TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO MÔN HỌC
XỬ LÝ NGÔN NGỮ TỰ NHIÊN

HỌ MÔ HÌNH NGÔN NGỮ BERT VÀ ỨNG DỤNG

Giảng viên hướng dẫn: **TS. Nguyễn Kiêm Hiếu**

Học viên thực hiện: **Nguyễn Đức Thắng**

Mã số học viên: **20166769**

HÀ NỘI, 10/2020

Mục lục

Lời mở đầu

Trong những năm gần đây, trí tuệ nhân tạo ngày càng phát triển và đi vào cuộc sống hàng ngày. Trong đó, xử lý ngôn ngữ tự nhiện (NLP) là một nhánh của trí tuệ nhân tạo tập trung vào các ứng dụng trên văn bản và ngôn ngữ, âm thanh. Trong trí tuệ nhân tạo thì xử lý ngôn ngữ tự nhiên là một trong những phần khó, vì để giúp máy hiểu được ngôn ngữ không phải là đơn giản. Các bài toán của xử lý ngôn ngữ bao gồm: chuyển văn bản sang giọng nói và ngược lại, phát hiện lỗi chính tả, tóm tắt văn bản, phân tích cảm xúc...

Hàng loạt các mô hình xử lý ngôn ngữ tự nhiên đang ngày càng phát triển và ra đời. Kể từ khi mô hình Transformer ra đời, nó được ví như một bước ngoặt lớn trong ngành NLP. Và được sử dụng để tạo rất nhiều mô hình nổi tiếng sau này như BERT và các biến thể của nó. Trong báo cáo này, với mục đích nghiên cứu về BERT và các biến thể, em thực hiện tìm hiểu lý thuyết về các mô hình và áp dụng vào bài toán trích xuất cụm từ nhấn mạnh cảm xúc trong câu.

Nội dung báo cáo được chia làm 3 chương

- **Chương 1:** Các kiến thức cơ sở: các mô hình Seq2seq, Cơ chế Attention và Transformer. Đây là những kiến thức nền tảng không thể thiếu để có thể tìm hiểu về BERT.
- **Chương 2:** Các kiến thức về mô hình BERT và một số biến thể của nó.
- **Chương 3:** Xây dựng mô hình (Bert, AlBert, Roberta) và đánh giá kết quả thực nghiệm trên các mô hình đã xây dựng.

Chân thành cảm ơn TS. Nguyễn Kiêm Hiếu đã hướng dẫn em môn học Xử lý ngôn ngữ tự nhiên để em hoàn thành được báo cáo này.

Mặc dù đã cố gắng rất nhiều, nhưng với kiến thức và thời gian hạn chế nên không thể tránh khỏi những thiếu sót. Rất mong được sự góp ý của thầy cô và bạn bè để báo cáo này được hoàn thiện hơn.

Chương 1

Kiến thức cơ sở

1.1 Mô hình sequence to sequence

1.1.1 Tổng quan về sequence to sequence

Kể từ khi RNN, LSTM ra đời đã giải quyết được nhiều bài toán. Trong LSTM, chúng ta đã biết cách để xây dựng một mô hình để dự đoán một phần tử ở bước thời gian t dựa vào bước thời gian trước đó là t-1. Nhưng trực tiếp sử dụng LSTM để ánh xạ ngôn ngữ ngày sang ngôn ngữ khác nhanh chóng gặp phải sự cố, vì với LSTM thì chuỗi đầu vào và chuỗi đầu ra phải giống nhau về độ dài. Nhưng với bài toán dịch thì chuỗi đầu vào và chuỗi đầu ra có thể khác nhau về độ dài.

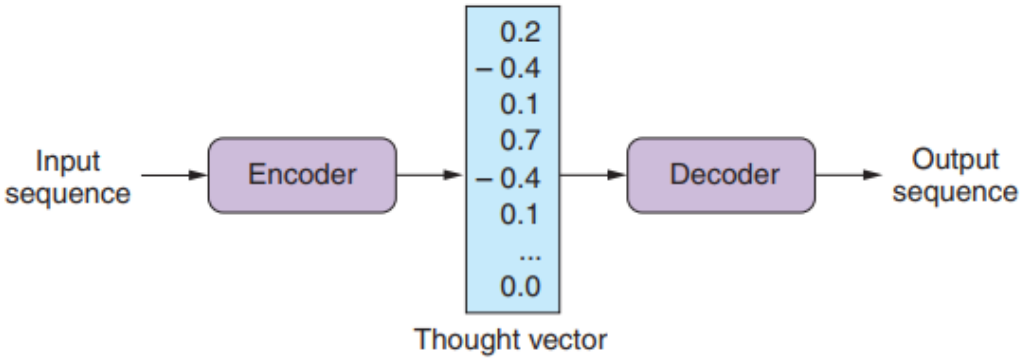
Mạng **sequence to sequence** hay gọi tắt là **seq2seq** giải quyết vấn đề này bằng cách tạo một biểu diễn đầu vào dưới dạng một vector (gọi là **thought vector**, hay **context vector**). Mô hình seq2seq sau đó sử dụng vector đó như một đầu vào cho mạng thứ 2 gọi là mạng decoder để sinh các chuỗi đầu ra.

Seq2seq là mô hình Deep Learning được giới thiệu bởi nhóm nghiên cứu của Google vào năm 2014. Seq2Seq được tạo ra nhằm mục đích tạo ra một output sequence từ một input sequence mà độ dài 2 chuỗi input và output có thể khác nhau.

Mục đích ban đầu của mô hình là áp dụng vào các bài Machine Translation, tuy nhiên hiện nay Seq2Seq cũng được áp dụng vào rất nhiều các bài toán khác nhau như Nhận dạng giọng nói, Tóm tắt văn bản, Chú thích ảnh, ...

Seq2Seq gồm 2 thành phần chính: Encoder và Decoder. Cả hai thành phần này đều được hình thành từ các mạng như RNN, LSTM, GRU. Nhiệm vụ của mỗi thành phần là:

- **Encoder:** Chuyển đổi dữ liệu đầu vào thành một vecto đại diện (*thought vector*).
- **Decoder:** Tạo ra chuỗi đầu ra từ vector đại diện được tạo ra từ Encoder.



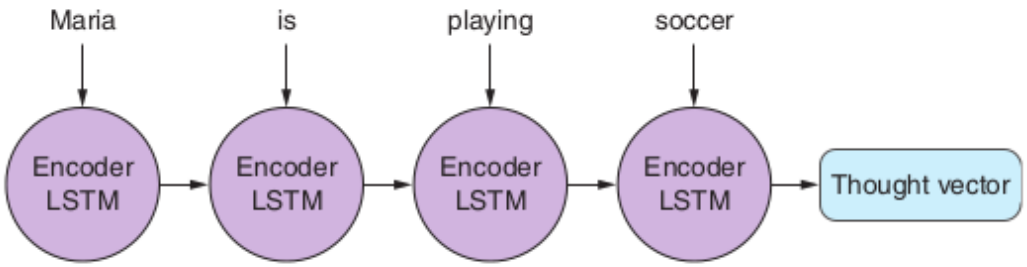
Hình 1.1: Sơ đồ mô hình seq2seq

Thought vector

Word vector (word embedding) là biểu diễn một từ thành một vector có độ dài cố định. Các từ có mối quan hệ tương tự nhau thì gần nhau trong không gian vector này. Một thought vector cũng rất giống như vậy, nó là một vector có độ dài cố định mà được nén thông tin từ một câu chứ không chỉ đơn thuần là một từ. Sau đó vector này được đưa vào bộ giải mã (decoder) để xử lý và đưa ra đầu ra.

1.1.2 Encoder

Thành phần encoder chứa các mạng con là phần tử của RNN (hoặc LSTM, GRU, ...), mục đích duy nhất của encoder là tạo ra thought vector, vector này sau đó được sử dụng làm trạng thái ban đầu của bộ giải mã (decoder).



Hình 1.2: Sơ đồ Encoder của seq2seq

Thought vector chính là vector state cuối cùng của chuỗi LSTM. Tại sao lại sử dụng vector state cuối cùng mà không phải toàn bộ?

