩa là khi các tham số càng xa thì bước cập nhật sẽ được điều chỉnh tự động để phù hợp với mức độ cần thiết.θ_t chỉ quan tới tới các θ ở gần nó mà thôi.

• **Thuật toán RMSprop** Tương tự như Adadelta nhưng chưa được công bố.

3.3 Adam (Adaptive Moment Estimation)

Tài liệu: here

• **Adam** ngoài việc lưu lại trung bình bình phương gradient trong quá khứ như **Adadelta và RMSprop**, **Adam** còn lưu lại trung bình giống **Momentum**

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t \Longrightarrow \text{Đo lường trung bình Gradient (Mean)}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2 \Longrightarrow \text{Đo lường phương sai Gradient (Variance)}$$

Thời điểm ban đầu m_0 và v_0 bằng 0, mà β_1 và β_2 gần tới 1 dẫn tới m_1 và v_1 gần tới 0 (bị bias tới 0). Cho nên tác giả quyết định xử lý vấn để này bằng cách:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Tương tự Adadelta và RMSprop

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

• Ở đây có 1 sự khác biệt thay vì dùng $\sqrt{\hat{v}_t + \epsilon}$ thì lại dùng $\sqrt{\hat{v}_t} + \epsilon$

• Ở đây $\beta_!$ và β_2 là các tham số trọng số không âm. Các lựa chọn phổ biến cho chúng là $\beta_1 = 0.9$ và $\beta_2 = 0.999$

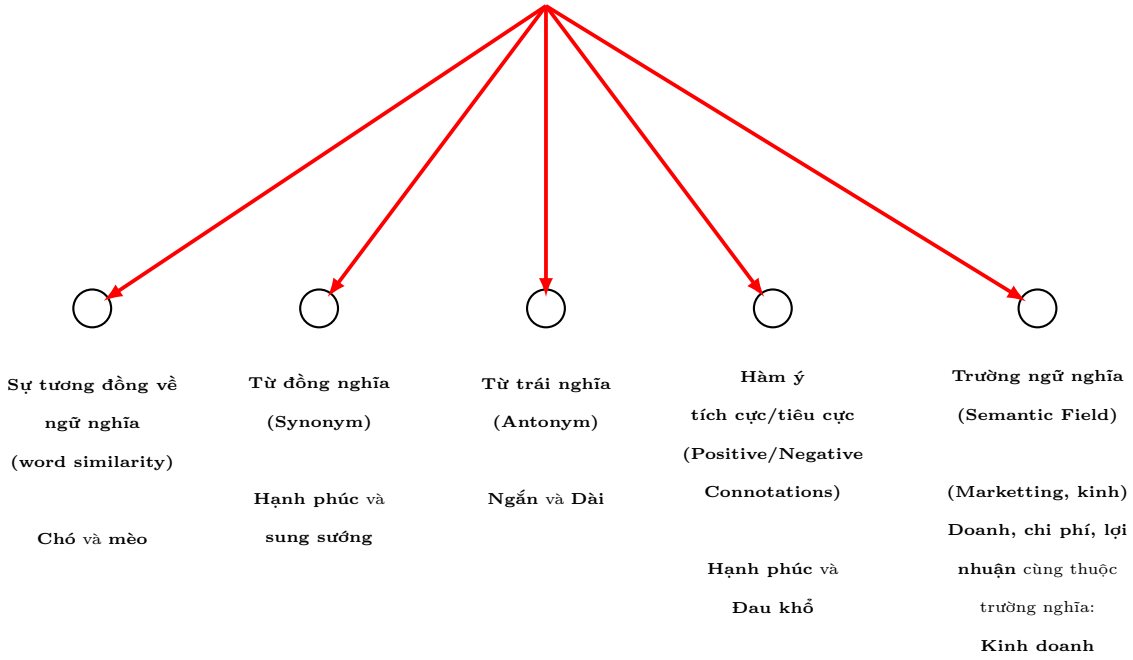
Chương 4

Vector Semantics và Embeddings

4.1 Vector Semantics

• Lexical semantics

Phân tích đại diện ngữ nghĩa trên đơn vị là **từ - word meaning** (nghiên cứu về ý nghĩa của một từ, và hệ thống kết nối ý nghĩa của một từ)



◦ Word similarity

Việc mở rộng phạm vi đánh giá 2 từ thay vì sử dụng mối quan hệ **Word senses**(đồng nghĩa, trái nghĩa) sang quan hệ tương đồng **(Word similarity)** làm **đơn giản độ phức tạp** của bài toán.

Simlex - 999 Data set (khoảng từ 0 - 10)

achieve	accomplish	8.57	⟵	Đồng nghĩa
accept	reject	0.83	⟵	Trái nghĩa
water	ice	6.47	⟵	Có sự tương đồng ngữ nghĩa

4.1.1 Vector ngữ nghĩa (Semantic vector)

• Ngữ nghĩa của một từ được định nghĩa thông qua **Phân bố (distribution)** của nó trong việc sử dụng ngôn ngữ.

• Hai từ có cùng **phân bố (similar distribution - các từ xung quanh tương đồng nhau)** sẽ có ngữ nghĩa giống nhau

$\Longrightarrow$ Ý nghĩa của **Vector ngữ nghĩa** là biểu diễn cho một từ dưới dạng **không gian ngữ nghĩa nhiều chiều (multidimensional sematic space)**, không gian này được xây dựng dựa trên mối quan hệ với các từ xung quanh.

4.2 Embeddings

Mô phỏng từ bằng vector nhiều chiều, những từ tương đồng nhau thì vector sẽ gần nhau.

Word embedding được sử dụng để biểu diễn từ trước khi đưa vào mô hình NLP

• **Vector thưa(Sparse vector)**: Yếu trong việc mô tả tương đồng ngữ nghĩa, ví dụ 2 vector của từ "Gà"và "Động vật"có thể hoàn toàn khác nhau.

• **Vector dày(Dense vector)**:(E.g. 300 chiều) hoạt động tốt hơn Sparse vector(E.g. 10000 chiều) trong các NLP task vì Dense vector yêu cầu ít params hơn giảm test error(generation error) - **tránh overfitting**

• **One-hot vector:** là vector chứa duy nhất một phần tử là 1 còn tất cả là 0. Giá trị của từ trong từ điển sẽ là vị trí của 1 trong vector. Chiều của one - hot vector bằng số lượng từ trong từ điển

`Dense vector` `Sparse vector` `One-hot vector`

$$\text{eat} = \begin{bmatrix} -0.2 \\ 0.1 \\ 0.5 \\ 2.8 \\ -1.7 \end{bmatrix} \quad \text{eat} = \begin{bmatrix} 0 \\ 20 \\ 0 \\ 3 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \quad \text{eat} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} (\text{dict['eat']} = 2)$$

4.2.1 Word2vec

Tài liệuhere

Word2Vec là một embedding tĩnh (static embedding), một từ được biểu diễn dưới dạng một vector cố định (fixed embedding)

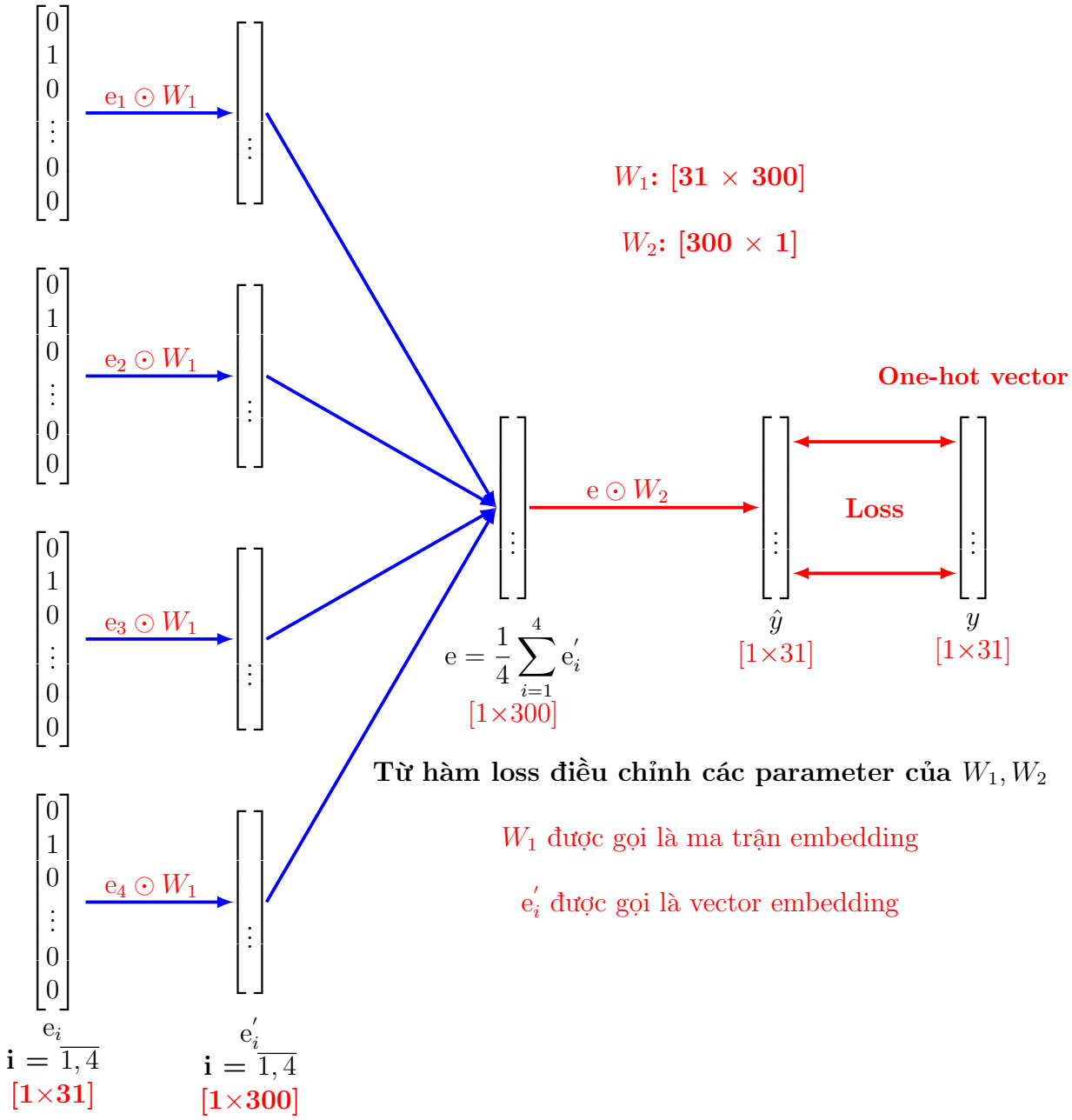
Những Embedding tiên tiến hiện nay như BERT hoặc ELMO biểu diễn từ dưới dạng vector ngữ cảnh động (dynamic contextual embedding), tức là một từ sẽ có biểu diễn khác nhau trong những ngữ cảnh khác nhau.

4.2.2 Continuous Bag Of Words(CBOW)

Giải thích toán: here

Ý tưởng: dùng các từ ngữ phía trước và sau để dự đoán từ ở giữa. Ở đây sẽ dùng 2 từ trước và 2 từ sau

Ví dụ từ điển có 30 từ + OOV = 31

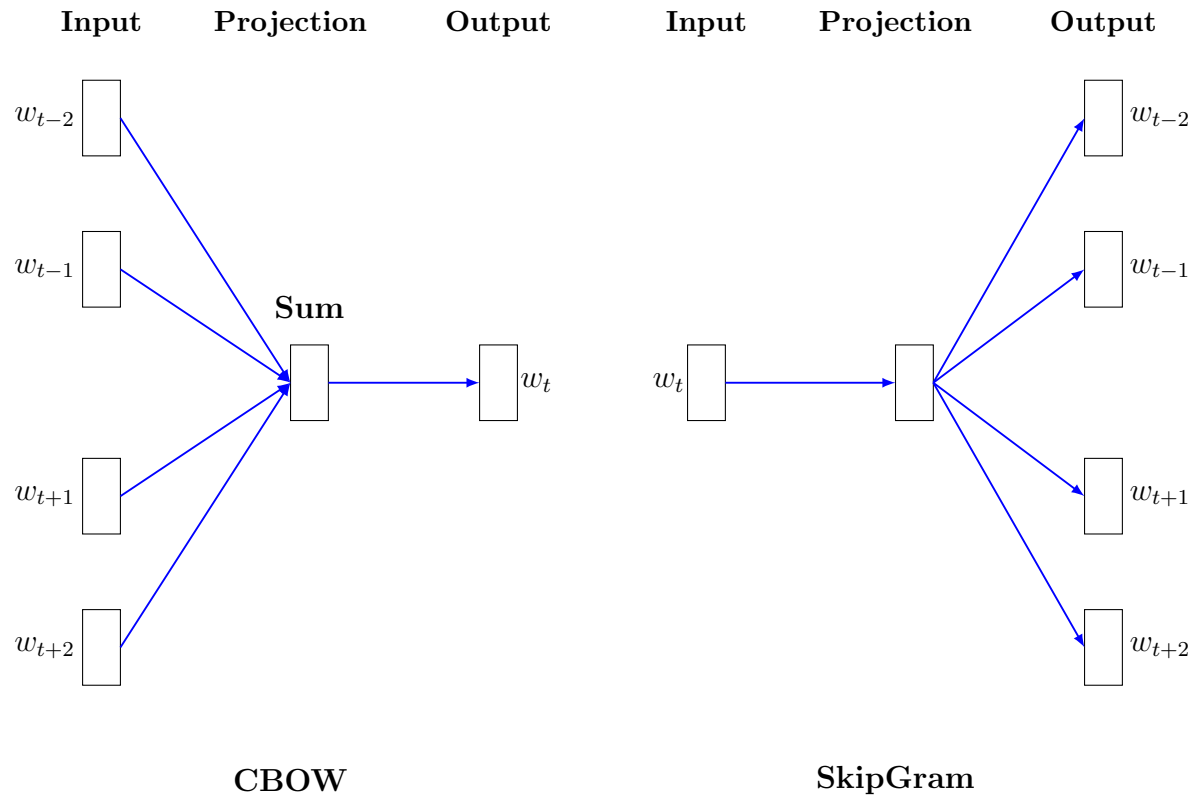


Với i là vị trí số 1 trong one-hot vector của 1 từ thì vector embedding của từ đó sẽ là hàng i của ma trận embedding.

4.2.3 SkipGram

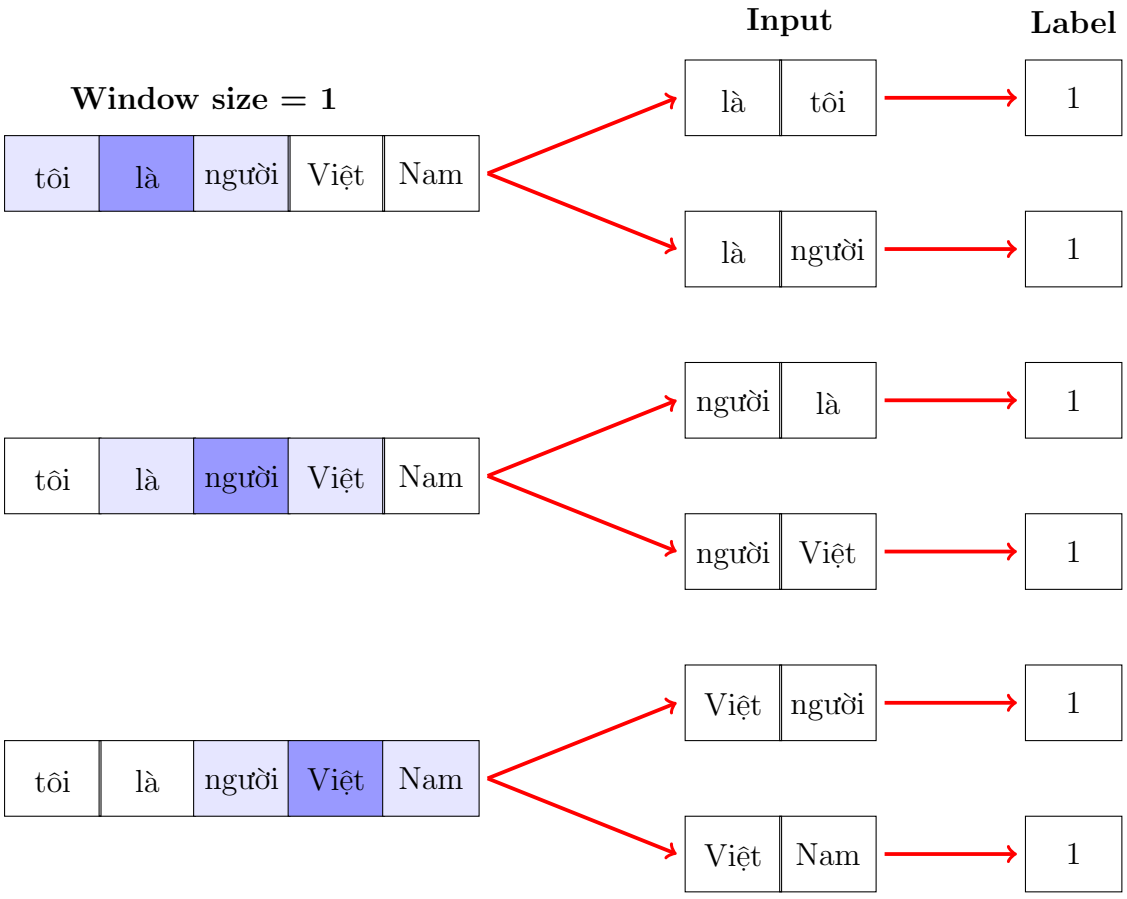
Vấn đề của CBOW là khi từ điển quá lớn thì tốc độ sẽ chậm đi đáng kể, để khắc phục điều này thì thuật toán SkipGram xuất hiện.