Ví dụ 1. *Giả sử mạng RNN với đầu vào* ***X="How are you"****.*

$$x_1 = How$$
$$x_2 = are$$
$$x_3 = you$$

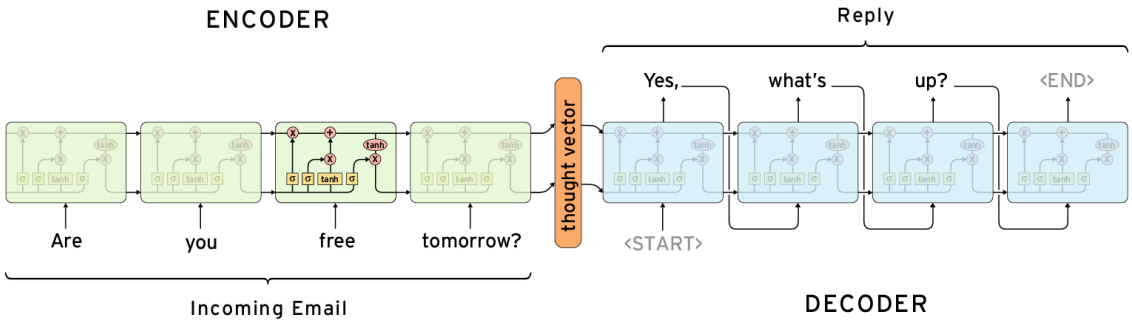
Công việc tính toán các state sẽ được thực hiện như sau:

$$S_1 = f(S_0, x_1)$$
$$S_2 = f(S_1, x_2) = f(f(S_0, x_1), x_2)$$
$$S_3 = f(S_2, x_3) = f(f(f(S_0, x_1), x_2), x_3)$$

Có thể thấy S_3 đã mang thông tin của x_1, x_2, x_3, như vậy, ta có thể kỳ vọng rằng trạng thái cuối cùng sẽ mang thông tin của cả câu đầu vào, và có thể đưa trạng thái này vào mạng decoder cho bước sau.

1.1.3 Decoder

Tương tự như Encoder thì Decoder cũng là ngăn xếp chứa các mạng con là phần tử của RNN (hoặc LSTM, GRU, ...) sẽ dự đoán đầu ra y_t tại thời điểm t. Mỗi phần tử này nhận đầu vào là trạng thái ẩn, đầu ra trước đó tạo ra kết quả đầu ra và trạng thái ẩn của chính nó.



Hình 1.3: Sơ đồ của seq2seq sử dụng LSTM

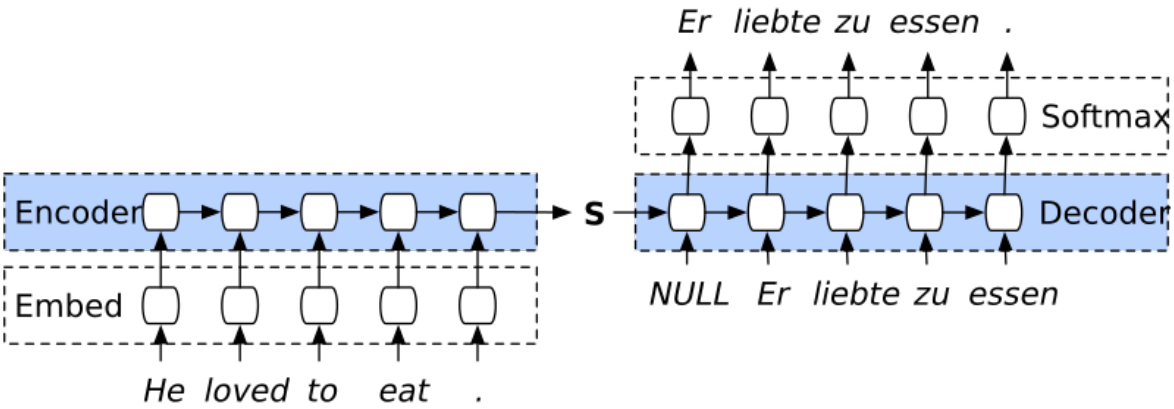
Hình 1.3 mô tả mô hình seq2seq sử dụng LSTM cho bài toán chatbot tiếng Anh.

Chuỗi đầu ra là tập các từ của câu trả lời, giả sử ta có tập các từ y_i trong đó i là thứ tự các từ thì đầu ra tại thời điểm t được tính bằng công thức:

$$y_t = \text{sofftmax}(W_s h_t)$$

Trong đó:

- W_s là ma trận trọng số của trạng thái ẩn và đầu ra.
- h_t là trạng thái ẩn ở bước t.



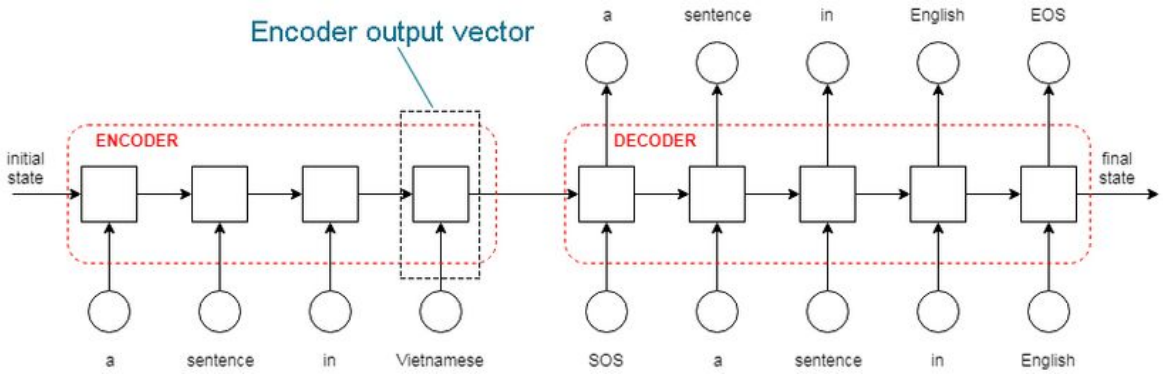
Hình 1.4: Mô hình seq2seq cho bài toán dịch tiếng Anh sang tiếng Pháp

Tùy từng bài toán cụ thể mà encoder và decoder có thể sẽ sử dụng các mạng deep learning khác nhau. Ví dụ như trong bài toán Machine Translate thì encoder thường là LSTM, GRU, Bidirectional RNN, còn trong bài toán chú thích ảnh thì encoder lại là CNN.

1.2 Cơ chế Attention

1.2.1 Tổng quan về cơ chế Attention

Trong mô hình sequence to sequence, chuỗi đầu vào được Encoder thành một vector ngữ cảnh có độ dài cố định đại diện cho lượng thông tin đầu vào. Sau đó, vector này được sử dụng làm trạng thái ban đầu của Decoder để sinh chuỗi đầu ra theo từng timestep. Cách hoạt động này yêu cầu đầu ra trạng thái cuối cùng của Encoder cần phải chứa tất cả các thông tin về câu nguồn. Dẫn đến việc vector context bị quá tải về mặt thông tin nếu như chuỗi nguồn dài.



Hình 1.5: Cơ chế Encoder-Decoder

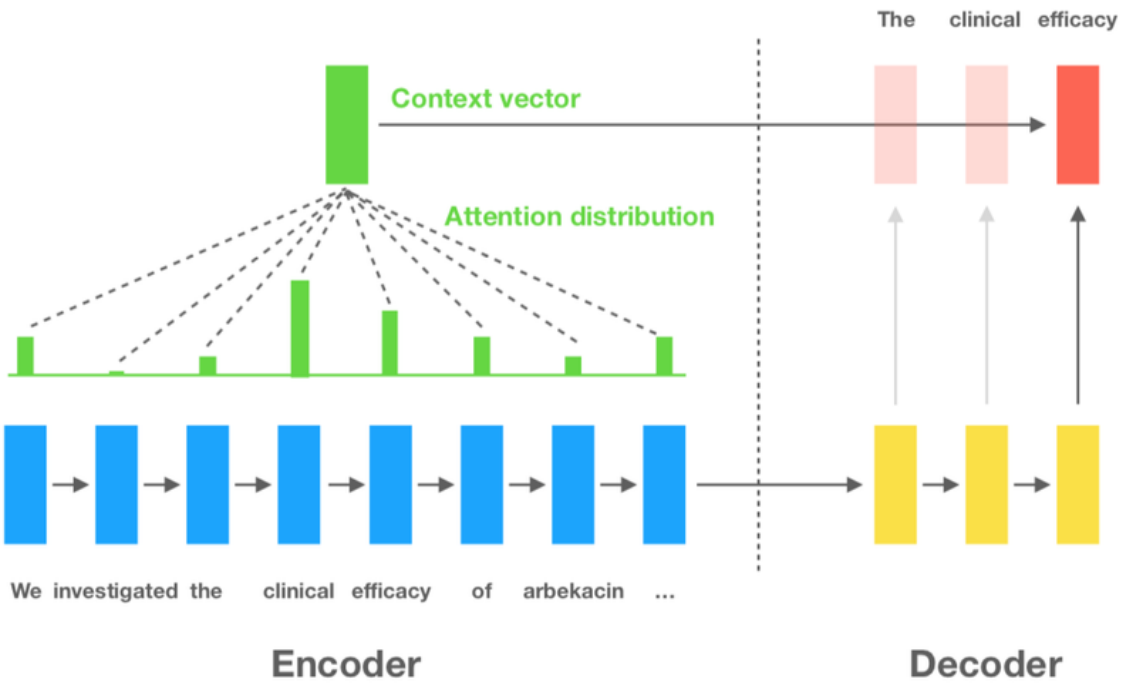
Cơ chế Attention được sinh ra dựa trên nhu cầu ghi nhớ các câu dài, một trong những vấn đề cơ bản của bài toán dịch máy. Attention cho phép mô hình tập trung vào một hoặc một vài ngữ cảnh địa phương trong câu, đó cũng là nguồn gốc tên gọi Attention. Thay vì để Encoder nén toàn bộ thông tin vào một vector trạng thái đầu ra cuối cùng, ta cho phép Decoder quan sát toàn bộ đầu ra của Encoder.