Ý tưởng: Ngược lại với CBOW, SkipGram dự đoán những từ xung quanh dựa vào 1 từ cho trước

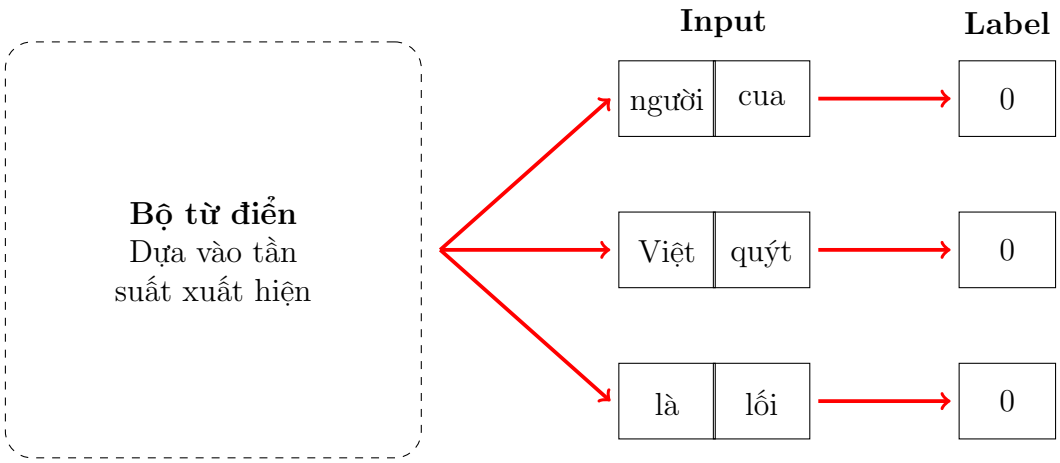


CBOW **SkipGram**

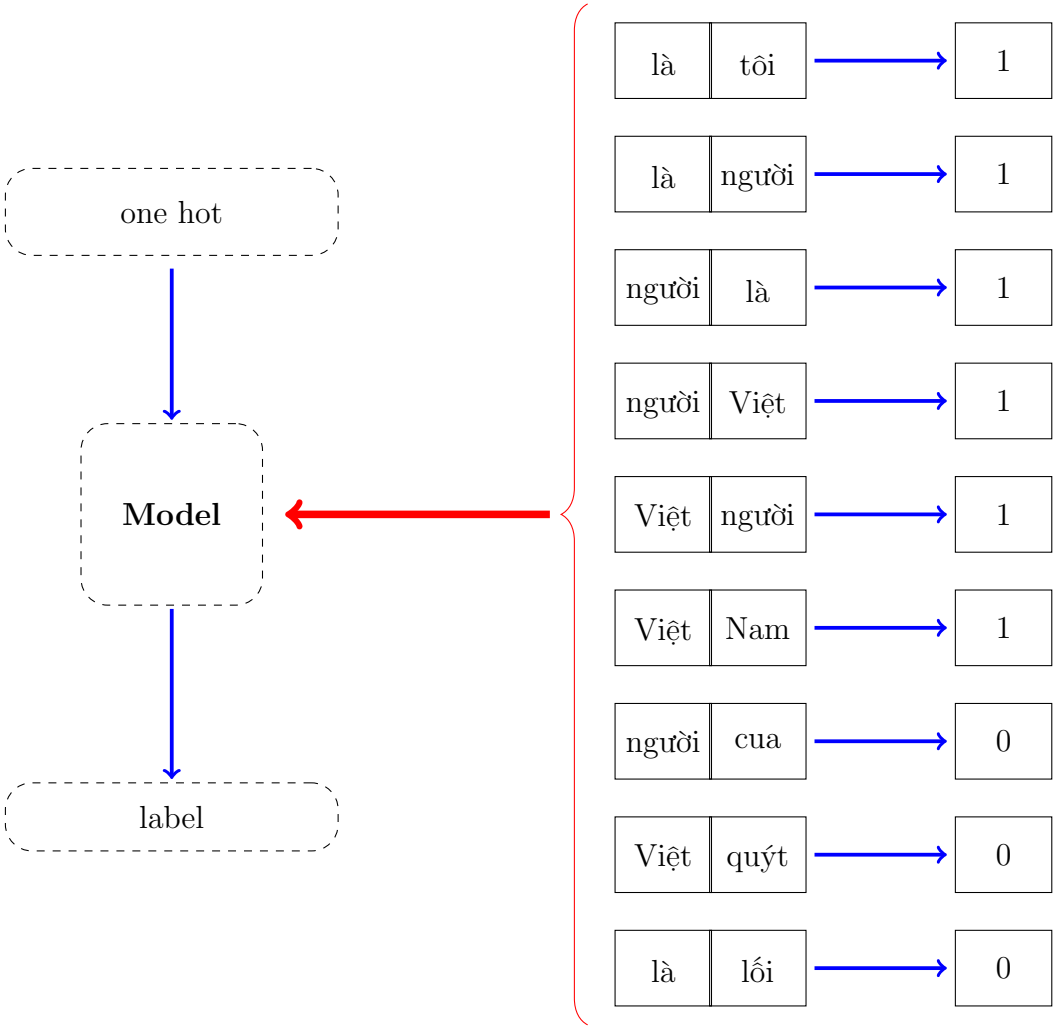
- **Sinh cặp token đi liền với nhau và đánh nhãu 1(positive)**



• **Negative sampling. Lựa chọn các cặp token không đi cùng nhau, đánh nhãn 0.**



• **Classification model(xây dựng mô hình phân loại)**



4.2.4 Glove(Global Vectors for Word Representation)

4.2.5 Fast test

Fast test là một thư viện mã nguồn mở, miễn phí, nhẹ nhàng cho phép người dùng **học các biểu diễn văn bản** và **phân loại văn bản.** Nó hoạt động trên phần cứng tiêu chuẩn chung.

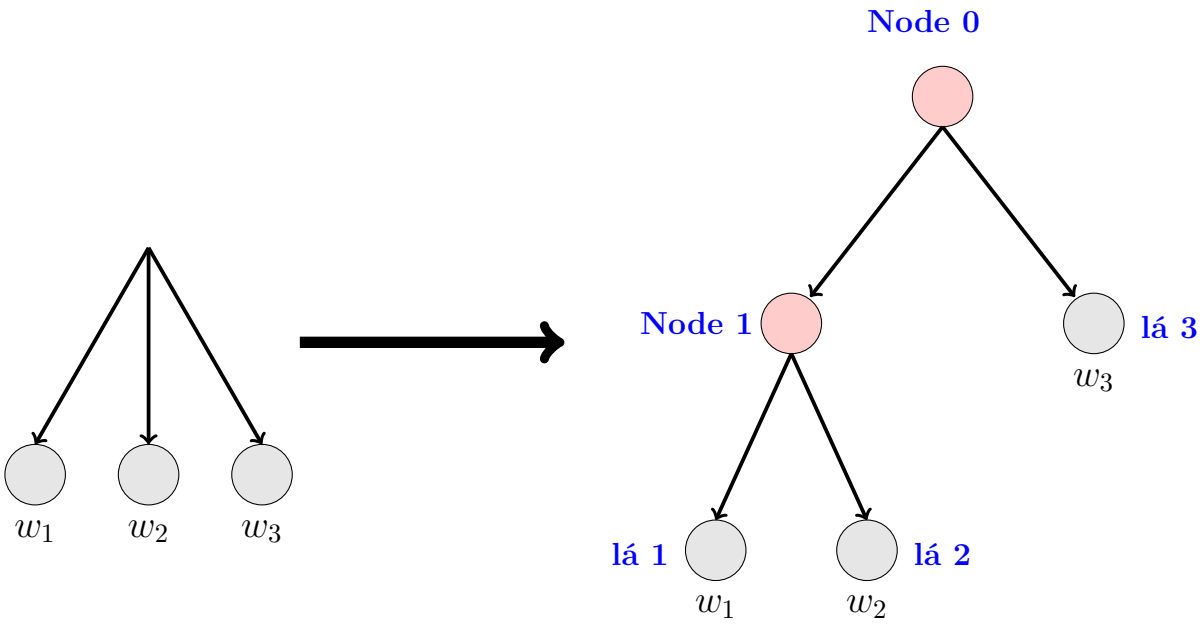
Fast test sử dụng CBOW tuy nhiên ứng dụng kĩ thuật tối ưu bổ xung nhằm tăng tốc training đó là **hierarchical softmax** (softmax phân cấp)

4.2.6 Hierarchical softmax

Hierarchical softmax thay thế cho softmax thông thường thành dạng cay trong đó các từ là node lá của cây.

Softmax thông thường yêu cầu tính toán trên bộ tập từ điển với độ phức tạp **O(N)**. Nếu chúng ta biến đổi thành cây nhị phân, chúng ta chỉ cần theo đường đến node mục tiêu với độ phức tạp **O(log(N))**

Chú ý: số node của cây = số từ trong từ điển(số lá) - 1



Mỗi node là mô hình phân loại đầu ra nhị phân

Sigmoid activation function:

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

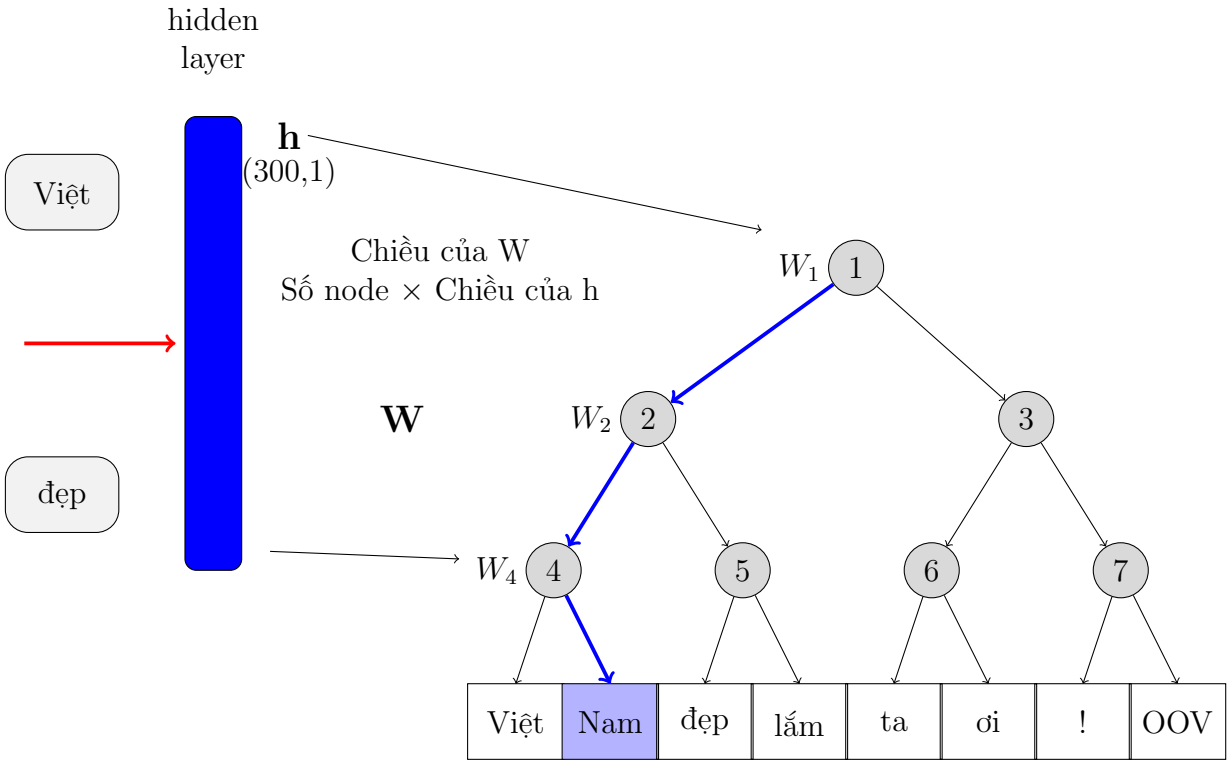
$$\sigma(-x) = 1 - \sigma(x)$$

Softmax function:

$$a_i = \frac{e^{z_i}}{\sum_i (e^{z_i})}$$

$$\sigma(-x) = 1 - \sigma(x)$$

example : **Việt Nam đẹp lắm ta ơi**



Mục tiêu cực đại hóa xác xuất: **P(Nam|context)**

P(Nam|context) = **P(Đi sang trái khi ở node 1|context)**
× **P(Đi sang trái khi ở node 2|context)**
× **P(Đi sang phải khi ở node 3|context)**

= **P(Đi sang trái khi ở node 1|context)**
× **P(Đi sang trái khi ở node 2|context)**
× **(1 - P(Đi sang trái khi ở node 3|context))**

= **sigmoid**($W_1.h$) × **sigmoid**($W_2.h$) × **sigmoid**(**-**$W_4.h$)

• **Quá trình training** : Khi training đã biết nhãn thì chỉ cập nhật các dòng của ma trận W để đi được đến nhãn khi duyệt cây.

• **Khi Inferrence:** Sử dụng W và kết quả sigmoid để xác định hướng. Sigmoid >0.5 → đi qua phải,Sigmoid <0.5 → đi qua trái.

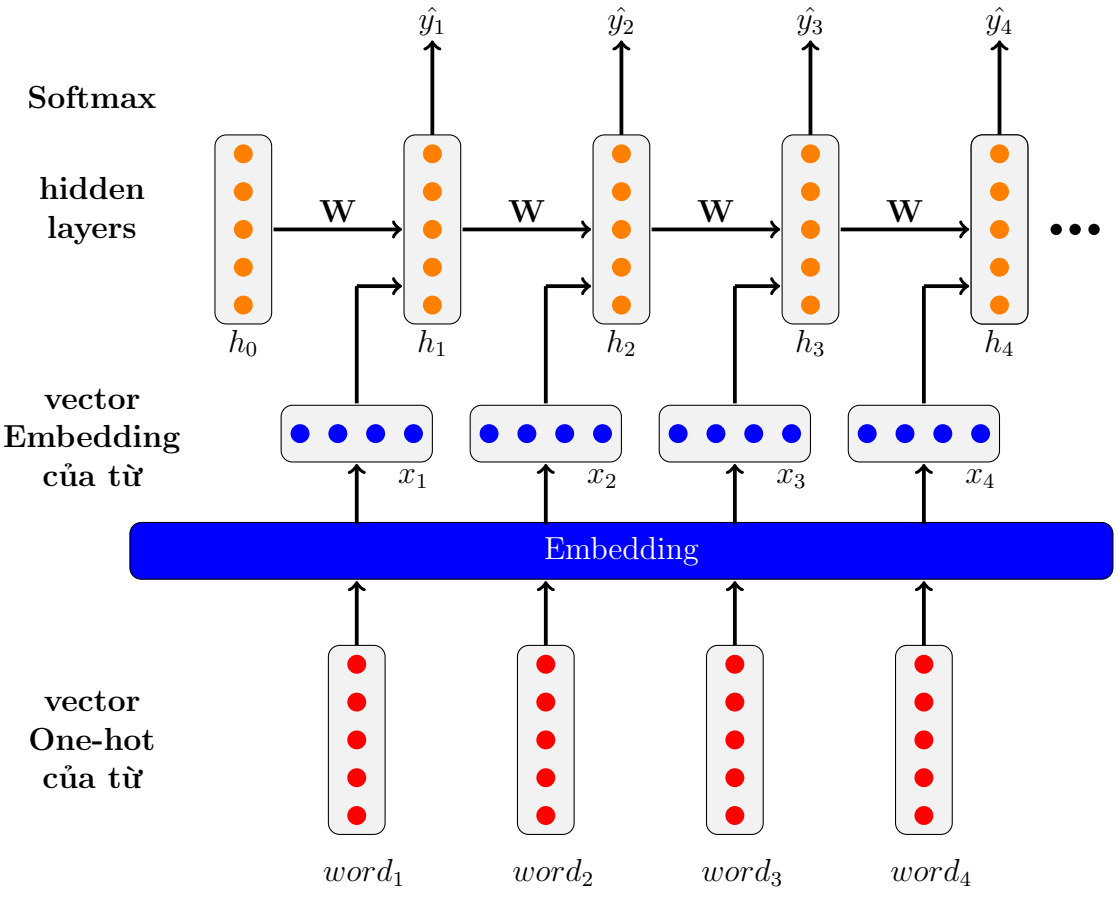
Chương 5

Mô hình ngôn ngữ(language model)

Mô hình ngôn ngữ (Language Model) được thiết kế với mục tiêu đo lường phân phối xác suất của các đơn vị ngôn ngữ (từ, chữ).

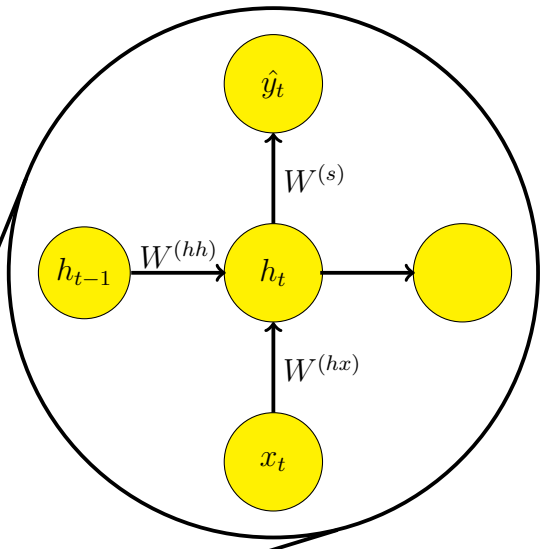
5.1 RNN(Recurrent Neural Network)

Ghi chú hay : here



$$h_t = \sigma(W^{(hh)}h_{t-1} + W^{hx}x_t + bias)$$

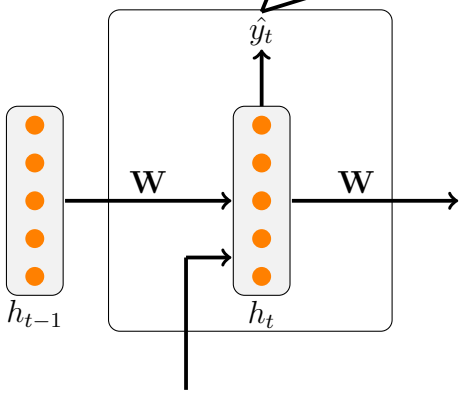
$$\hat{y}_t = softmax(W^(s)h_t)$$



Xác suất xảy ra một từ
bất kỳ trong từ điển
khi cho lịch sử

$$P(x_{t+1} = v_j | x_t, \cdots, x_1) = \hat{y}_{tj}$$

$$W = \{W^{(hh)}, W^{(hx)}, W^{(s)}\}$$



$$x \in \mathbf{R^D}$$

$$W_{(hh)} \in \mathbf{R^{D_h \times D_h}}$$

$$W_{(hx)} \in \mathbf{R^{D_h \times d}}$$

$$\hat{y} \in \mathbf{R^{|V|}}$$

$$h_t \in \mathbf{R^{D_h}}$$

$|V|$: chiều của từ điển (20000)

D_h : chiều của vector h (32)

d : chiều của vector embedding (64)

Mất mát tại thời điểm t: $J^{(t)}(W) = -\sum_{j=1}^{|V|} y_{t,j} log(\hat{y}_{t,j})$

Mất mát trên câu dài T: $J = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_{t,j} log(\hat{y}_{t,j})$

Ưu điểm :

- Khả năng xử lí đầu vào với bất kì độ dài nào
- Kích cỡ mô hình không tăng theo kích cỡ đầu vào
- Quá trình tính toán sử dụng các thông tin cũ

• Trọng số được chia sẻ trong suốt thời gian

Nhược điểm:

• Tính toán chậm

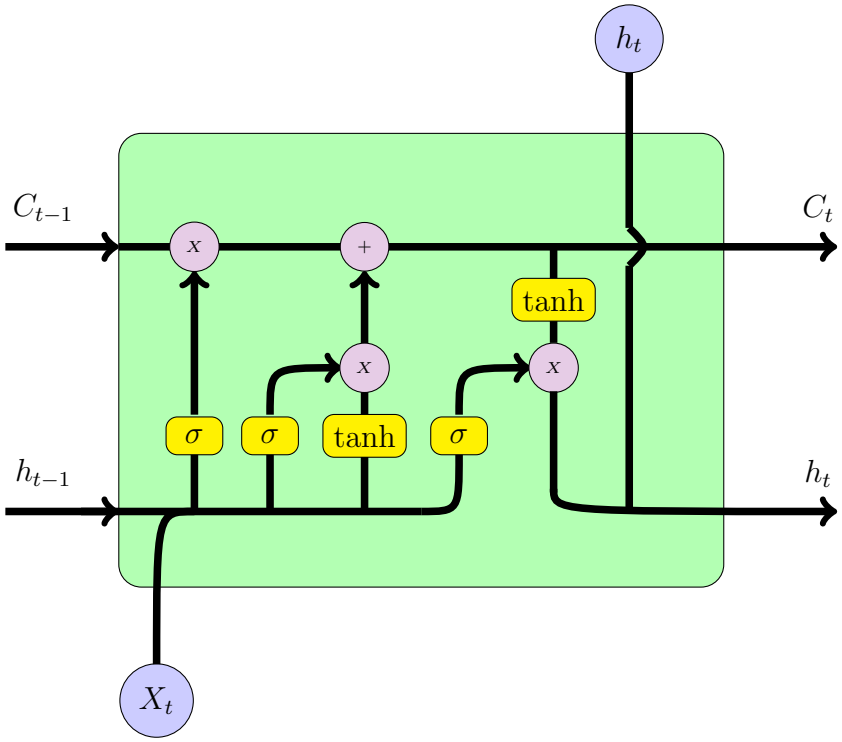
• Khó để truy cập các thông tin từ một khoảng thời gian dài trước đây

• Không thể xem xét bất kì đầu vào sau này nào cho trạng thái hiện tại

5.2 LSTM (Long short-term memory)

Cũng cùng ý tưởng như RNN nhưng thay vì việc lấy tất cả chúng ta sẽ chỉ lấy 1 phần nào đó.

Add 3 gates: Forget Ouput Input



$f_t = \sigma(W_f x_t + U_f h_{t-1})$: **Forget gate** quên đi bao nhiêu (giữ lại bao nhiêu của cái cũ)

$i_t = \sigma(W_i x_t + U_i h_{t-1})$: **Input gate** lấy bao nhiêu của đầu vào mới

$\tilde{C}_t = \mathbf{tanh}(W_c x_t + U_c h_{t-1})$: **New memory cell** học như bình thường

$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t$: **Final memory cell** học f_t của cũ và i_t của mới học được ở $\tilde{C}_t$

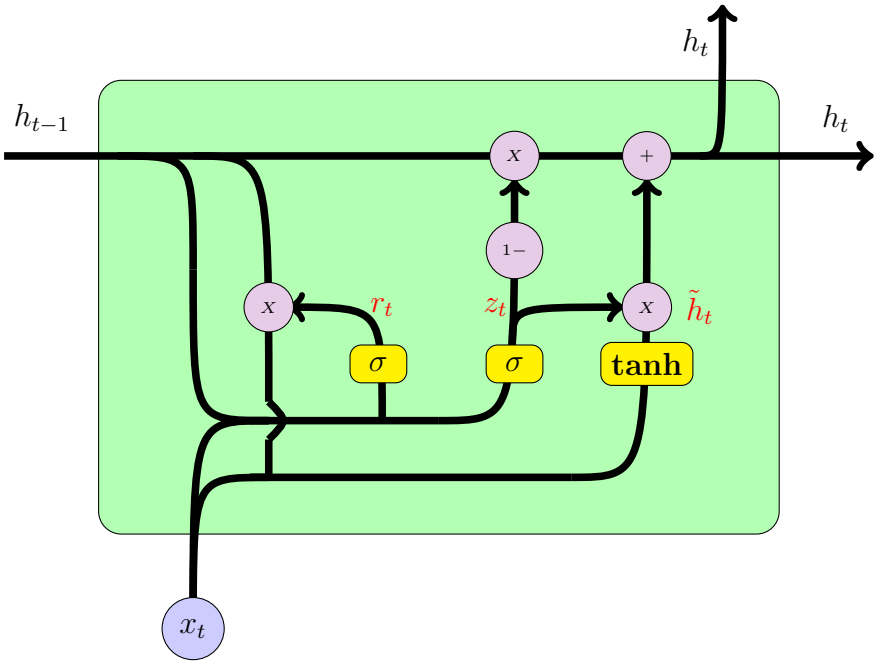
$o_t = \sigma(W_o x_t + U_o h_{t-1})$: **output/Explode gate** xuất ra bao nhiêu

$h_t = o_t \circ \mathbf{tanh}(C_t)$: chỉ muốn out ra o_t đã học được

• Quyết định của các cổng đều **phụ thuộc** vào hidden state phía trước và từ hiện tại thông qua các giá trị của hàm Sigmoid(trong khoảng từ 0 đến 1)

5.3 GRU(Gated Recurrent Unit)

GRU có cơ chế tương tự LSTM tuy nhiên ít tham số hơn và không sử dụng cell state



$z_t = \sigma(W_s.[h_{t-1}, x_t])$

$r_t = \sigma(W_r.[h_{t-1}, x_t])$

$\tilde{h}_t = \textbf{tanh}(W.[r_t * h_{t-1}, x_t)]$

$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$

• **Perplexity(PP)**

PP của một Language Model trên một tập test là xác xuất nghịch đảo (inverse probability), và được chuẩn hóa(normalized) trên số lượng tự. pp **CÀNG NHỎ** thì model **CÀNG TỐT**

Gỉa sử tập test: $W = w_1 w_2 ... w_n$

$$PP(W) = P(w_1 w_2 ... w_n)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_n)}}$$

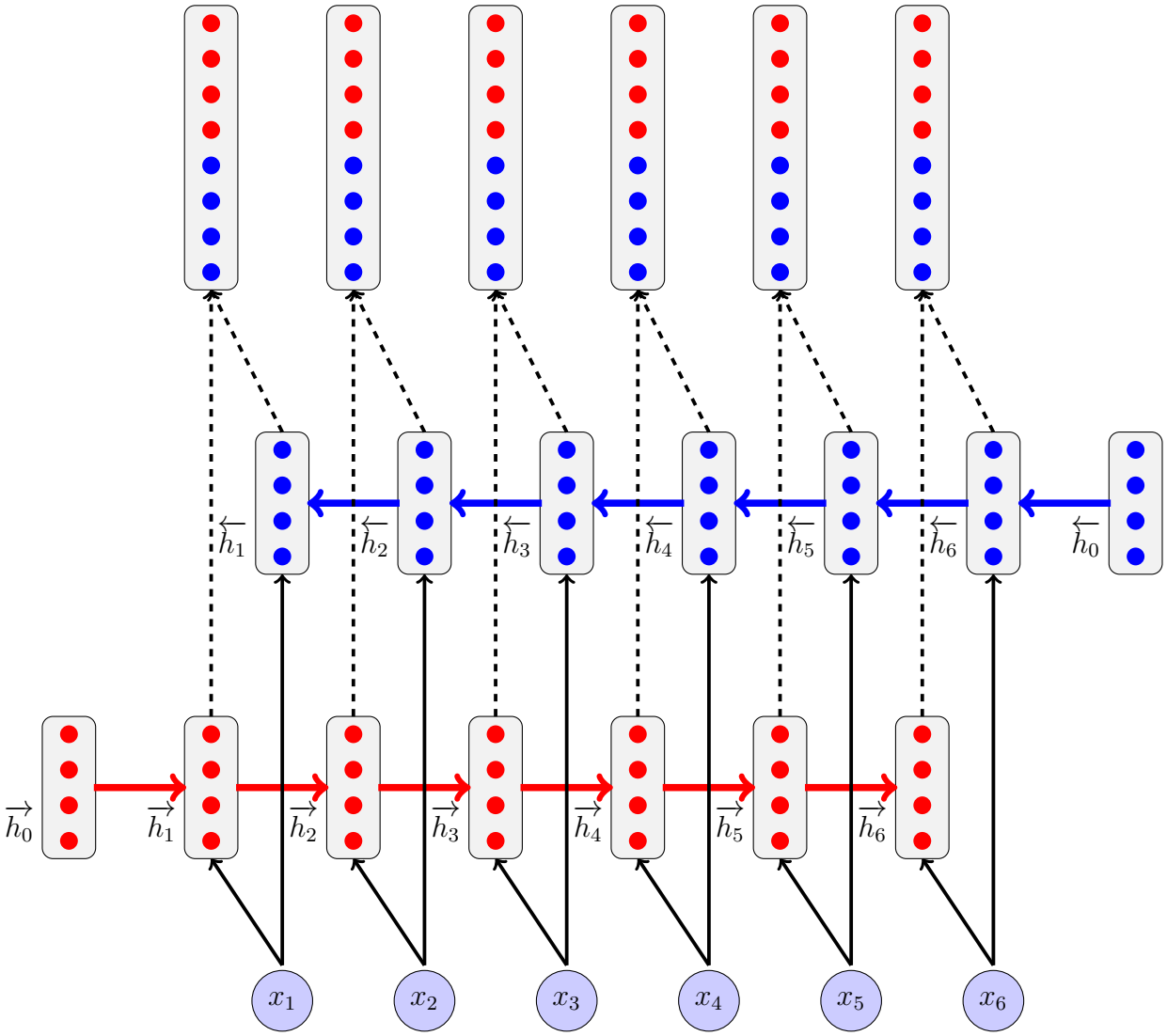
Sử dụng chain rules:

$$P(w_1 w_2 ... w_n) = P(w_1)P(w_2|w_1)...P(w_n|w_1 ... w_{n-1}) = \sum_{i=1}^{n} P(w_i|w_1 ... w_{i-1})$$

Kết luận:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{n} \frac{1}{P(w_i|w_1 ... w_{i-1})}}$$

5.4 Bidirectional RNN



• **RNN1**(Backward layer): Tổng hợp thông tin từ trái sang phải

$$\overrightarrow{h_t} = RNN_1(\overrightarrow{h_{t-1}}, x_t)$$

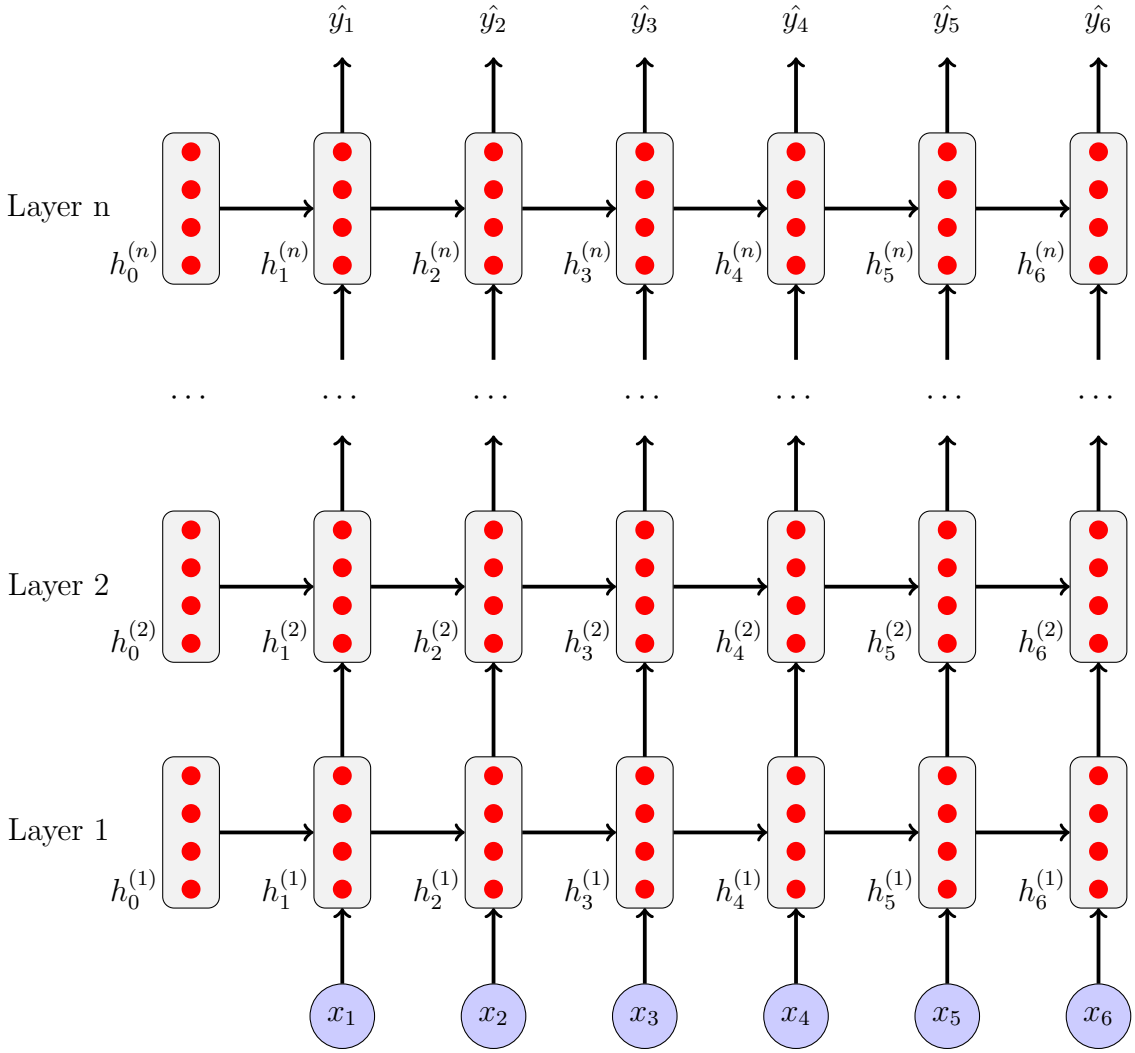
• **RNN2**(forward layer): Tổng hợp thông tin từ phải sang trái

$$\overleftarrow{h_t} = RNN_2(\overleftarrow{h_{t+1}}, x_t)$$

$\Longrightarrow$ sau đó 2 vector này được nối vào nhau đại diện cho thông tin ngữ cảnh tại thời điểm t .

$$h_t = \left[\overrightarrow{h_t}, \overleftarrow{h_t}\right]$$

5.5 Deep RNN


$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$ $\hat{y}_5$ $\hat{y}_6$
Layer n
$h_0^{(n)}$ $h_1^{(n)}$ $h_2^{(n)}$ $h_3^{(n)}$ $h_4^{(n)}$ $h_5^{(n)}$ $h_6^{(n)}$
Layer 2
$h_0^{(2)}$ $h_1^{(2)}$ $h_2^{(2)}$ $h_3^{(2)}$ $h_4^{(2)}$ $h_5^{(2)}$ $h_6^{(2)}$
Layer 1
$h_0^{(1)}$ $h_1^{(1)}$ $h_2^{(1)}$ $h_3^{(1)}$ $h_4^{(1)}$ $h_5^{(1)}$ $h_6^{(1)}$
x_1 x_2 x_3 x_4 x_5 x_6

Chương 6

Machine Translation and Attention

6.1 Bài toán dịch máy

6.1.1 Mô hình dịch máy thống kê(Statitic Machine Translation)

Ví dụ:

x: **câu Tiếng Việt**

y: **Câu Tiếng Anh**

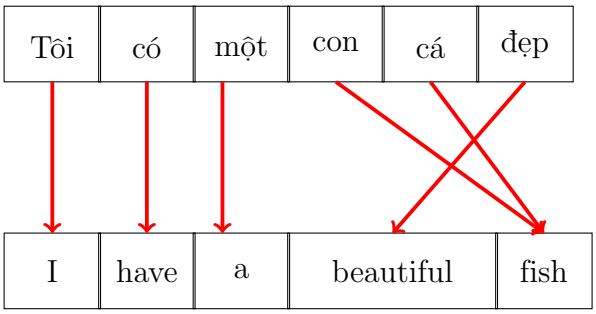
Dịch từ tiếng việt sang Tiếng Anh tìm y phù hợp nhất khi có x.

$$\arg\max_y P(y|x) = \arg\max_y P(x|y)P(y)$$

Mô hình dịch
Translate model
học cách dịch từ và cụm từ
sử dụng ngữ liệu song song

Mô hình ngôn ngữ
Language model
Học cách viết Tiếng Anh tốt nhất
sử dụng dữ liệu đơn ngữ

- **Sự tương xứng**



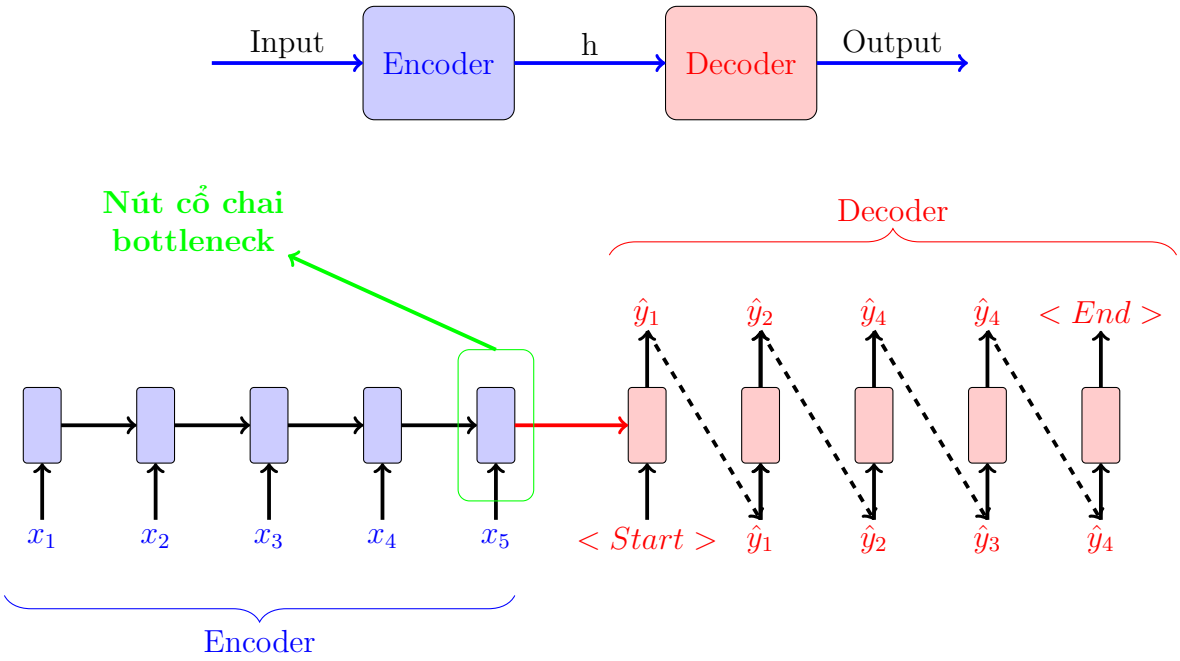
Alignment là **sự tương xứng** giữa những từ nguồn cụ thể với những từ được dịch tương ứng.