Ví dụ 2.

Giả sử cần dịch một câu *This is a book* sang tiếng Việt là *Đây là một quyển sách*. Ta có thể thấy sự tương ứng giữa từ ngữ ở 2 câu: *This* - *Đây* và *book* - *quyển sách*. Như vậy, việc dịch ra từ *quyển sách* thì từ *book* có ý nghĩa quan trọng hơn là các từ *this* hay *a* .... Và đó cũng chính là ý nghĩa cốt lõi của Attention.

Thay vì sử dụng context vector duy nhất một lần tại đầu vào của Decoder, ta sử dụng mỗi context vector riêng biệt cho từng phần để dự đoán ra từ kế tiếp, mỗi context vector được tổng hợp có trọng số từ các trạng thái ẩn trong Encoder (Hình 1.6).

Cách thức tổng hợp để ra được vector context (hay vector attention) có nhiều cách, trong phần này, chúng ta tìm hiểu về cơ chế tổng hợp attention của Bahdanau gọi là **Align and Jointly model** hay **Additive Attention** hay **Soft-Attention**.



Hình 1.6: Cơ chế Encoder-Decoder với Attention

Mô hình toán học của cơ chế Encoder-Decoder

Trong mô hình Encoder-Decoder, Encoder đọc một câu đầu vào gồm một chuỗi các vector $x = (x_1, \ldots, x_T)$ thành vector context c. Khi sử dụng RNN thì ta có:

$$h_t = f(x_t, h_{t-1})$$

và

$$c = q(\{h1, \ldots, h_T\})$$

Trong đó:

- $h_t \in \mathbb{R}^n$: trạng thái ẩn của tại bước t.
- c: Vector sinh bởi chuỗi trạng thái ẩn.
- f, q: Các hàm phi tuyến.

Decoder dự đoán y_t dựa vào context vector c và tất cả các từ trước nó $\{y_1, \ldots, y_{t-1}\}$. Nói cách khác, Decoder định nghĩa một xác xuất qua phép dịch y như sau:

$$p(y) = \prod_{t=1}^{T} p(y_t | \{y_1, \ldots, y_{t-1}\}, c)$$

Với: $y = (y_1, \ldots, y_T)$.
Khi sử dụng RNN cho Decoder, ta có:

$$p(y_t|\{y_1, \ldots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

Trong đó:

- g là hàm phi tuyến.
- s_t là trạng thái ẩn của RNN.

Mô hình toán học Encoder-Decoder với cơ chế Attention

Trong kiến trúc mới với Attention, xác suất để tạo ra đầu ra y_i tại Decoder biết các đầu tra trước đó $y_1, \ldots, y_{i-1}$ và câu nguồn x là:

$$p(y_i|\{y_1, \ldots, y_{i-1}\}, x) = g(y_{i-1}, s_i, c_i)$$

Với s_i là trạng thái ẩn của RNN tại bước thứ i: $s_i = f(s_{i-1}, y_{i-1}, c_i)$. Khác với mô hình Encoder-Decoder thông thường, tại mỗi đầu ra y_i, chúng ta sử dụng vector ngữ cảnh c_i. Context vector c_i là vector phụ thuộc vào bộ $h_1, \ldots, h_T$:

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j$$

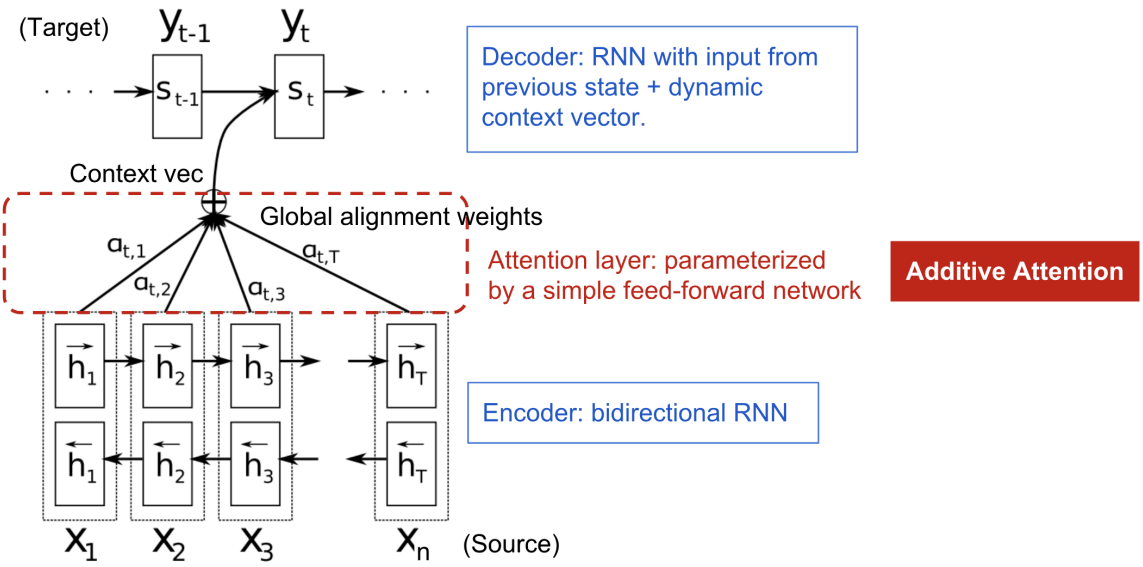
Trong đó:

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T} exp(e_{ik})}$$

biểu thị trọng số cho từ mục tiêu y_i được dịch từ từ nguồn x_j.Tại đây:

$$e_{ij} = a(a_{i-1}, h_j)$$

được gọi là **alignment model** với mục đích đánh giá mức độ tương quan của từ tại vị trí j của Encoder với từ đầu ra tại vị trí i của Decoder bằng việc gán trọng số vào α_{ij}. e_{ij} biểu thị tầm quan trọng của h_j với trạng thái ẩn trước đó s_{i-1} trong việc quyết định trạng thái tiếp theo s_i và tạo ra y_i.