Vấn đề của mô hình dịch máy thống kê: phức tạp, tốn nguồn lực con người.

6.1.2 Dịch máy dùng mạng nơ ron(Neural Translate Machine)

Sử dụng mạng nơ ron để cải thiện việc dịch.

Mạng có tên Sequence-to-sequence(viết tắt seq2seq) bao gồm 2 mạng RNN nhỏ.



• **The Bottleneck Problem:**

–Khả Năng Dài Hạn Kém: RNN khó giữ thông tin từ các chuỗi dài.

–Gradient Vanishing/Exploding: Khó khăn trong việc học các mối quan hệ dài hạn.

–Thiếu Tập Trung: Không có khả năng tập trung vào các phần quan trọng của chuỗi đầu vào.

–Hiệu Suất Kém Với Chuỗi Dài: Xử lý chuỗi đầu vào dài kém hiệu quả.

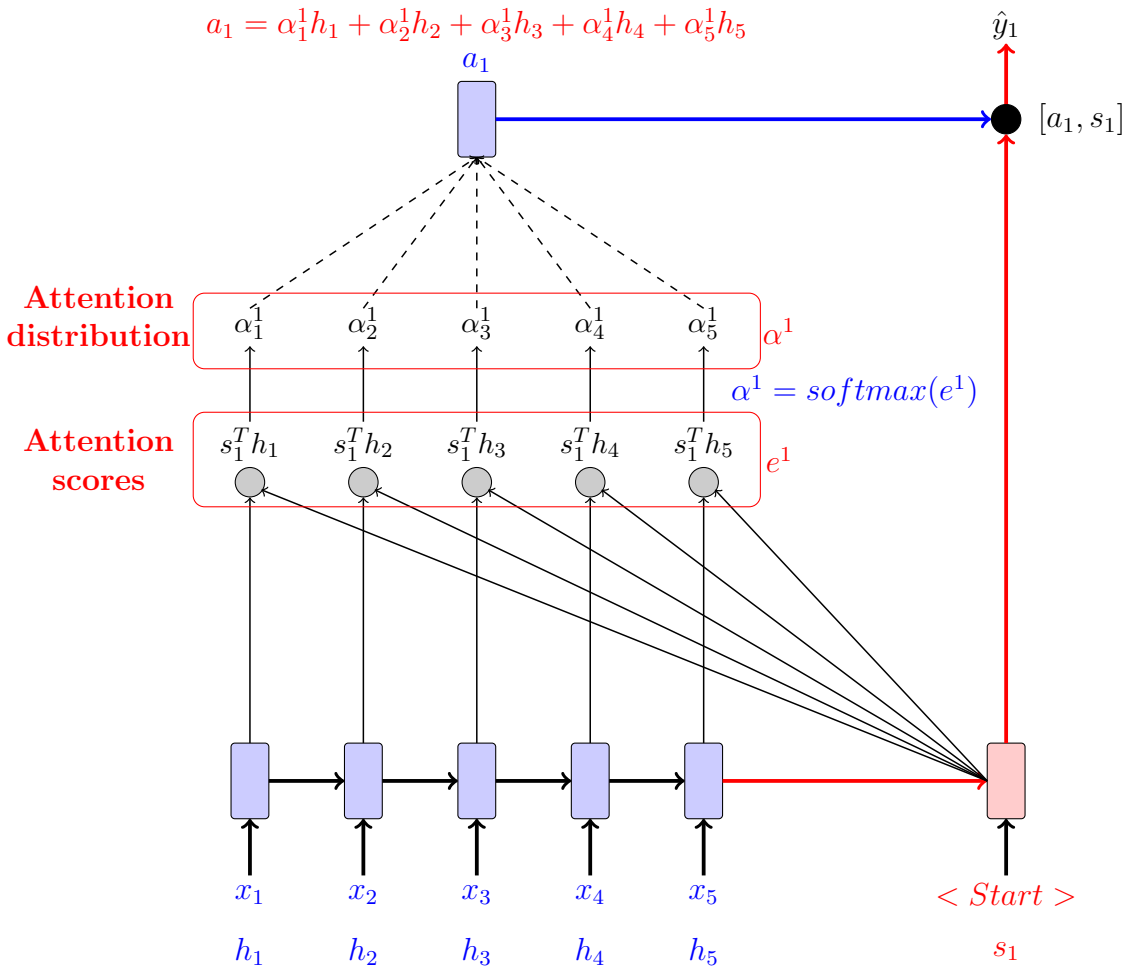
–Khó Cập Nhật Mô Hình: Tinh chỉnh mô hình khó khăn hơn.

–Tốc Độ Huấn Luyện Chậm: Huấn luyện tốn thời gian và tài nguyên.

–Tổng Hợp Thông Tin Kém: Tổng hợp thông tin trong một vector ẩn duy nhất hạn chế.

6.2 Cơ chế Attention

Ý tưởng: ở mỗi bước của **Decoder** kết nối trược tiếp đến **Encoder** để tập trung vào **một phần cụ thể** của **câu nguồn**.



Cơ chế Attention:

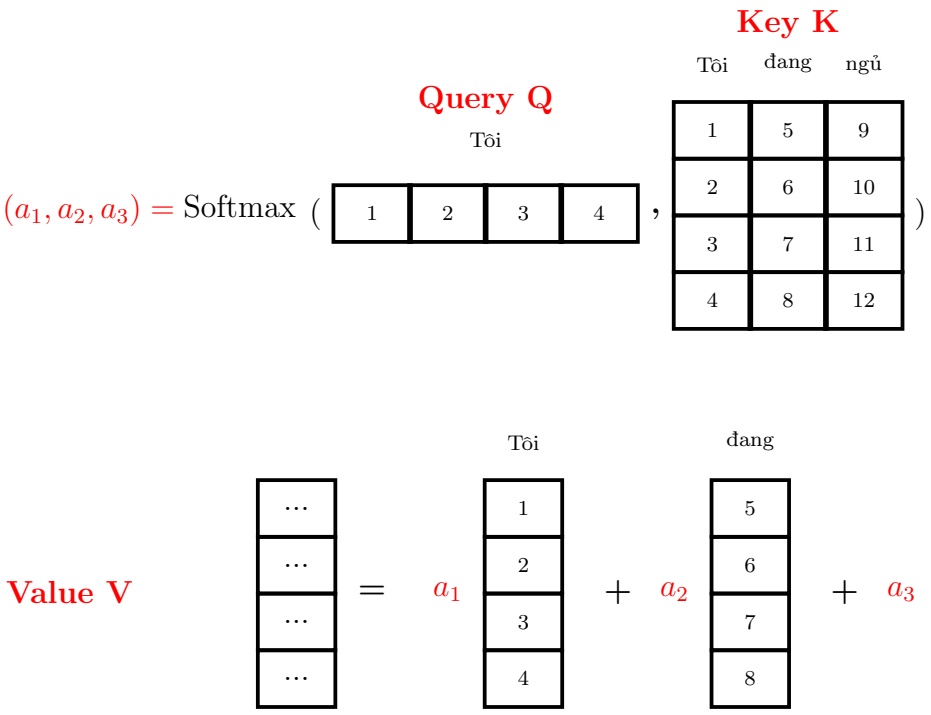
- **Cải thiệu hiệu năng** của NMT
- **Giải quyết** vấn đề tính tương xứng
- **Giải quyết** vấn đề nút cổ chai

6.3 Bleu scores

Tài liệu : here

Code: here

6.4 Self-Attention

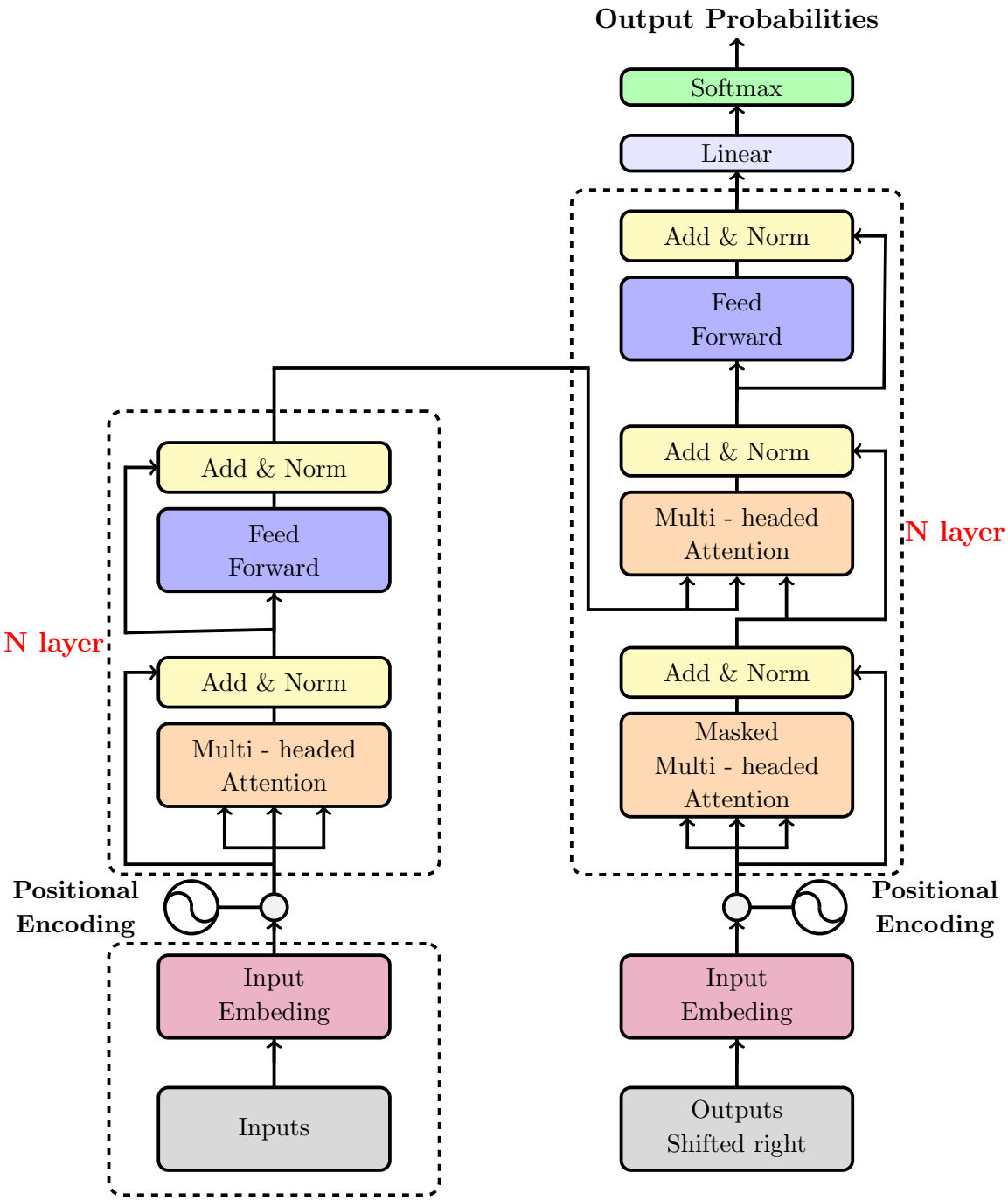


6.5 Cross-Attention

Tương tự như Self-Attettion chỉ khác là query sẽ là token bên còn lại chữ ko nằm trong đầu vào.

Chương 7

Transformer



note hay

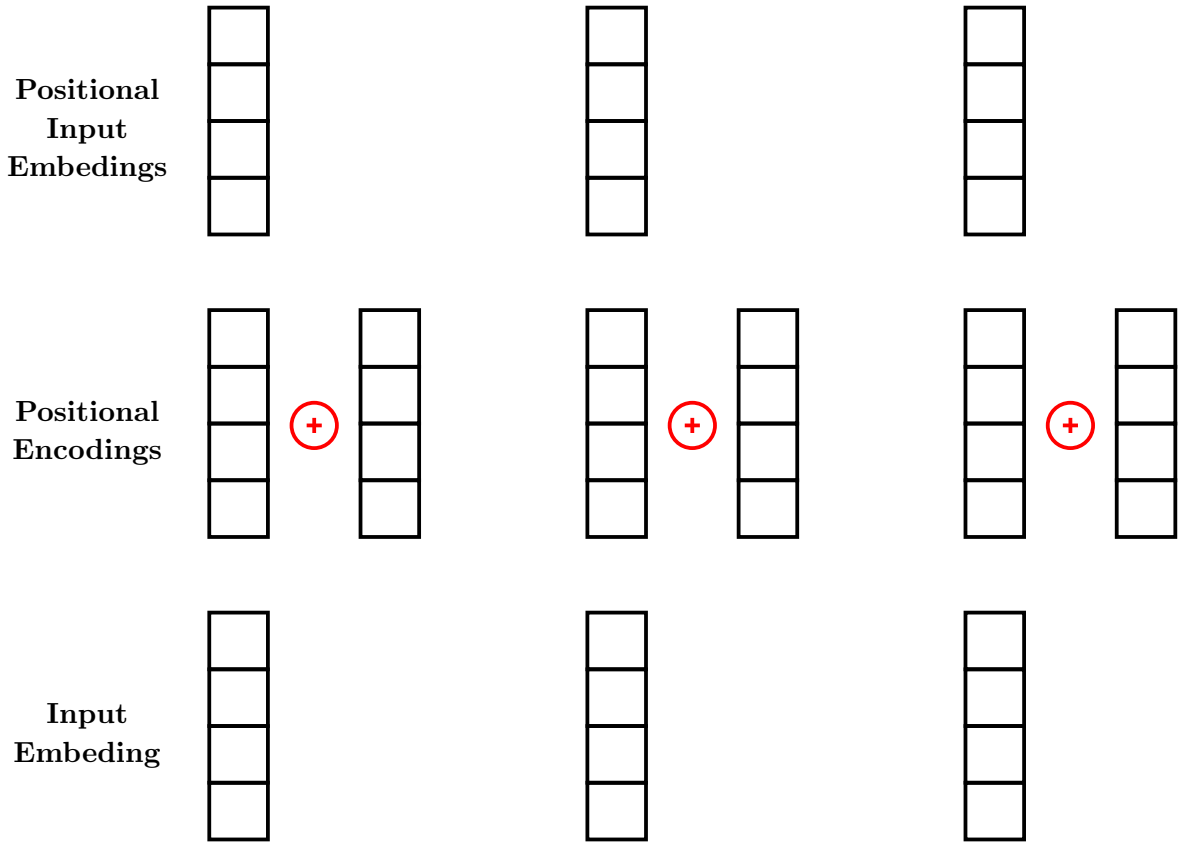
7.1 Transformer Encoder

- **Nhiệm vụ của Encoder :**
  - Học mối tương quan của mỗi từ trong câu với các từ còn lại (self attention).
  - Bổ xung mối quan hệ này vào embedding của từng từ đầu vào.
- **Encoder bao gồm : 6 layer giống nhau**, mỗi layer gồm 2 phần nhỏ :
  - Multi-headed Attention
  - Position-wire fully connected feed-forward network

7.1.1 Positional Encodings

Tương tự như RNN thì mô hình vẫn sử dũng Embedding để mô phỏng từ dưới dạng vector.

Vì model không có hồi quy(recurrence) hay tích chập(convolution) vì thế để đảo bảo thứ tự của các token trong chuỗi, chúng ta sẽ phải bổ sung thêm thông tin về vị trí tương đối và tuyệt đối của các tokens trong chuỗi.



7.1.2 Sinusoid

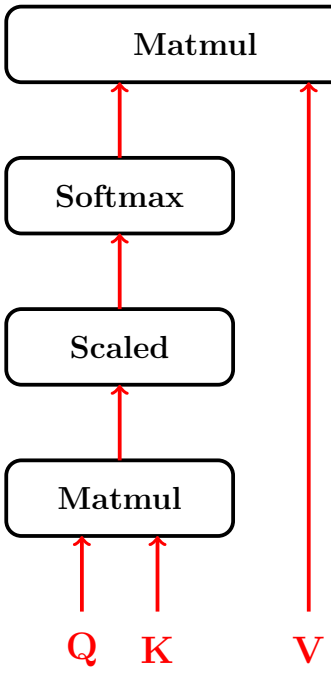
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{1000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{1000^{\frac{2i}{d_{model}}}}\right)$$

	pos = 0	pos = 1	pos = 2	
$d = 0 = 2 * 0$	1.	0.84147096	0.9092974	$i = 0$
$d = 1 = 2 * 0 + 1$	0.	0.5403023	−0.4161468	$i = 0$
$d = 2 = 2 * 1$	1.	0.00999983	0.01999867	$i = 1$
$d = 3 = 2 * 1 + 1$	0.	0.99995	0.9998	$i = 1$

$$PE_{(1,1)} = cos\left(\frac{1}{1000^{\frac{2*0}{4}}}\right) \qquad PE_{(2,2)} = sin\left(\frac{2}{1000^{\frac{2*1}{4}}}\right)$$

7.1.3 Scaled dot product attention

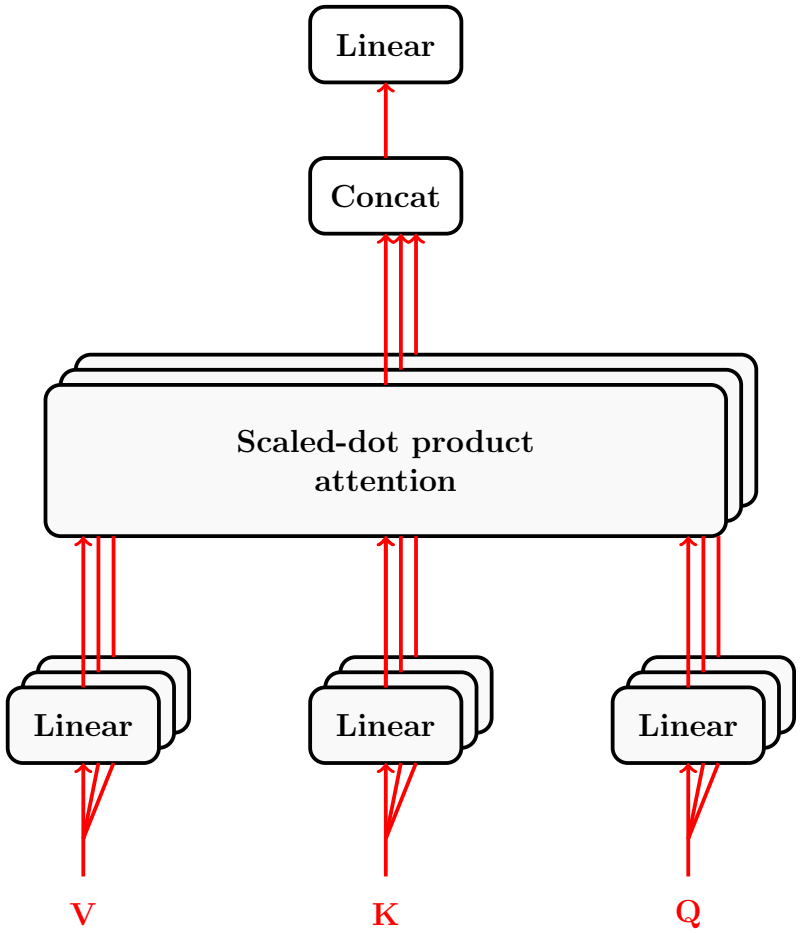


$$\textbf{Attention(Q,K,V)} = softmax\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

7.1.4 Multi-headed Attention

Khi sử dụng duy nhất một Self-Attention, model thực hiện biến đổi tuyến tính trên toàn bộ embedding của các từ, cho nên không khả năng tương tự như Convolution **bóc tách từng phần nhỏ thông tin tại các vị trí nhất định**

Để giải quyết vấn đề này mô hình sẽ sử dụng nhiều Self-Attention, mỗi attention sẽ phụ trách học một phần thông tin của câu.



• Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$Multihead(Q, K, V) = Concat(head_1, \cdots, head_h)W^0$$

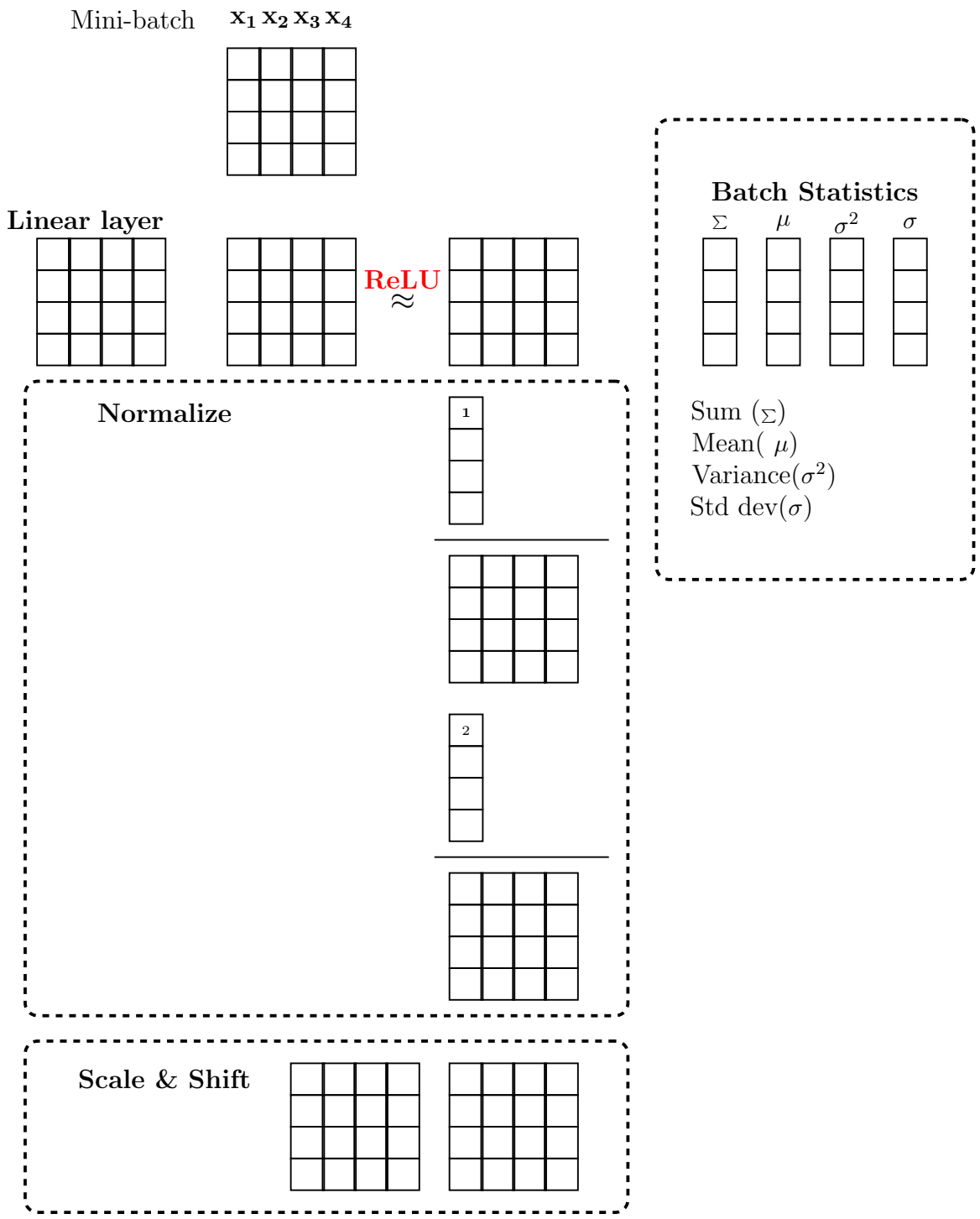
$$Where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

• Where the projections are parameter matrices $W_i^Q \in \mathbf{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbf{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbf{R}^{d_{model} \times d_V}$ and $W^0 \in \mathbf{R}^{hd_v \times d_{model}}$.

• In this work we employ h = 8 parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{model}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

7.1.5 Add & Norm

- **Batch Normalization**



Transformer sử dụng **Layer Norm** để chuẩn hóa dữ liệu nhằm cải tiến khả năng hội tụ của mô hình.

Khác với **Batch Normalization**, Layer Norm không làm trên batch, thay vào đó chuẩn hóa từng mẫu.

Khác với Batch Norm, tất cả các hidden units trong một lớp sử dụng chung một thông số chuẩn hóa (**normalization term - mu và sigma**), những data point khác nhau

sử dụng thông số chuẩn hóa khác nhau.

Trong **LayerNorm**, không có sự liên quan tới độ lớn của minibatch.

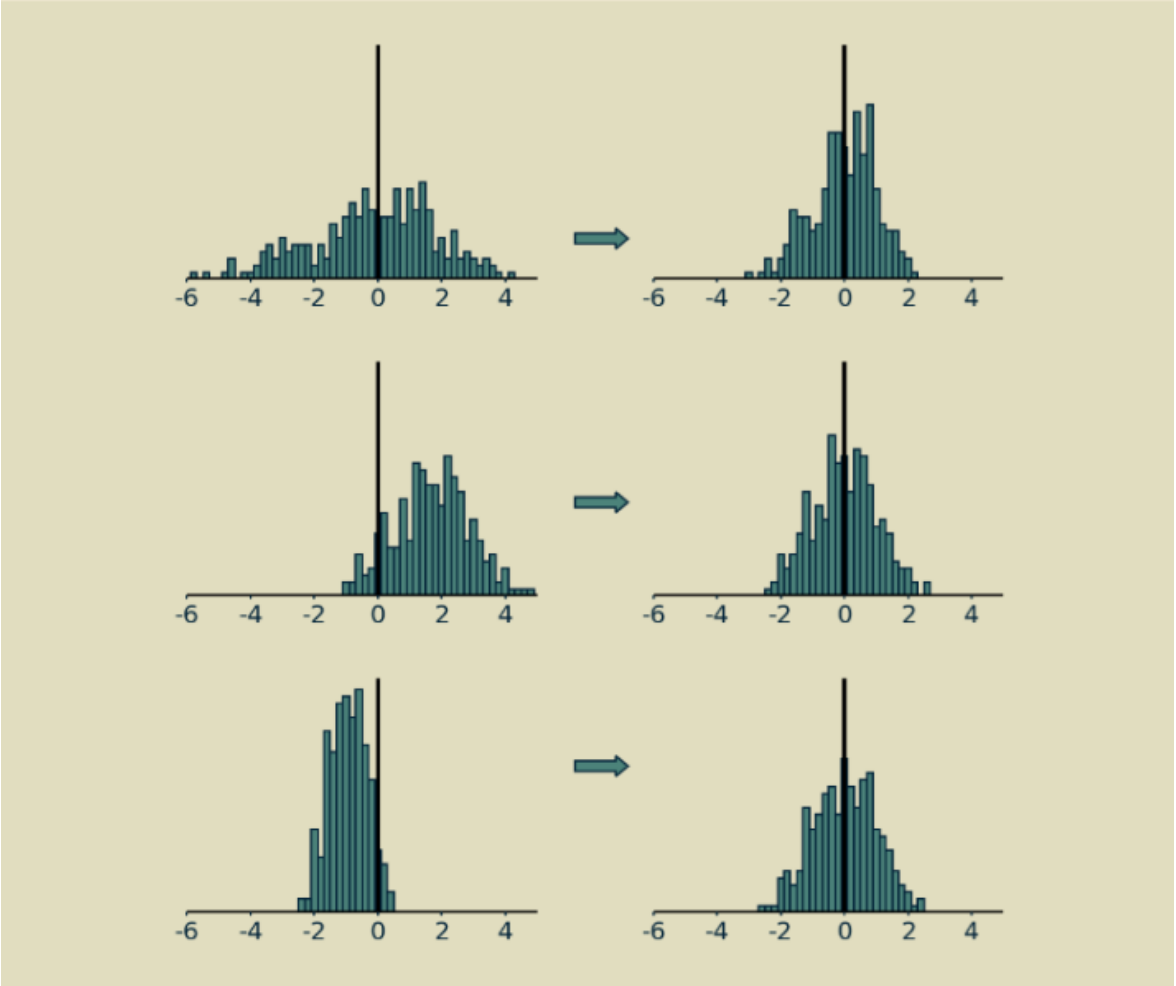
Paper about LayerNorm: đây và đây

$$\mu = \sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_1 - \mu)^2}$$

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \odot \gamma + \beta$$

γ, β are learnable parameters.



7.1.6 Residual Layer(skip connection)

$$x + f(x)$$

Ngoài việc giúp Gradient Flow tốt hơn, cải thiện việc training, nó còn giúp bổ xung thêm thông tin vị trí khi trước đó qua nhiề lớp đã bị mất dần từ layer thấp lên layer cao hơn.

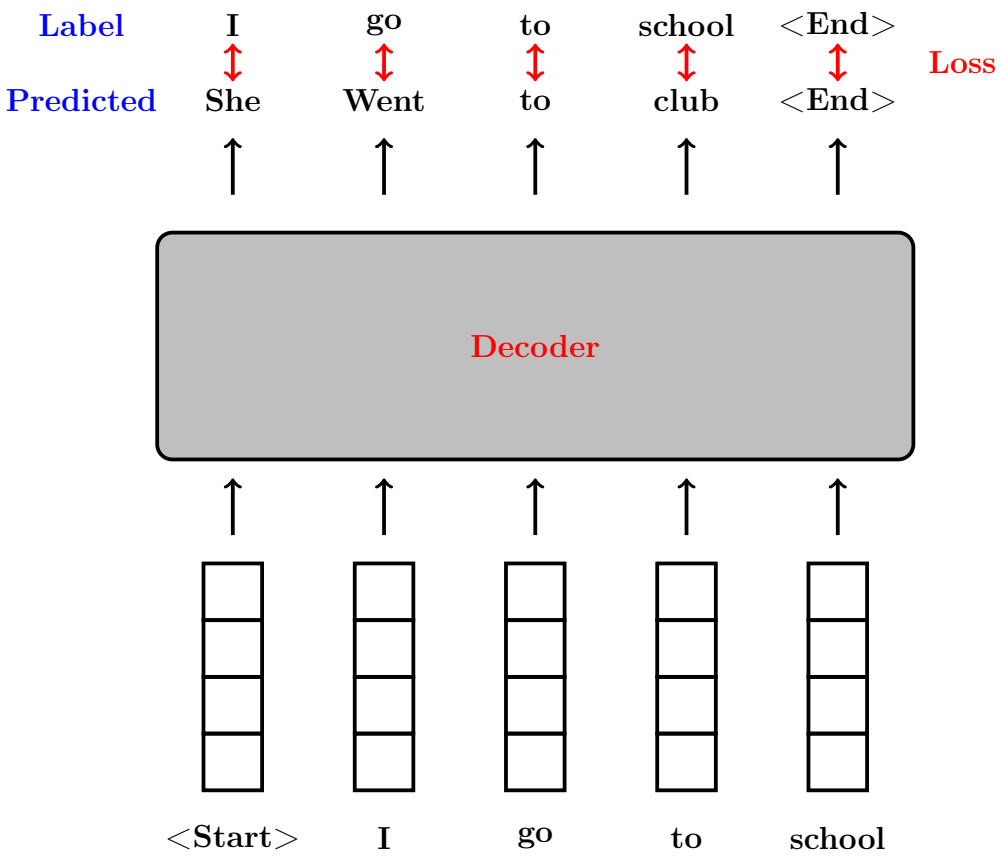
7.1.7 Feed Forward

$$FFN(x, W_1, W_2, b_1, b_2) = max(0, xW_1 + b_1)W_2 + b_2$$

7.2 Transformer Decoder

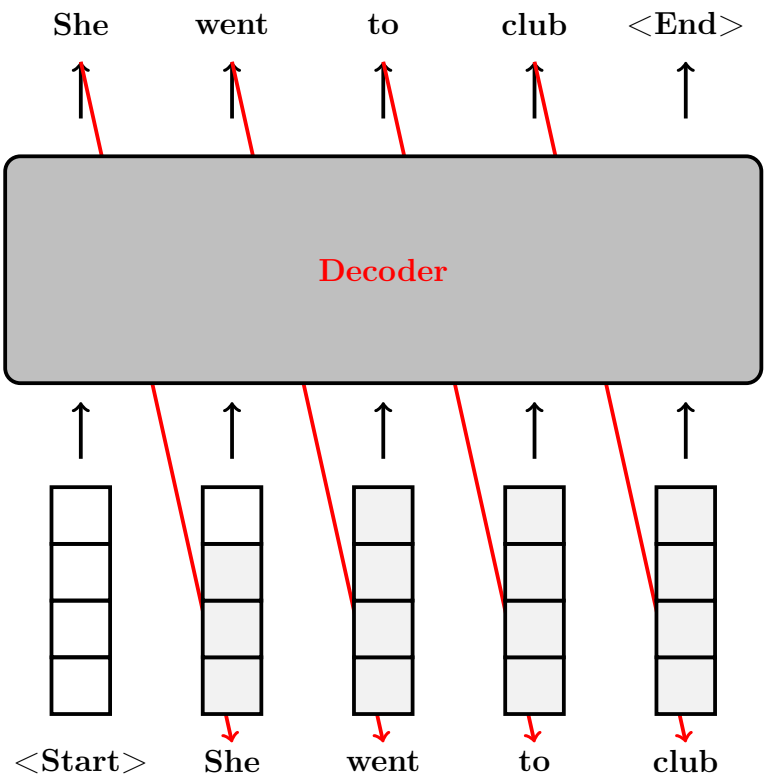
7.2.1 Decoder Training

- Truyền câu mục tiêu vào Decoder để tìm ra câu dự đoán.
- So sánh trực tiếp câu dự đoán với nhãn(Teacher Forcing) để tìm ra giá trị mất mát.



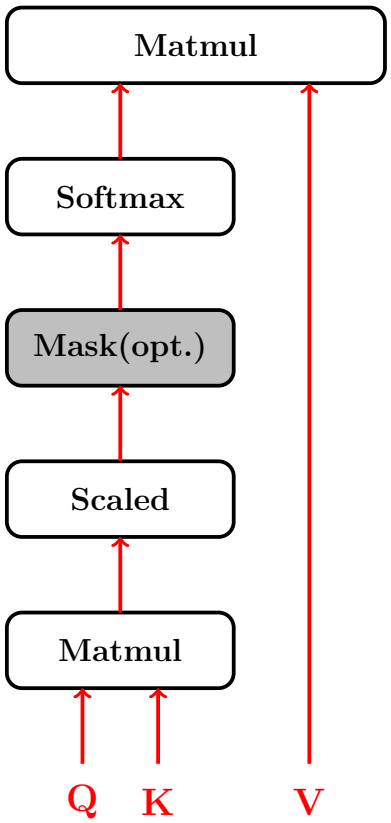
7.2.2 Decoder Inference

- Truyền từng từ vào để dự đoán từ tiếp theo. Nối từ tiếp theo và dự đoán tiếp tục như thế.Quá trình kết thúc khi gặp token <End> hoặc đạt tới max_len.



7.2.3 Look Ahead Mask

• Casual Attention: Cho phép từ kết nối với các từ đã có trước đó và không thể kết nối với các từ phái sau.