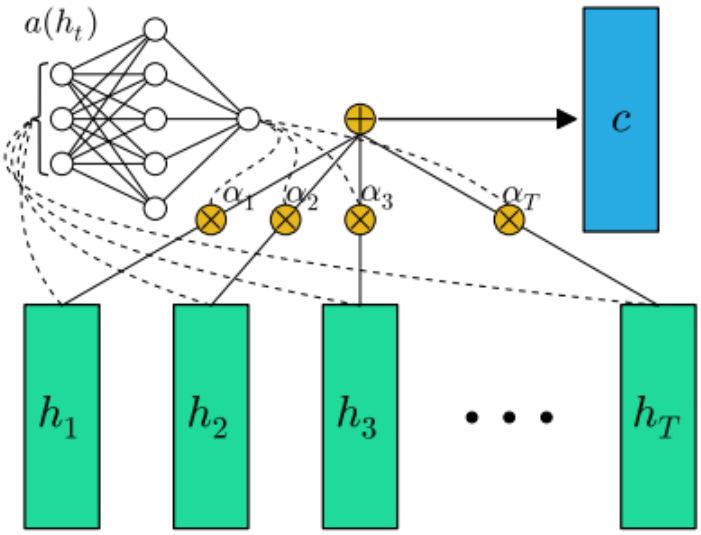


Hình 1.7: Hoạt động của Attention

Trong paper của tác giả Bahdanau, **alignment model** a được chọn với công thức như sau:

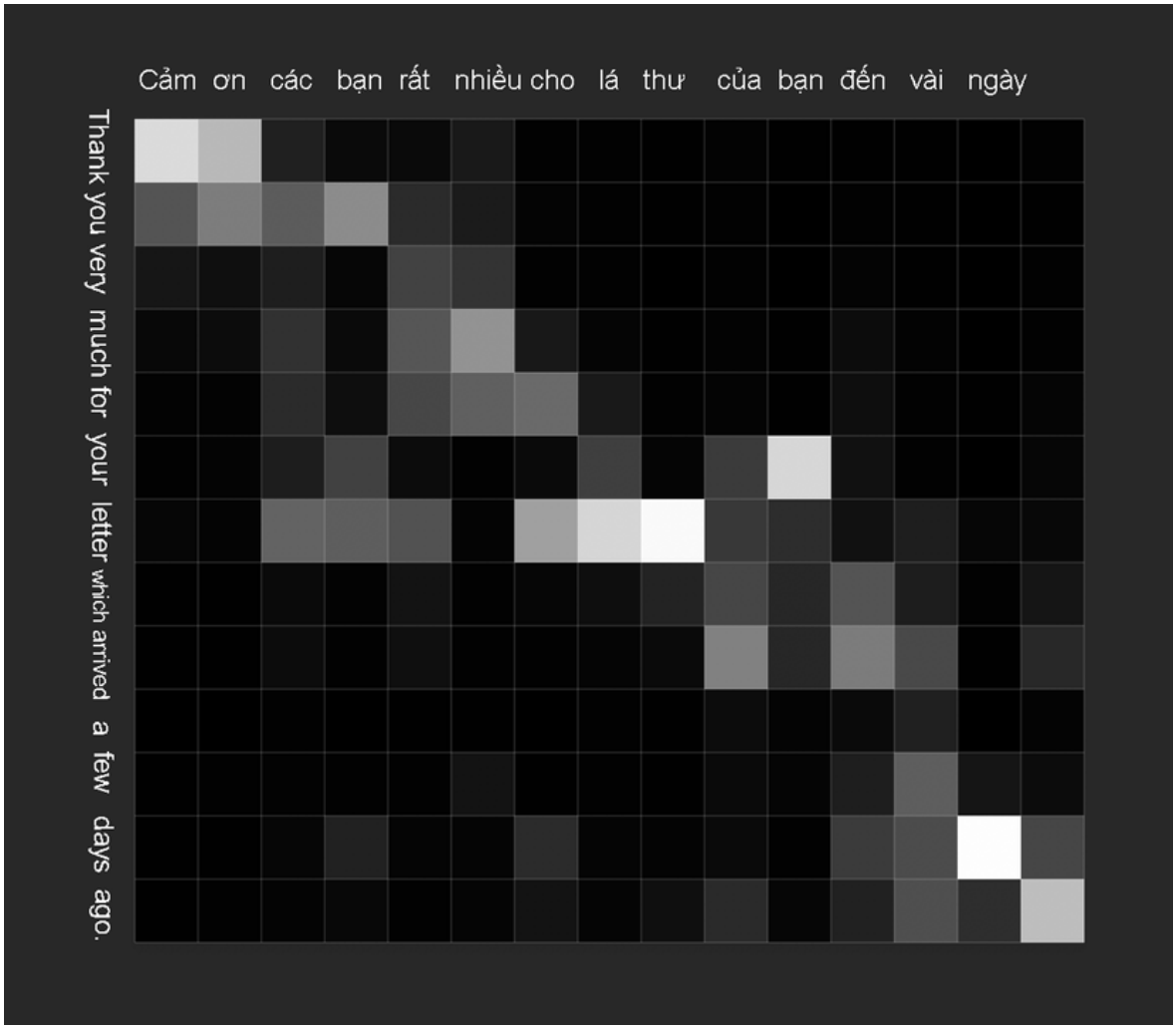
$$\text{score}(s_{i-1}, h_j) = a(s_{i-1}, h_j) = v_a^T \tanh(W_a s_{i-1} + U_a h_j)$$

thực chất là một multi-layer perceptron với W_a, U_a, v_a là các ma trận trọng số.



Hình 1.8: Cơ chế Attention Feed forward

Ma trận bất đối xứng (confusion matrix) được tạo ra bởi **alignment score** thể hiện mức độ tương quan giữa câu nguồn (source) và câu đích (target).



Hình 1.9: Ma trận bất đối xứng thể hiện mức độ tương quan của câu nguồn và câu đích

Nói cách khác, thực chất cơ chế Attention giúp mô hình tập trung vào phần quan trọng trên dữ liệu nguồn , bằng việc tạo ra một **alignment model a** tính các **alignment score** α_{ij} để thay đổi lại các trọng số của trạng thái ẩn của Encoder. Trong hình 1.9, các ô càng màu sáng thì biểu thị mức độ tương quan càng cao giữa từ x_j (source) và từ y_i (target).

1.2.2 Các biến thể của cơ chế Attention

Có nhiều cách tính **alignment score** như sau:

1. Content-base Attention [3]:

$$score(s_{i-1}, h_j) = \text{consine}[s_{i-1}, h_j]$$

2. Multiplicative Attention hay General Attention [4]:

$$score(s_{i-1}, h_j) = s_{i-1}^T W_a h_j$$

3. Dot Product [4]:

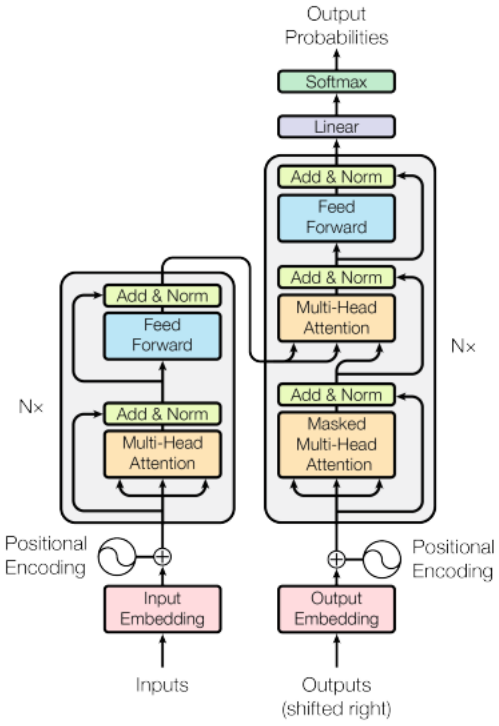
$$score(s_{i-1}, h_j) = s_{i-1}^T h_j$$

Ngoài ra có thể tham khảo thêm một số biến thể khác như Hard Attention [5], Hierarchical Attention [6].

1.3 Mô hình Transformer

1.3.1 Tổng quan về mô hình Transformer

Trong bài báo **Attention is all you need** [7], các tác giả giới thiệu về kiểu attention model được sử dụng trong các tác vụ học máy. Trong bài báo cũng đưa ra một kiến trúc mới là **Transformer** hoàn toàn khác so với các kiến trúc RNN trước đây, mặc dù cả hai đều thuộc lớp model seq2seq nhằm chuyển một câu văn đầu vào ở ngôn ngữ A sang 1 câu văn đầu ra ở ngôn ngữ B. Quá trình biến đổi (transorming) được dựa trên 2 phần encoder và decoder. Nghiên cứu chỉ ra rằng với cơ chế Attention không cần cần đến RNN đã có thể cải thiện hiệu quả của các tác vụ dịch máy và nhiều tác vụ khác.



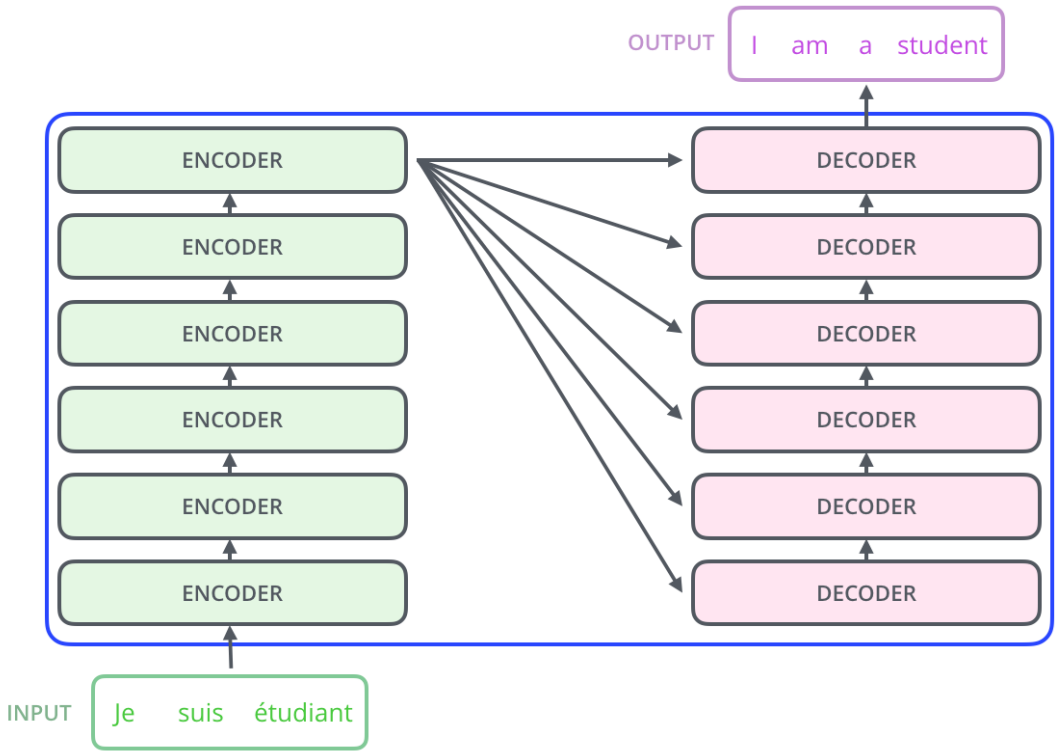
Hình 1.10: Kiến trúc mô hình Transformer

Kiến trúc mồ hình **Transformer** gồm hai phân Encoder bên trái và Decoder bên phải (Hình 1.10).

- **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 attention. Layer thứ hai đơn thuần là các fully-connected feed-forward layer. Một lưu ý là chúng ta sẽ sử dụng một kết nối residual ở mỗi sub-layer, kiến trúc này tương tự như mạng resnet trong CNN. Đầu ra của mỗi sub-layer là *LayerNorm(x+Sublayer(x))* có số chiều là 512 như bài báo.

- **Decoder**: Cũng là tổng hợp của 6 layers xếp chồng nhau, kiến trúc tương tự như các sub-layer của Encoder ngoại trừ thêm một 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.

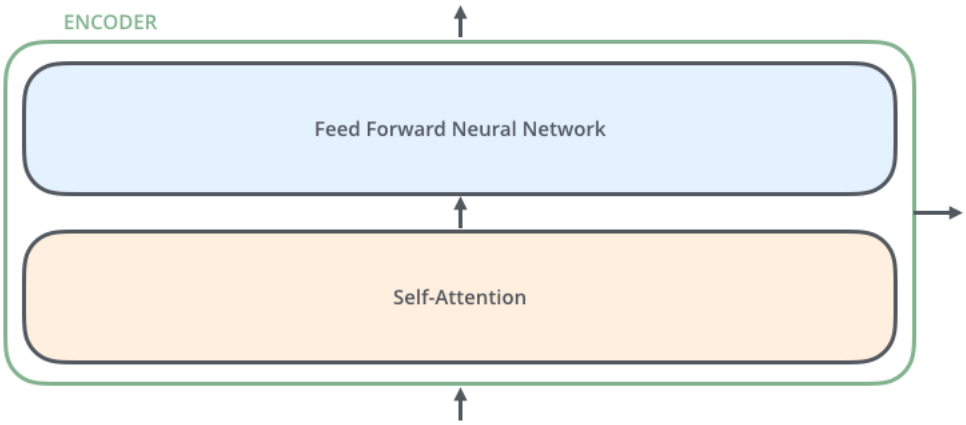
Ngoài ra, chúng ta có một bước cộng thêm **Positional Encoding** vào các input của encoder và decoder nhằm thêm yếu tố thời gian vào mô hình để tăng độ chính xác. Đây đơn giản là phép cộng vector mã hóa vị trí của từ trong câu với vector biểu diễn từ. Chúng ta có thể mã hóa vị trí dưới dạng $\{0, 1\}$ vector vị trí hoặc sử dụng hàm *sin, cos* như trong bài báo.



Hình 1.11: Kiến trúc mô hình Transformer sơ lược

1.3.2 Self attention

Như đã giới thiệu ở phần trên, Encoder và Decoder cơ bản giống nhau về cấu trúc, gồm nhiều layer xếp chồng lên nhau, trong mỗi layer này lại được chia thành 2 sub-layer (Hình 1.12).

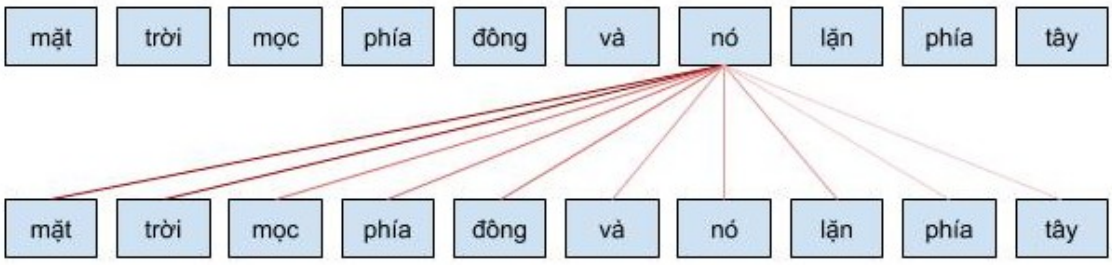


Hình 1.12: Kiến trúc của một layer trong Encoder và Decoder

Self Attention cho phép mô hình khi mã hóa một từ có thể sử dụng thông tin của những từ liên quan tới nó. Chúng ta có thể tưởng tượng cơ chế self attention giống như cơ chế tìm kiếm. Với một từ cho trước, cơ chế này sẽ cho phép mô hình tìm kiếm trong các từ còn lại, từ nào "giống" để sau đó thông tin sẽ được mã hóa dựa trên tất cả các từ trên.

Ví dụ 3.

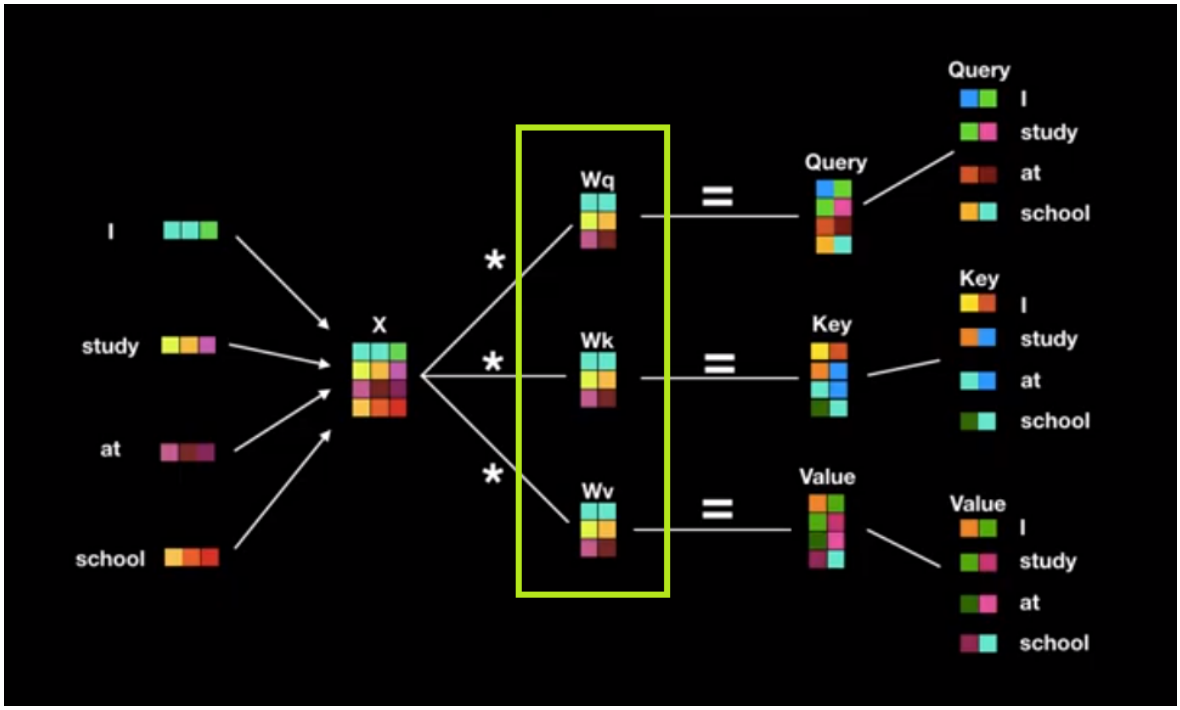
Chúng ta có câu: *mặt trời mọc phía đông và nó lặn ở phía tây* thì cơ chế selff attention sẽ giúp mô hình khi mã hóa từ *nó* thì sẽ tập trung vào từ liên quan là *mặt trời*.