	<S>	Tôi	đang	học
<S>	1	5	9	13
Tôi	2	6	10	6
đang	3	7	11	15
học	6	8	8	9

Look Ahead Mask

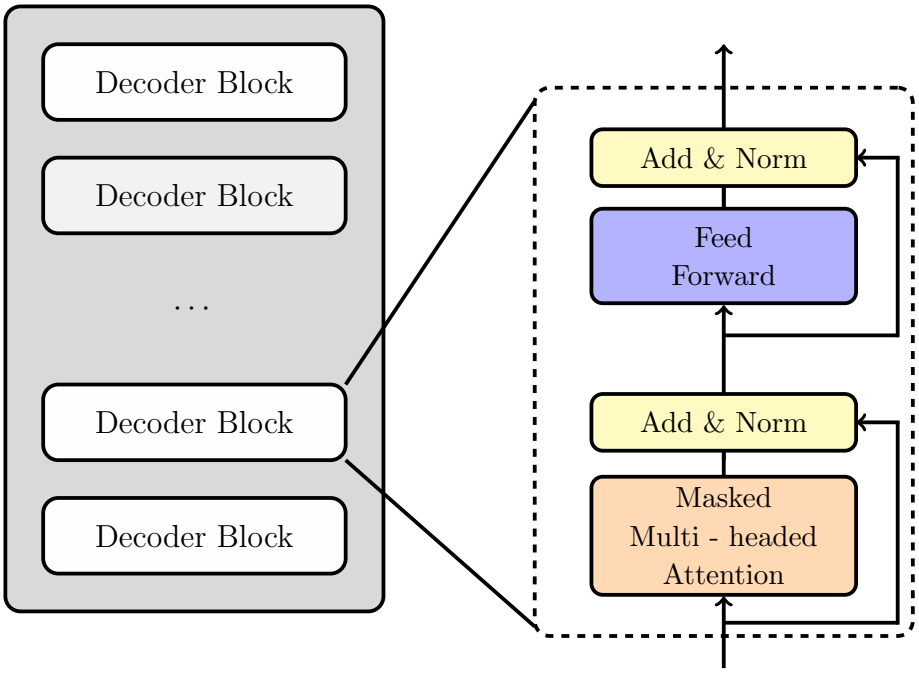
	<S>	tôi	đang	học
<S>	1	-inf	-inf	-inf
Tôi	2	6	-inf	-inf
đang	3	7	11	-inf
học	6	8	8	9

Softmax

	<S>	tôi	đang	học
<S>	1	0	0	0
Tôi	0.018	0.982	0	0
đang	0.001	0.063	0.935	0
học	0.028	0.206	0.206	0.560

7.2.4 GPT

GPT sử dụng Transformer decoder, là auto-regressive model.



Chương 8

BERT(Bidirectional Encoder Representations from Transformers)

• BERT được xây dựng là một pre-trained học từ dữ liệu không nhãn. BERT được pre-trained từ hai bài toán:

  ◦ Dự đoán từ còn thiếu(Masked Sentence)

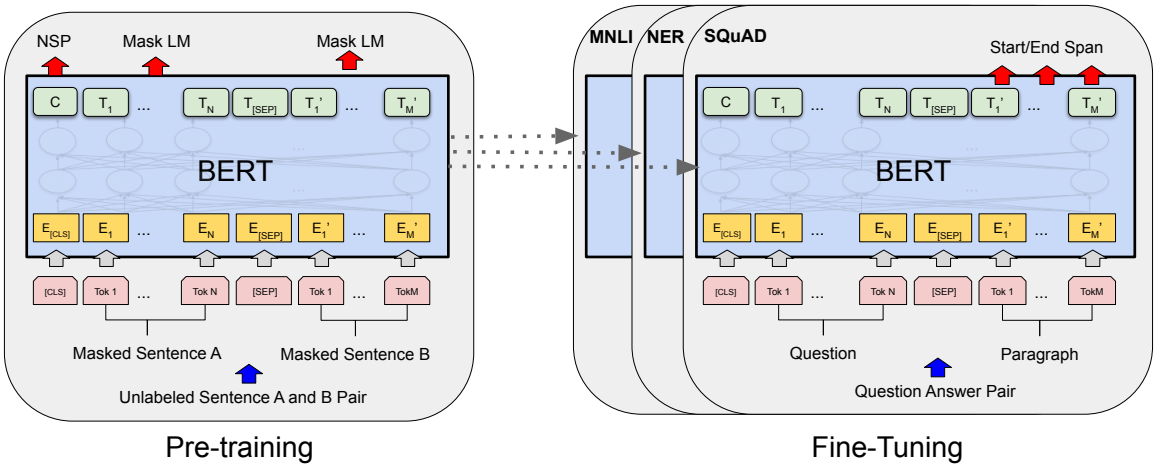
  ◦ Hai câu có hay đi liền với nhau không.

• Từ pre-trained này có thể sử dụng để finetune trên dữ liệu có nhãn.

  ◦ NER - Nhận diện thực thể trong câu

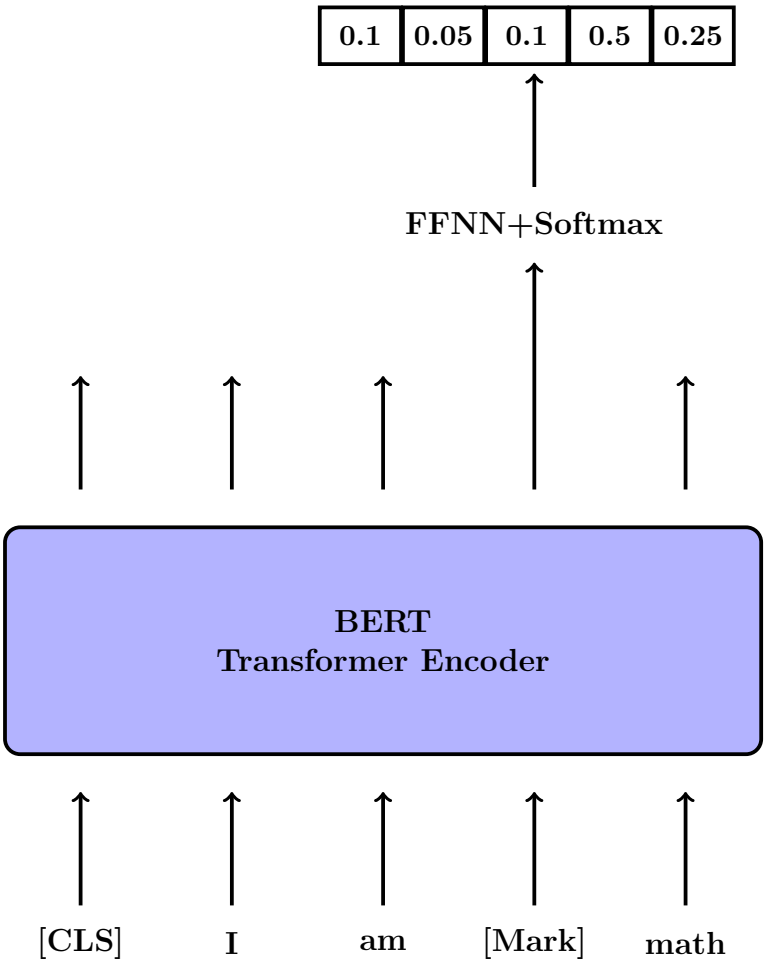
  ◦ SQuAD - trả lời đáp án trong văn bản(token start - end)

  ◦ MNLI - Xác định mối quan hệ của hai câu (có đi liền nhau không, tương phản, bổ trợ, ...)



• Quy trình pre-training BERT:

◦ **Nhiệm vụ 1: Masked Language Model**



- Để train mô hình này, việc ngẫu nhiên che đi (mask) một lượng % tokens trong văn bản sau đó huấn luyện lượng tokens còn lại để dự đoán ra tokens bị che này.

- Cụ thể che đi 15% số lượng WordPiece Tokens ở mỗi chuỗi đầu vào một cách ngẫu nhiên.

- Trong 15% này:

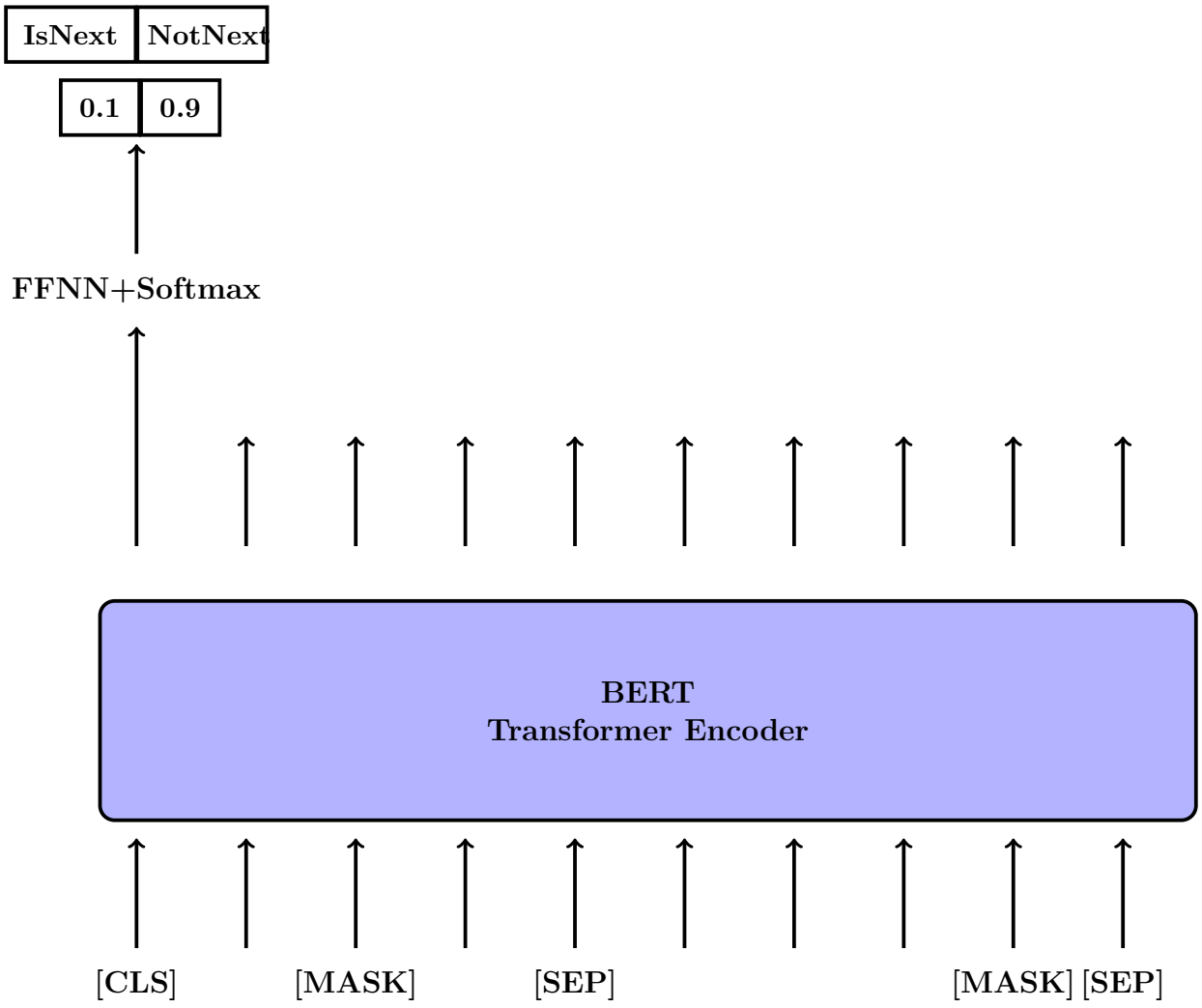
+ thay thế 80% bằng token [MASK]

+ thay thế 10% bằng token ngẫu nhiên

+ giữ nguyên 10% còn lại.

- Cách làm này là do token [MASK] chỉ Sat hiện khi **train pre-trained** và không xuất hiện khi chúng ta **fine-tune**.

◦ **Nhiệm vụ 2: Dự đoán câu tiếp theo**



- Pre-train sẽ được train cho bài toán phân loại hai câu có đi liền với nhau hay không.

- Dữ liệu được train sẽ bao gồm các cặp câu A-B trong đó:

+ 50% câu B đi theo sau câu A với nhãn dự đoán là IsNext

+ 50% câu B ngẫu nhiên từ trong ngữ liệu với nhãn dự đoán là NotNext

- Hai câu này được thêm những Token để trở thành dạng sau:

[CLS] A [SEP] B [SEP]

- Trong đó:

+ [CLS] là token sentence-level classification ở đầu. Token này sẽ thực hiện attention với tất cả các từ trong câu. Thực tế token này sau khi đi qua BERT thì vector của nó có thể sử dụng cho bài toán phân loại câu.

+ [SEP] là token để chia cắt các chuỗi câu. Ví dụ bài toán dịch máy thì [SEP] chia câu được dịch và câu dịch.

• Một số Token đặc biệt:

Token	Type	Describe
[CLS]	phân loại	BERT chèn token này vào đầu mỗi chuỗi đầu vào.n Trong các nhiệm vụ liên quan đến phân loại hoặc biểu diễn toàn bộ chuỗi đầu vào(như phân tích cảm xúc), trạng thái ẩn cuosi cùng tương ứng với token này được sử dụng làm biểu diễn tổng hợp cho các nhiệm vụ phân loại.
[SEP]	Phân tách	Token "phân tách"được sử dụng trong BERT để phân tách các câu hoặc phân tách các đoạn khác nhau trong cùng một đầu vào. Nó sử dụng cho các nhiệm vụ liên quan đến nhiều đầu vào như trả lời câu hỏi(nơi mà mô hình cần phân biệt giữa câu hỏi và đoạn văn) hoặc các nhiệm vụ cặp câu(như phát hiện cụm từ đồng nghĩa).
[PAD]	Đệm	Token "đệm"này được sử dụng để lấp đầy các chuỗi để tất cả các đầu vào trong một batch có cùng độ dài. Việc đệm là cần thiết vì kiến trúc Transformer(mà BERT dựa trên) yêu cầu đầu vào phải có cùng kích thước(size).
[MASK]	Che	Được sử dụng chủ yếu trongt quá trình tiền huấn luyện của BERT trên nhiệm vụ MLM (Mask Language Model) nơi các token ngẫu nhiên trong đầu vào được thay thế bằng token này. BERT sau đó được huấn luyện để dự đoán giá trị gốc của các từ bị chem dựa chỉ vào ngữ cảnh của chúng. Điều này giúp mô hình hiểu ngôn ngữ tốt hơn và cải thiện khả nặng dự đoán các từ bị thiếu.
[UNK]	Không biết	Nó được sử dụng khi một từ không được tìm thấy trong vocal của BERT. Vì BERT sử dụng từ vựng cố định, bất kì từ nào không tìm thấy sẽ thay thế bằng token [UNK]

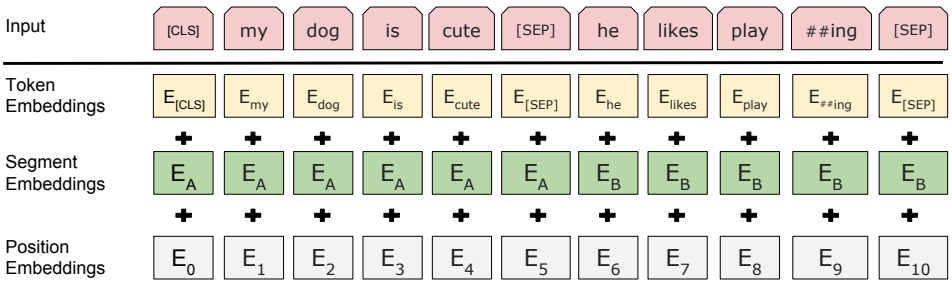
8.1 Tokenizing and reperesentation

• Quy trình tiếp theo: Câu sẽ được biểu diễn lại:

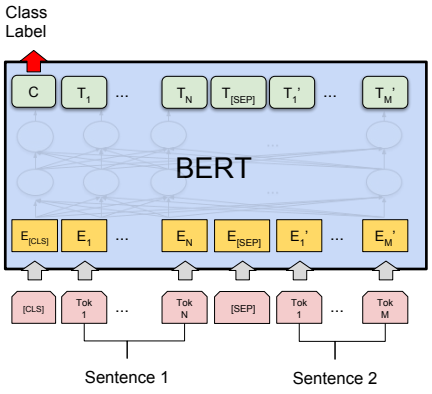
◦ Qua Token Embeddings để tách token

◦ Qua Segment Embeddings để cập nhật thông tin thuộc chuỗi nào. Ví dụ token thuộc chuỗi A sẽ được cộng thêm giá trị E_A có thể bằng 0 còn token thuộc chuỗi B sẽ được cộng thêm giá trị E_B có thể bằng 1.

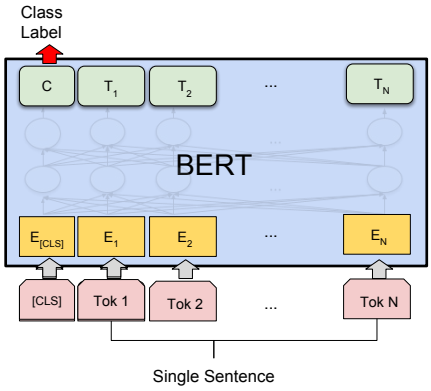
◦ Bổ sung thêm Embdding vị trí cho từng token



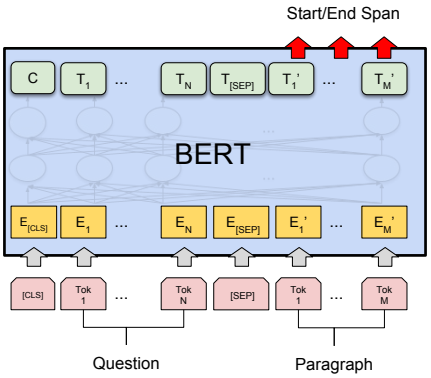
8.2 Fine-tuning BERT on Difference Tasks



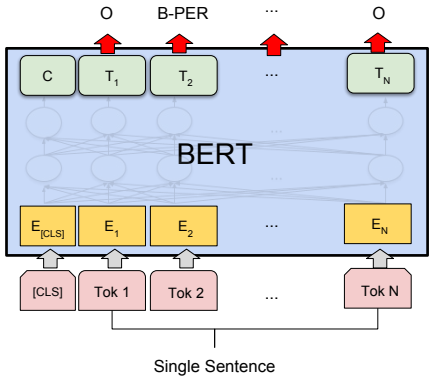
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