Hình 1.13: Ví dụ cơ chế self attention

Scale dot product attention

Scale dot product attention là một cơ chế self attention. Sau bước nhúng từ (đi qua embedding layer) ta có đầu vào của encoder và decoder là ma trận X kích thước $m \times n$ với m, n lần lượt là độ dài câu và số chiều của một vector nhúng từ.



Hình 1.14: Self attention

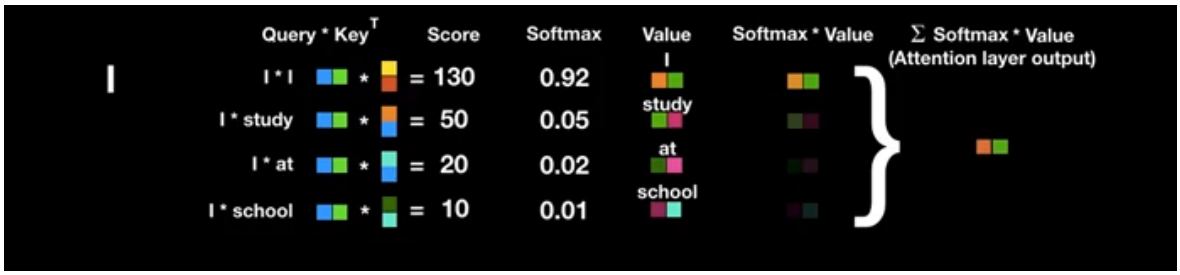
Trong hình 1.14, khung màu vàng là 3 ma trận W_q, W_k, W_v là những hệ số mà mô hình cần huấn luyện. Sau khi nhân các ma trận này với ma trận đầu vào X ta thu được ma trận Q, K, V (tương ứng gọi là Query, Key, Value). Ma trận Query và Key có tác dụng tính toán ra phân phối score cho các cặp từ. Ma trận Value sẽ dựa trên phân phối score để tính ra vector phân phối xác suất đầu ra.

Như vậy, mỗi từ sẽ được gán bởi 3 vector querry, key và value là các dòng của ma trận Q, K, V (Hình 1.15).

- **query vector**: Vector dùng để chứa thông tin của các từ được tìm kiếm, so sánh. Giống như câu query của google search.
- **key vector**: Vector dùng để biểu diễn thông tin của các từ được so sánh với từ cần tìm kiếm ở trên. Ví dụ như các trang web mà google sẽ so sánh với từ khóa mà chúng ta tìm kiếm.
- **value vector**: Vector biểu diễn nội dung, ý nghĩa của các từ. Giống như nội dung trang web được hiển thị cho người dùng sau khi tìm kiếm.

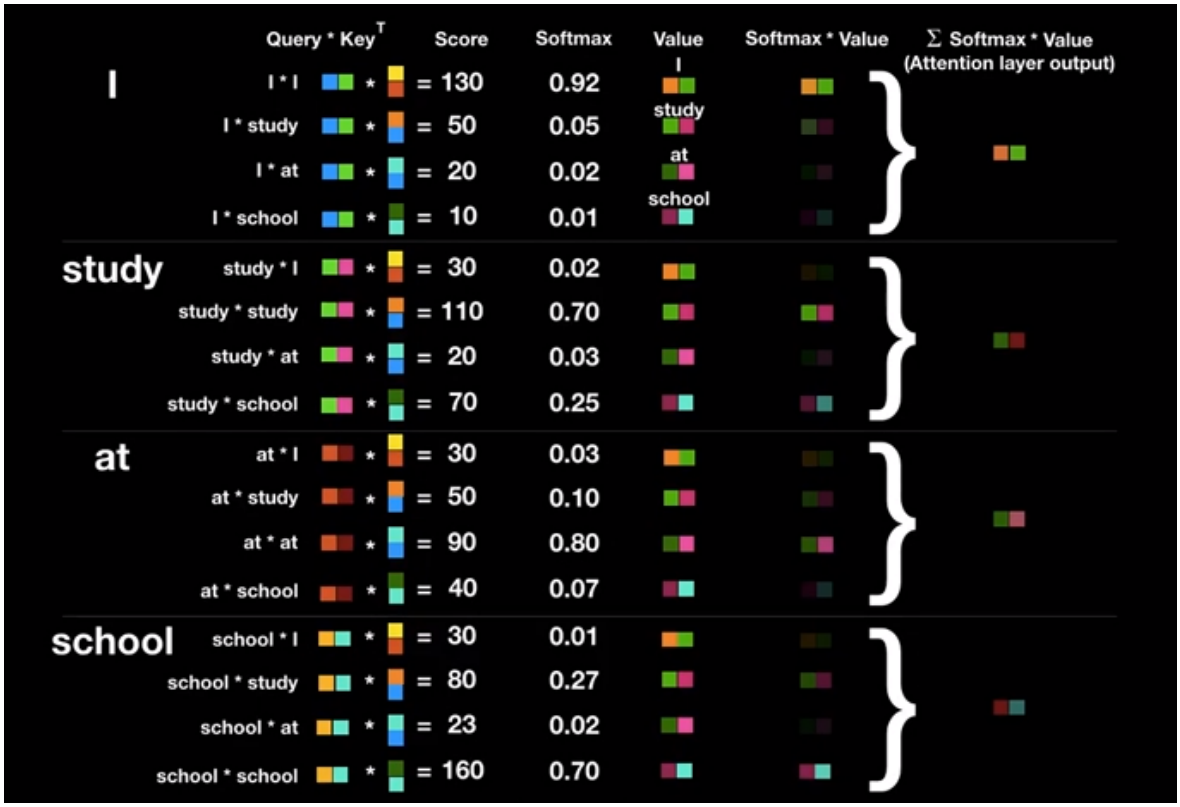
Hình 1.15: Các từ đầu vào tương ứng với key, query và value

Để tính tương quan, hay còn gọi là score giữa mỗi cặp từ (w_j, w_j), chúng ta tính dot-prroduct (tích vô hướng) giữa query và key, phép tính này nhằm tìm ra mối liên hệ trọng số của các cặp từ. Tuy nhiên, điểm số sau cùng là điểm chưa được chuẩn hóa. Do đó, chúng ta cũng có thể chuẩn hóa bằng hàm softmax để đưa về một phân phối xác suất mà độ lớn sẽ đại diện cho mức độ chú ý của từ query tới từ key. Trọng số càng lớn thì càng chứng tỏ từ w_i trả về một sự chú ý lớn hơn đối với từ w_j. Sau đó, chúng ta nhân hàm softmax với các vector value của từ này để tìm ra vector đại diện (attention vector) sau khi đã học trên toàn bộ câu đầu vào. Hình 1.16 mô tả ví dụ về quá trình tính toán này.



Hình 1.16: Qúa trình tính toán trọng số attention vector cho từ *I* trong câu *I study at school*

Tương tự khi tính với các từ khác trong câu, ta thu được kết quả như hình 1.17 minh họa.

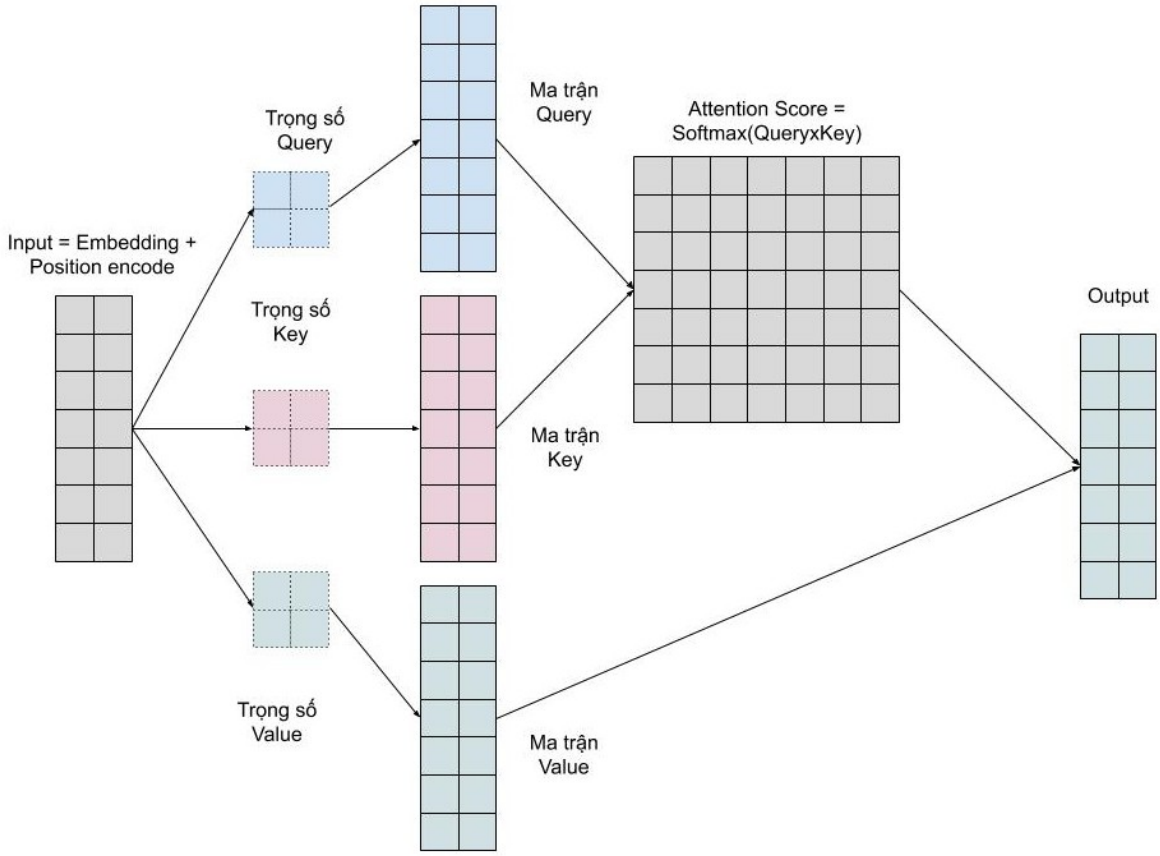


Hình 1.17: Kết quả tính attention vector cho toàn bộ các từ trong câu

Từ triển khai trên các vector dòng, chúng ta có thể tóm tắt triển khai trên ma trận. Phương trình Attention như sau:

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

Trong đó d_k là số chiều key vector. Việc chia cho d_k nhằm tránh tràn luồng nếu số mũ là quá lớn.



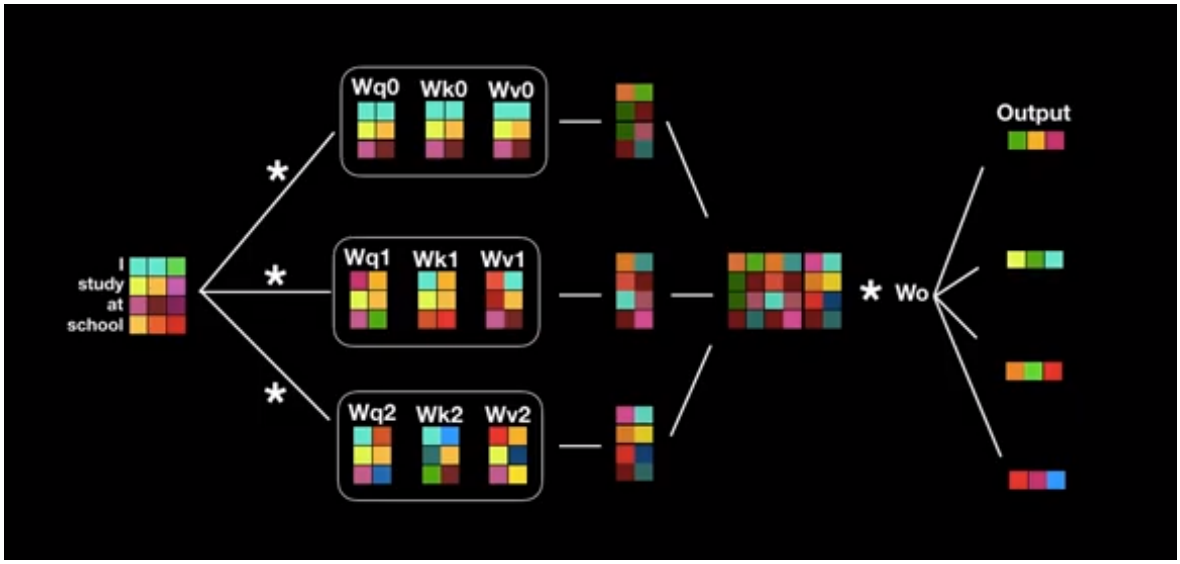
Hình 1.18: Mô phỏng cách tính toán trên self attention

Multi-head Attention

Như vậy sau quá trình scale dot product chúng ta sẽ thu được ma trận Attention. Các tham số mà mô hình cần tinh chỉnh chính là các ma trận W_q, W_k, W_v. Mỗi quá trình như vậy được gọi là 1 head của attention. Khi lặp này quá trình này nhiều lần (trong bài báo là 3 laafn) ta sẽ thu được quá trình **Multi-head Attention**. Chúng ta muốn mô hình có thể học nhiều kiểu mối quan hệ giữa các từ với nhau. Với mỗi self-attention, chúng ta học được một kiểu pattern, do đó để có thể mở rộng khả năng này, chúng ta đơn giản là thêm nhiều self-attention. Công thức Multi-head Attention như sau:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{concatenate}(head_1, head_2, ..., head_h)\mathbf{W_0}$$

Ở đây $head_i = \text{Attention}(\mathbf{Q_i}, \mathbf{K_i}, \mathbf{V_i})$. Để trả về output có cùng kích thước với ma trận đầu vào chúng ta chỉ cần nhân với ma trận $\mathbf{W_0}$ chiều cao bằng với chiều rộng của ma trận đầu vào.

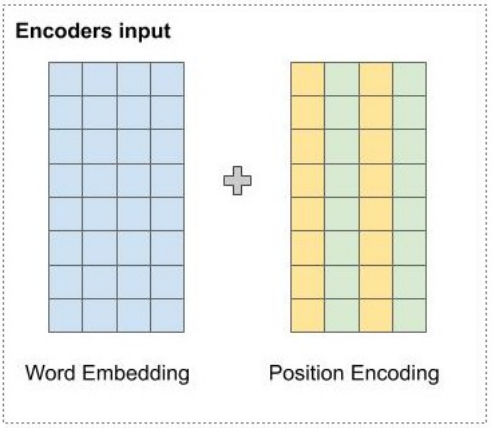


Hình 1.19: Sơ đồ cấu trúc multi-head attention

1.3.3 Position encoding

Transformer không dự báo tuân theo time step như RNN. Toàn bộ dữ liệu đầu vào sẽ được truyền thẳng vào mô hình cùng một thời điểm. Sẽ rất khó để nhận biết vị trí của các từ đầu vào trong câu ở những lớp mô hình không tuân theo time step. Do đó thêm position encoding sẽ cho mô hình thêm các thông tin về vị trí của từ.

Position encoding vector sẽ được cộng trực tiếp vào embedding vector. Embeddings biểu diễn một token trong một không gian d chiều nơi mà các token có cùng ý nghĩa sẽ gần nhau hơn. Nhưng embedding vector không chứa thông tin vị trí của từ trong câu. Do đó sau khi thêm position encoding vector, một từ sẽ gần với những từ khác hơn dựa trên ý nghĩa của chúng và khoảng cách vị trí của chúng trong câu trong không gian d chiều.



Hình 1.20: Phương pháp Position Encoding

Cụ thể, tại vị trí chẵn, tác giả sử dụng hàm sin, và với vị trí lẽ tác giả sử dụng hàm cos để tính giá trị tại chiều đó.