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

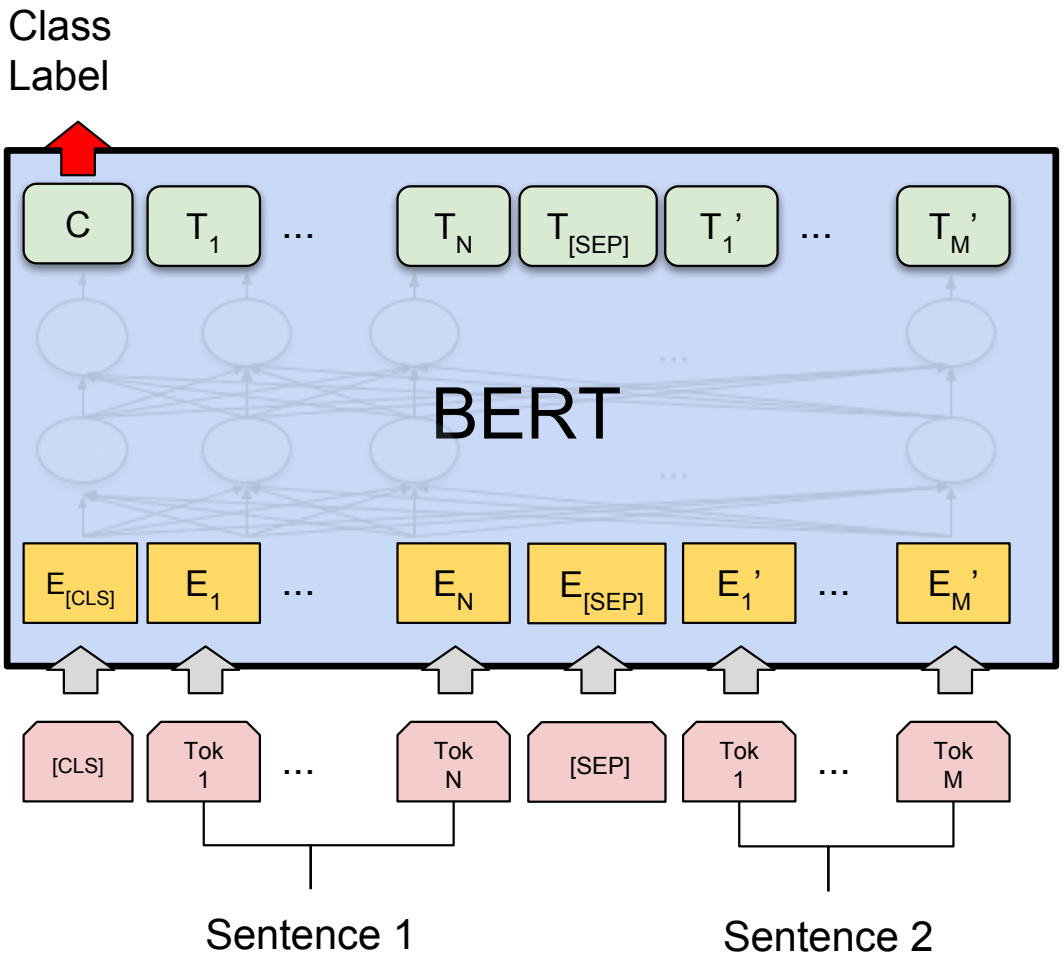


(c) Question Answering Tasks: SQuAD v1.1



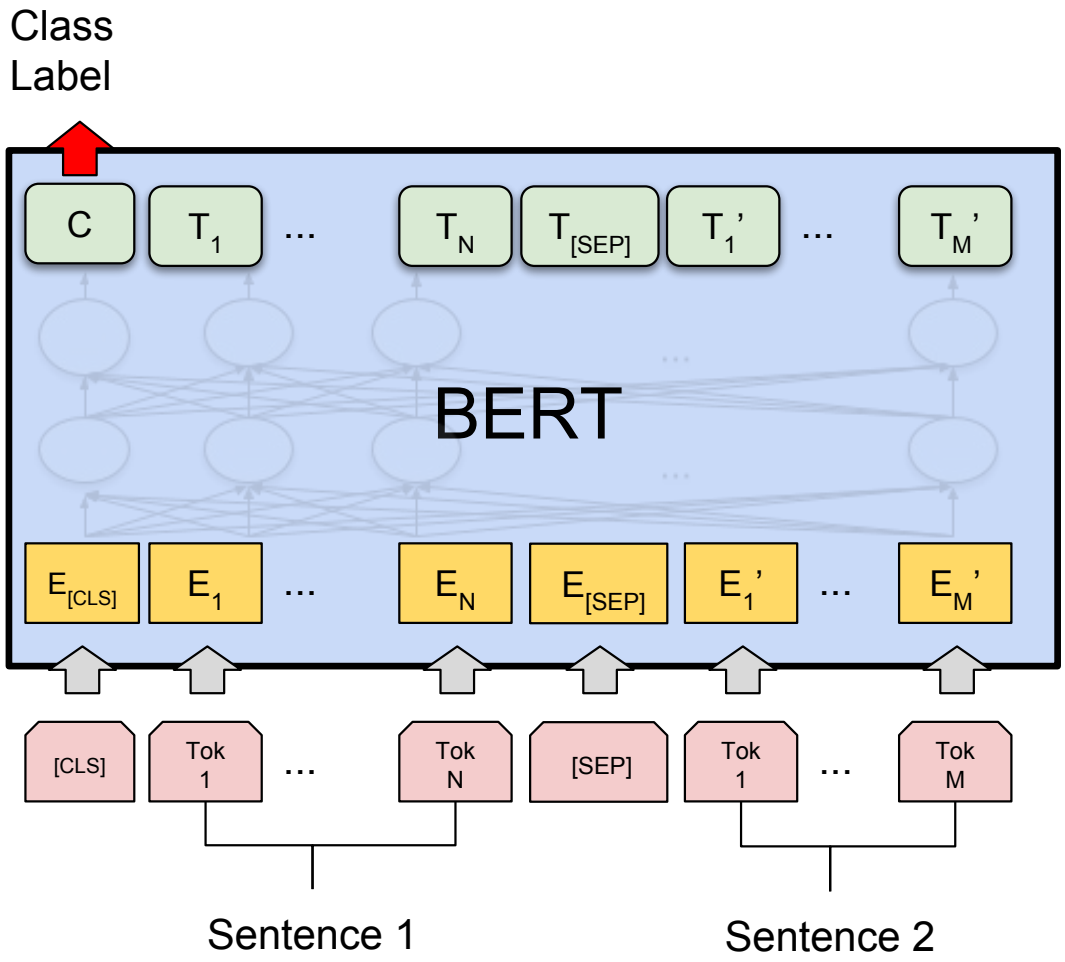
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

8.2.1 Fine-tune on the sentence pair problem



MNLI	Multi Natural Language Interface
QQP	Quara Question Pairs
QNLI	Question Natural Language Interface
STS-B	The Semantic Textual Similarity Benchmark
MRPC	Microsoft Reseach Paraphrase Corpus
RTE	Recognizing Textual Entailment
SWAG	Situations With Adversarial Generation

8.2.2 Fine-Tune on the sentence pair problem



Input :

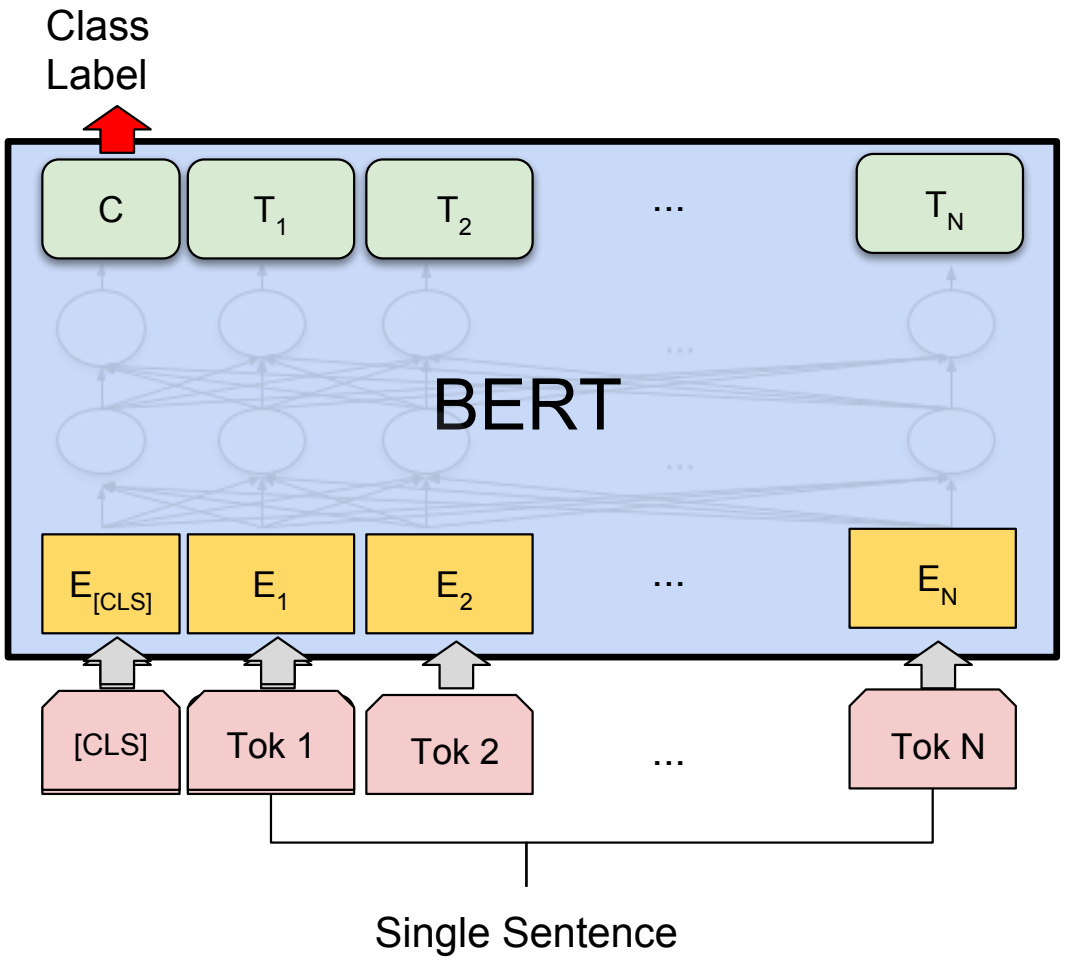
[CLS] $Token_1\ Token_2 \cdots Token_n$ **[SEP]** $Token'_1\ Token'_2 \cdots Token'_m$ **[SEP]**

Label: 0-5

Explain:

Sentence 1	sentence 2	Label
A plane is taking off	An airplane is taking off	5.00
Three men are playing chess	Two men are playing chess	2.60
Severe Gales As Storm Clodagh Hits Britain	Merkel pledges NATO solidarity with Latvia	0.00
A man is playing a large flute	A man is playing a flute	3.80
China, India vow to further bilateral ties	China Scrambles to Reassure Jittery Stock Traders	0.00
A man is playing the cello	A man seated is playing the cello	4.25

8.2.3 Fine-tune on the sentence classification problem



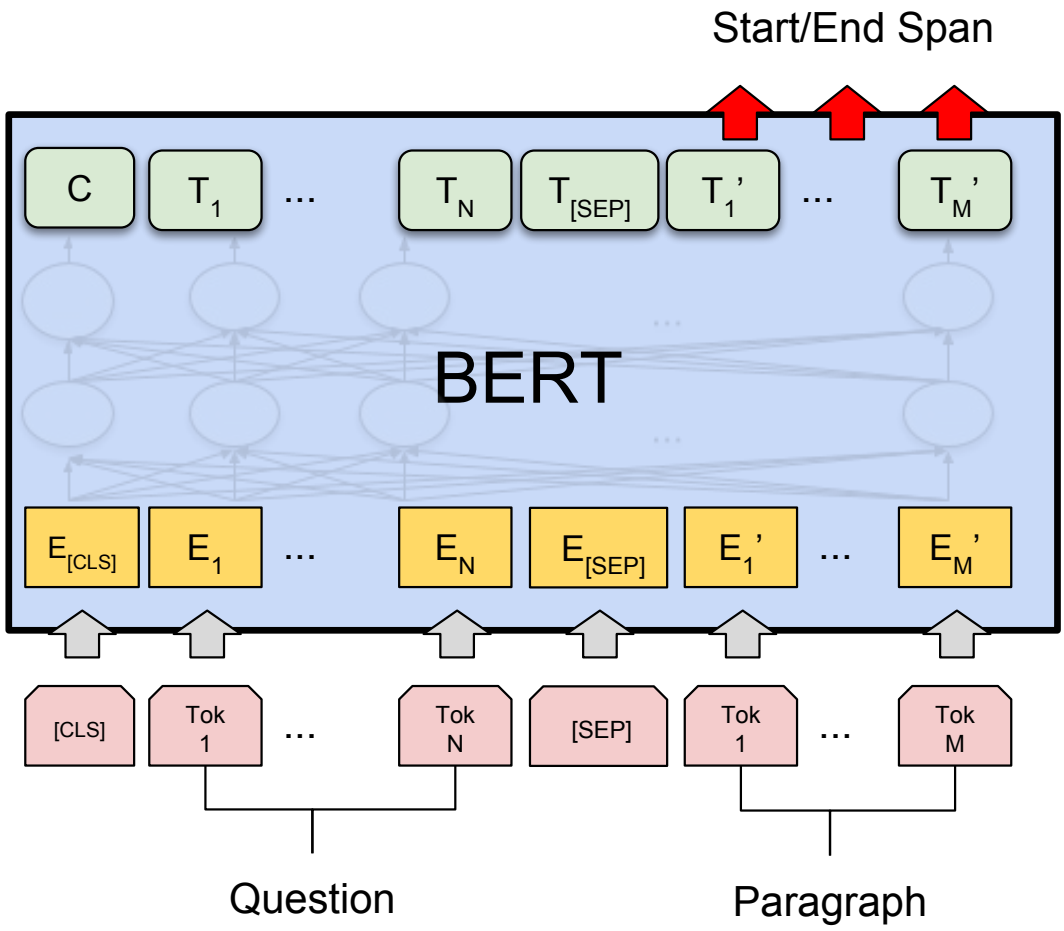
Input: Single Sentence

Label: Positive (1), negative (0)

Explain:

Sentence	label
The data shows a significant increase in sales over the last quarter	1
The research data reveals promising trends in renewable energy usage	1
The survey data highlights concerns regarding product qualit	0
The data reveals a drop in revenue compared to last year	0

8.2.4 Fine-tune on the question answering problem



Input:

Question: Lợi ích chính của việc sử dụng nguồn năng lượng tái tạo là gì?

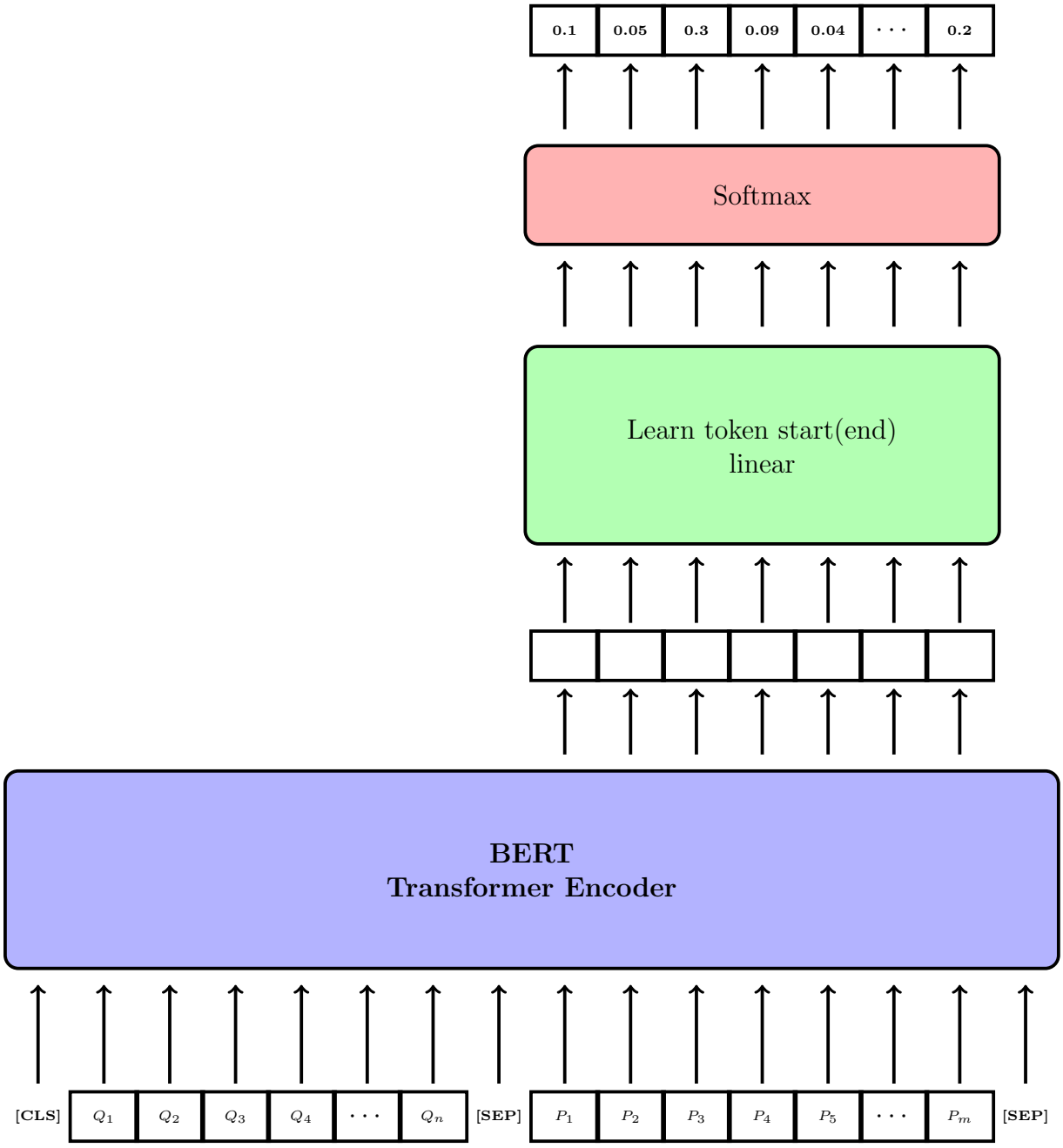
Paragraph: Nguồn năng lượng tái tạo, như năng lượng mặt trời, gió và thủy điện, mang lại nhiều lợi ích chính. Chúng giúp giảm đáng kể khí thải nhà kính, góp phần chống biến đổi khí hậu và cải thiện chất lượng không khí. Ngoài ra, chúng cung cấp giải pháp bền vững lâu dài vì những nguồn này được tái tạo tự nhiên, giảm thiểu sự phụ thuộc vào nhiên liệu hóa thạch có hạn. Hơn nữa, đầu tư vào năng lượng tái tạo tạo ra việc làm trong sản xuất, lắp đặt và bảo trì, thúc đẩy kinh tế địa phương. Cuối cùng, chúng nâng cao an ninh năng lượng bằng cách đa dạng hóa nguồn cung và giảm thiểu rủi ro về biến động giá cả của thị trường nhiên liệu hóa thạch.

Output:

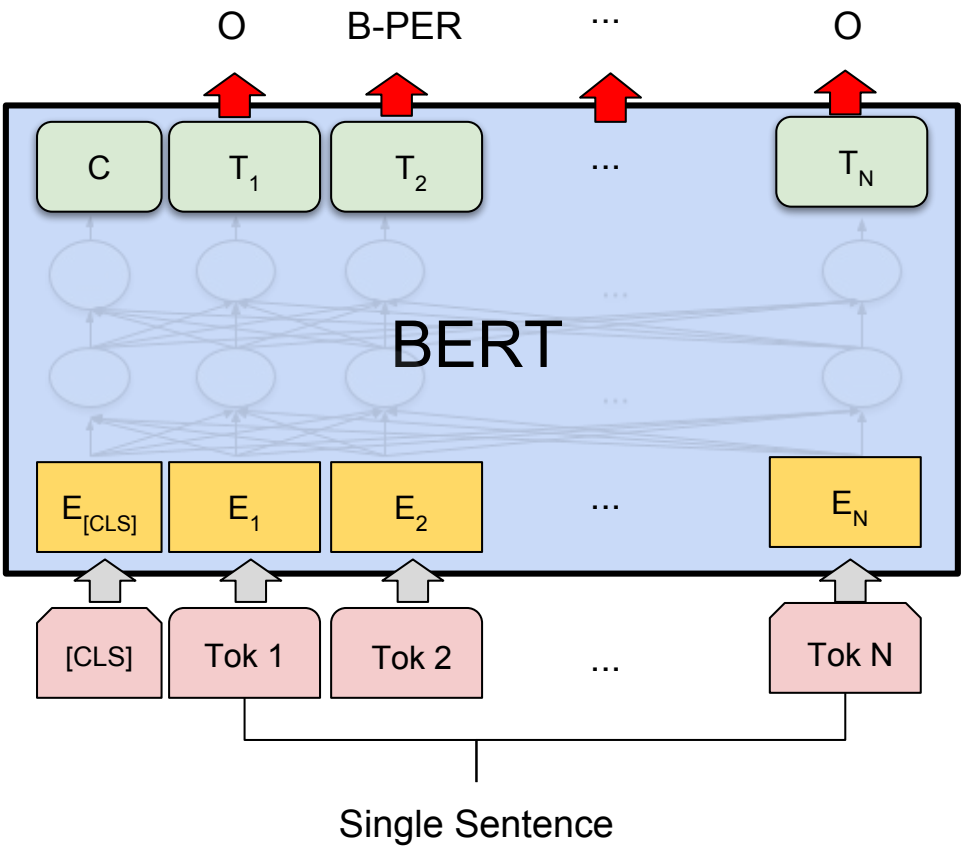
start = 21

end = 43

→ Answer:Chúng giúp giảm đáng kể khí thải nhà kính, góp phần chống biến đổi khí hậu và cải thiện chất lượng không khí


0.1
0.05
0.3
0.09
0.04
· · ·
0.2
Softmax
Learn token start(end)
linear
BERT
Transformer Encoder
[CLS]
Q_1
Q_2
Q_3
Q_4
· · ·
Q_n
[SEP]
P_1
P_2
P_3
P_4
P_5
· · ·
P_m
[SEP]

8.2.5 Fine-tune on the token classification problem



Bài toán nhận diện POS Tag (Post-of-speech tagging)

Bài toán nhận diện thực thể NER(Named Entity Recognition):

Example:

Input:

[CLS] The capital of Viet Nam is Ha Noi [SEP].

Output:

o o o B-location I-loction o B-location I-location.

Explain:

o: outside name entity

Với mỗi entity kiểu T, ta có hai nhãn B-T và I-T. B-T là begin type T, I-T là inside type T.

Chương 9

T5(The Text-To-Text Transfer Transformer)

Chương 10

Transformer Avanced

Chương 11

SMT Statitic Machine Translate

Chương 12

LLM Large Language Models

Phần IV

Application

Chương 1

Truy vấn thông tin (Information Retrieval)

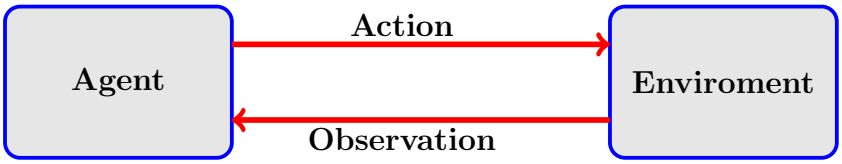
Chương 2

RAG (Retrieval Augmented Generation)

Chương 3

AI Agent

3.1 Reinforcement Learning

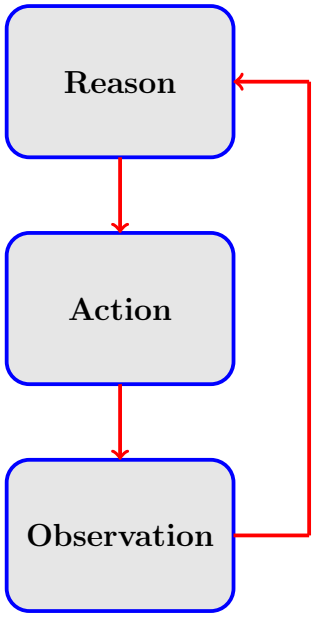


- Agent tương tác với môi trường và nhận về các observation.
- Ví dụ: Agent làm sai sẽ bị phạt, Agent làm đúng sẽ được cộng điểm.

3.2 React

React = Reason + action

- React là một Framework giúp các tác nhân (Agents) giải quyết vấn đề một cách hiệu quả bằng cách kết hợp Lý luận(Reason) và hành động(Act) theo từng bước.



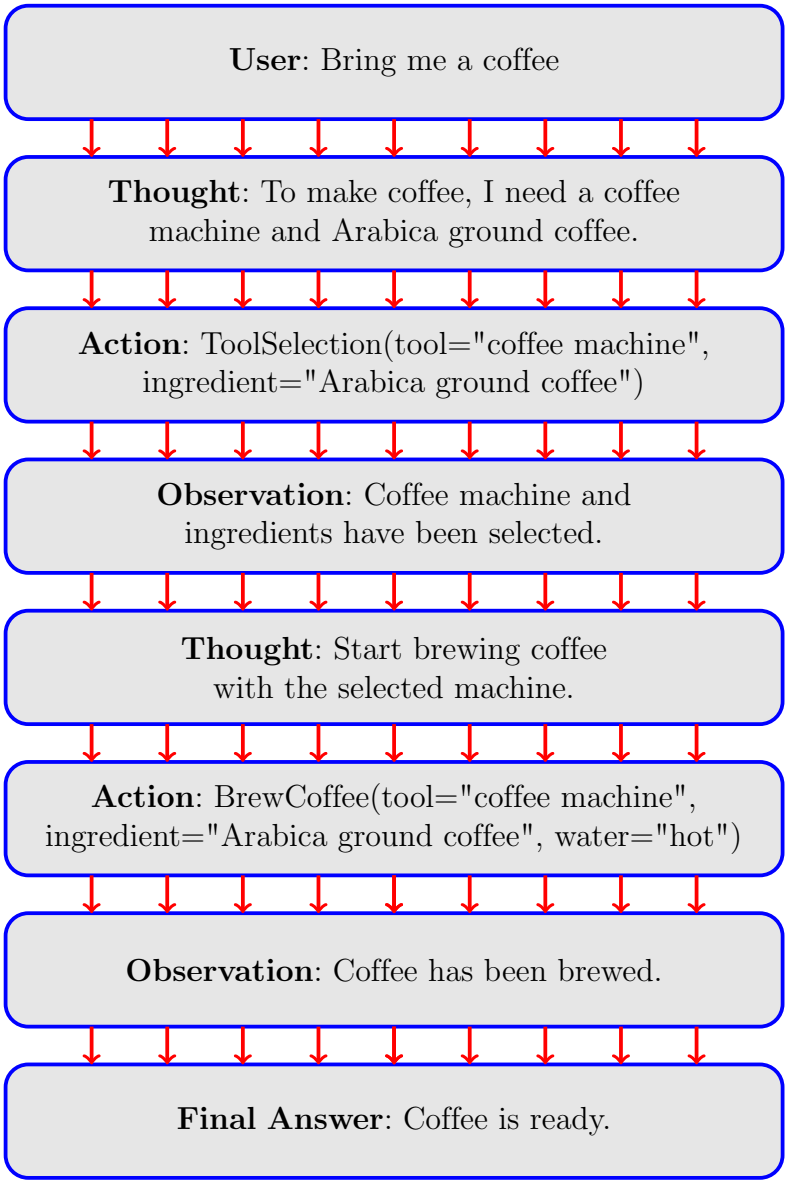
- Tác nhân suy nghĩ logic để đưa ra các giả định hoặc phán đoán đựa trên thông tin hiện có. Lý luận này dựa vào các bước phân tích vấn để để tìm ra hướng giải quyết phù hợp.

- Sau mỗi bước suy nghĩ, tác nhân thực hiện một hành động cụ thể. Hành động này cung cấp thêm thông tin hoặc giải pháp để tiến tới bước tiếp theo.

- Mỗi hành động thực hiện sẽ mang lại kết quả hoặc phản hồi. Kết quả này được sử dụng để bổ sung vào quá trình lý luận cho các bước tiếp theo.

- Chu trình Lý luận ⇒ hành động ⇒ Quan sát được lặp lại cho đến khi đặt được mục tiêu cuối cùng. Tác nhân không dừng lại cho đến khi tìm được câu trả lời hoặc giải pháp thỏa đáng.

Một ví dụ nhỏ để hình dung được:



3.3 Prompt

3.3.1 Prompt Element

- Yêu cầu (Instruction): rõ ràng, xúc tích.
- Ngữ cảnh (Context): đầy đủ thông tin.
- Dữ liệu đầu vào (Input Data)
- Định dạng đầu ra (Output Indicator): chặt chẽ để tương thích với ứng dụng(ví dụ csv, json, ...)

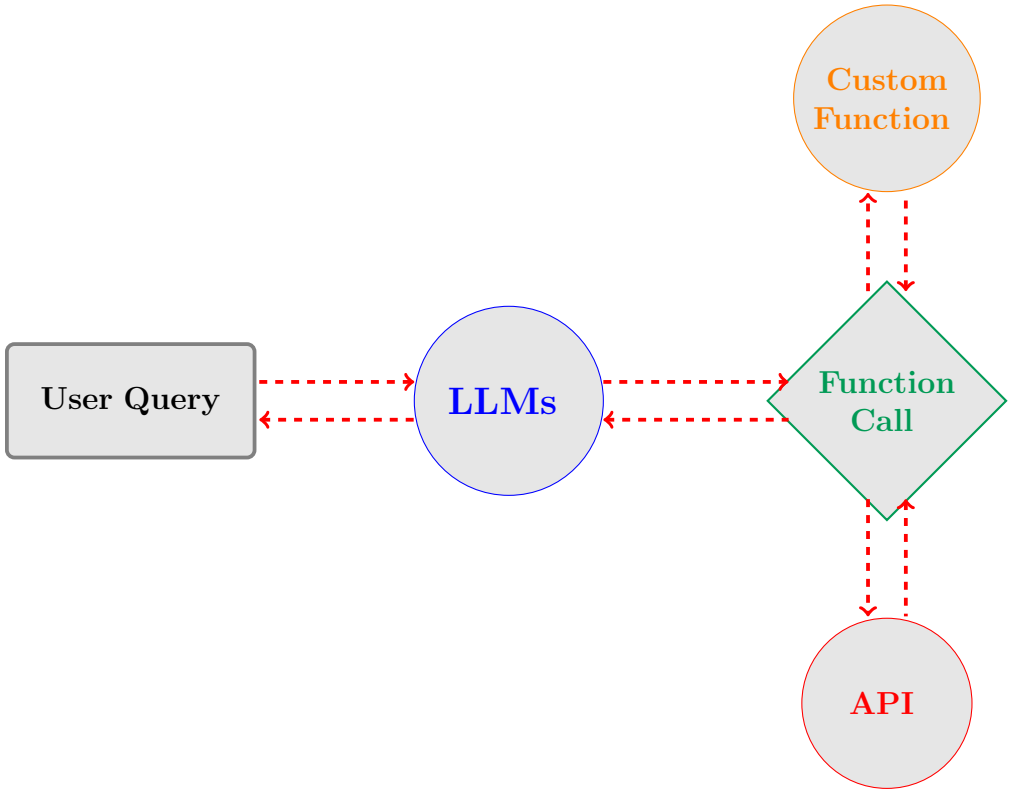
∗ **Tip:** Thêm ### ví dụ: ### Yêu cầu ### → khớp với mô hình được đào tạo.

3.4 Funtion calling

- Gọi hàm (Function Calling) cho phép các Mô hình Ngôn ngữ Lớn (LLMs) tương tác với các công cụ, API và hàm bên ngoài dựa trên đầu vào của người dùng. Thay vì chỉ tạo ra văn bản thuần túy, LLM có thể nhận biết khi nào cần thực hiện một hành động cụ thể và sau đó yêu cầu một hàm bên ngoài xử lý tác vụ đó.
- Với khả năng này, người dùng có thể tương tác với các hệ thống phức tạp chỉ bằng ngôn ngữ tự nhiên, trong khi LLM đảm nhiệm việc thực thi các chức năng nền tảng. Điều này giúp mô hình không chỉ tạo ra nội dung mà còn giải quyết các vấn đề thực tế một cách hiệu quả.
- Ví dụ, nếu người dùng hỏi về thời tiết, thay vì chỉ trả lời chung chung, mô hình có thể gọi API thời tiết để lấy dữ liệu theo thời gian thực. Hơn thế nữa, nếu dự báo có mưa, LLM thậm chí có thể nhắc bạn mang theo ô khi ra ngoài, giúp trải nghiệm tương tác trở nên thông minh và hữu ích hơn.

a

How Function Calling Works:



- **Truy vấn của người dùng (User Query)**: Quá trình bắt đầu khi người dùng đặt câu hỏi hoặc yêu cầu một hành động (ví dụ: "Có những khách hàng tiềm năng nào trong CRM của tôi?"hoặc "Kiểm tra xem sản phẩm X còn hàng không").
- **Xử lý bởi LLM (LLM Processing)**: LLM phân tích truy vấn và nhận ra rằng cần có dữ liệu bên ngoài hoặc hành động cụ thể để đáp ứng yêu cầu. Ví dụ:
  - Nếu người dùng hỏi về khách hàng tiềm năng trong CRM, LLM xác định rằng cần lấy dữ liệu thời gian thực.
  - Nếu người dùng muốn kiểm tra tồn kho sản phẩm, LLM sẽ kích hoạt một truy vấn vào cơ sở dữ liệu.
- **Quyết định gọi hàm (Function Call Decision)**: LLM quyết định thực hiện một lệnh gọi hàm, có thể là:
  - **Gọi API (API Calls)**: Kết nối với các dịch vụ bên ngoài (ví dụ: API của CRM để lấy dữ liệu khách hàng tiềm năng theo thời gian thực từ Salesforce).
  - **Hàm tùy chỉnh (Custom Functions)**: Truy cập vào các công cụ hoặc cơ sở dữ liệu nội bộ (ví dụ: Kiểm tra mức tồn kho trong hệ thống quản lý kho).
- **Truy xuất dữ liệu (Data Retrieval)**: Hàm được gọi sẽ tìm nạp dữ liệu cần thiết (ví dụ: danh sách khách hàng tiềm năng từ API của Salesforce hoặc thông tin tồn kho từ cơ sở dữ liệu kho hàng).
- **Tích hợp dữ liệu (Data Integration)**: Dữ liệu thu thập được gửi lại cho LLM, sau đó mô hình xử lý và tạo ra một phản hồi có ngữ cảnh và chính xác cho người dùng.

Example of how it works:

Step 1: Define available functions (schemas for LLM)

```
function_definitions = [
  {
    "name": "book_flight",
    "description": "Book a flight ticket between two cities",
    "parameters": {
      "type": "object",
      "properties": {
        "from_city": {
          "type": "string",
          "description": "Departure city"
        },
        "to_city": {
          "type": "string",
          "description": "Destination city"
        },
        "date": {
          "type": "string",
          "description": "Flight date in YYYY-MM-DD format"
        }
      },
      "required": ["from_city", "to_city", "date"]
    }
  }
]
```

Step 2: User query input

```
user_query = "Book a flight from New York to Los Angeles on
   March 15."
```

Step 3: Simulate LLM extracting function_call output In real use, GPT-4,OPEN AI, LLama, Gemini,etc would return this automatically

```
function_call = {
  "name": "book_flight",
  "arguments": {
    "from_city": "New York",
    "to_city": "Los Angeles",
    "date": "2025-03-15"
  }
}
```

Step 4: Define the actual function to be called

```
def book_flight(from_city, to_city, date):
  return f"Flight booked from {from_city} to {to_city} on {date
     }."
```

Step 5: Execute the function using parsed arguments from LLM

```
if function_call["name"] == "book_flight":
  result = book_flight(**function_call["arguments"])
  print(result)
```

Result:

```
  Flight booked from New York to Los Angeles on 2025-03-15.
```

Có thể sinh trực tiếp function_definitions tại OPEN AI Prompts

3.5 Định nghĩa về AI Agent

AI Agent là sự kết hợp chặt chẽ giữa các thành phần như lý luận (Reason), hành động (Action), phản hồi (Observation), và khả năng gọi hàm (Function Calling) để thực thi nhiệm vụ một cách tự động và thông minh.

- AI Agent hoạt động theo một chu trình lặp gồm: Lý luận ⇒ Hành động ⇒ Quan sát, tương tự như mô hình React đã trình bày ở phần trước.
- Trong mỗi bước hành động, AI Agent có thể thực hiện Function Calling — gọi đến các hàm hoặc API bên ngoài để thu thập dữ liệu, thực hiện tác vụ hoặc truy xuất kết quả.
- Việc sử dụng Function Calling giúp AI Agent không chỉ xử lý logic nội tại, mà còn tương tác hiệu quả với môi trường bên ngoài (như hệ thống dữ liệu, công cụ hỗ trợ, hoặc API).
- Các function được gọi có thể là:
  - API từ hệ thống bên ngoài (ví dụ: tra cứu thời tiết, thông tin kho hàng).
  - Hàm tùy chỉnh trong hệ thống nội bộ (ví dụ: xử lý CRM, báo cáo doanh thu).
- AI Agent thường được xây dựng dưới dạng một hệ thống có khả năng:
  - **Lập kế hoạch (Planner)**: xác định các bước cần thực hiện để hoàn thành mục tiêu.
  - **Thực thi (Executor)**: gọi các hàm tương ứng để xử lý từng bước.
  - **Ghi nhớ (Memory)**: lưu thông tin, quan sát và kết quả phục vụ cho các bước sau.
  - **Lặp lại (Loop)**: tiếp tục quá trình suy nghĩ – hành động – quan sát cho đến khi đạt mục tiêu cuối cùng.
- Nhờ đó, AI Agent có khả năng tự động hóa quy trình, ra quyết định thông minh và phản ứng linh hoạt trước nhiều tình huống phức tạp trong thế giới thực.