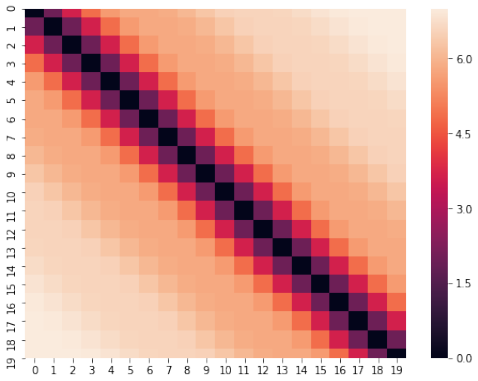
$$PE(pos, 2i) = sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$
$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

Trong đó pos là vị trí hiện tại của từ, i là chỉ số của phần tử nằm trong vector encoding và d_{model} là kích thước các vector positional embedding. Vector này có kích thước bằng với các vector embedding từ để có thể thực hiện được phép cộng.

	0	1	2	3	4	5
0	$\sin(\frac{1}{1000^{0/512}} \times 0)$	$\cos(\frac{1}{1000^{0/512}} \times 0)$	$\sin(\frac{1}{1000^{2/512}} \times 0)$	$\cos(\frac{1}{1000^{2/512}} \times 0)$	$\sin(\frac{1}{1000^{4/512}} \times 0)$	$\cos(\frac{1}{1000^{4/512}} \times 0)$
1	$\sin(\frac{1}{1000^{0/512}} \times 1)$	$\cos(\frac{1}{1000^{0/512}} \times 1)$	$\sin(\frac{1}{1000^{2/512}} \times 1)$	$\cos(\frac{1}{1000^{2/512}} \times 1)$	$\sin(\frac{1}{1000^{4/512}} \times 1)$	$\cos(\frac{1}{1000^{4/512}} \times 1)$
2	$\sin(\frac{1}{1000^{0/512}} \times 2)$	$\cos(\frac{1}{1000^{0/512}} \times 2)$	$\sin(\frac{1}{1000^{2/512}} \times 2)$	$\cos(\frac{1}{1000^{2/512}} \times 2)$	$\sin(\frac{1}{1000^{4/512}} \times 2)$	$\cos(\frac{1}{1000^{4/512}} \times 2)$

Hình 1.21: Ví dụ Position Encoding với số chiều là 6

Hình 1.22 minh họa cho cách tính position encoding. Giả sử chúng ta có word embedding có 6 chiều, thì position encoding cũng có tương ứng là 6 chiều. Mỗi dòng tương ứng với một từ. Giá trị của các vector tại mỗi vị trí được tính toán theo công thức như hình.

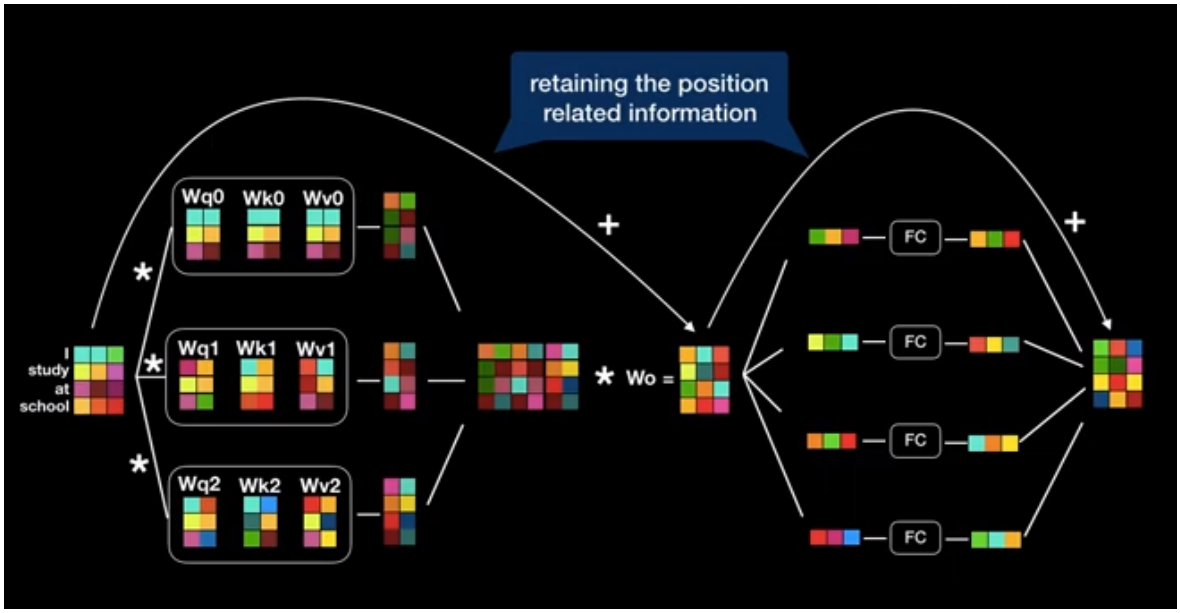


Hình 1.22: Ma trận position encoding

Có thể thấy sau khi mã hóa, ma trận position encoding thu được các vector biểu diễn thể hiện được tính chất khoảng cách giữa 2 từ. 2 từ cách càng xa nhau thì khoảng cách càng lớn hơn.

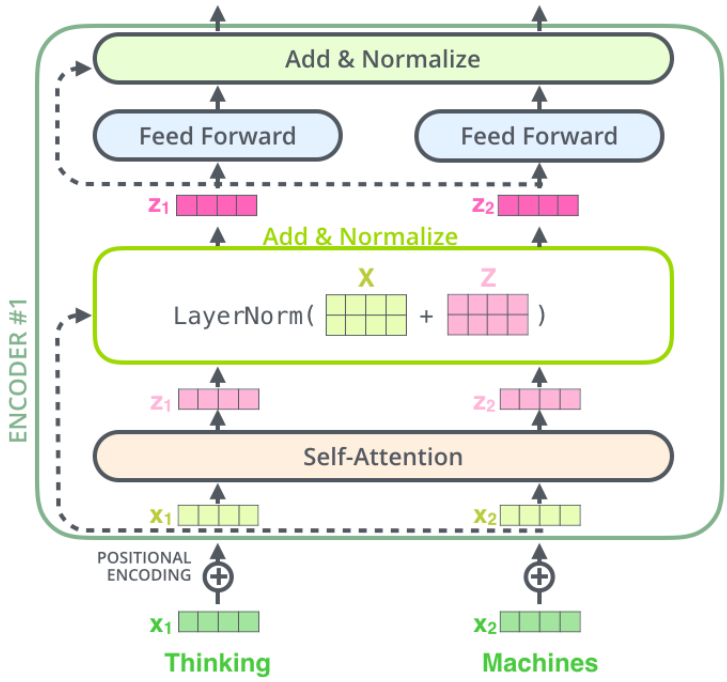
1.3.4 Quá trình encoder và decoder

Encoder đầu tiên sẽ nhận ma trận biểu diễn của các từ đã được cộng với thông tin vị trí thông qua positional encoding. Sau đó, ma trận này sẽ được xử lý bởi Multi Head Attention. Ở sub-layer thứ 2 chúng ta sẽ đi qua các kết nối fully connected và trả ra kết quả ở đầu ra có shape trùng với input. Mục đích là để chúng ta có thể lặp lại các block này Nx lần.



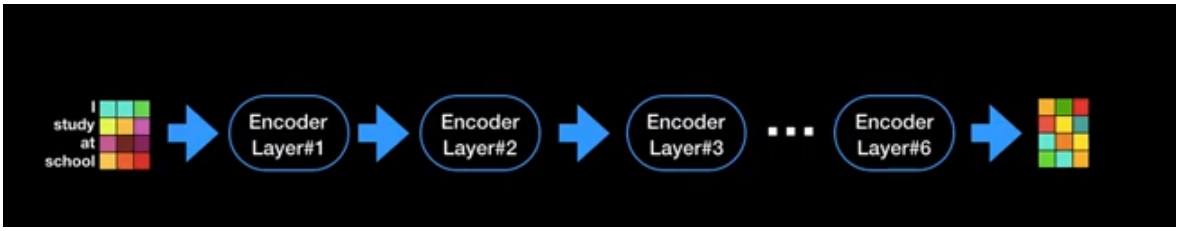
Hình 1.23: Sơ đồ của 1 block layer áp dụng multi-head attention

Sau các biến đổi backpropagation chúng ta thường mất đi thông tin về vị trí của từ. Do đó chúng ta sẽ áp dụng một residual connection để cập nhật thông tin trước đó vào output. Để quá trình training là ổn định chúng ta sẽ áp dụng thêm một layer Normalization nữa ngay sau phép cộng.



Hình 1.24: Cơ chế hoạt động của residual connetion

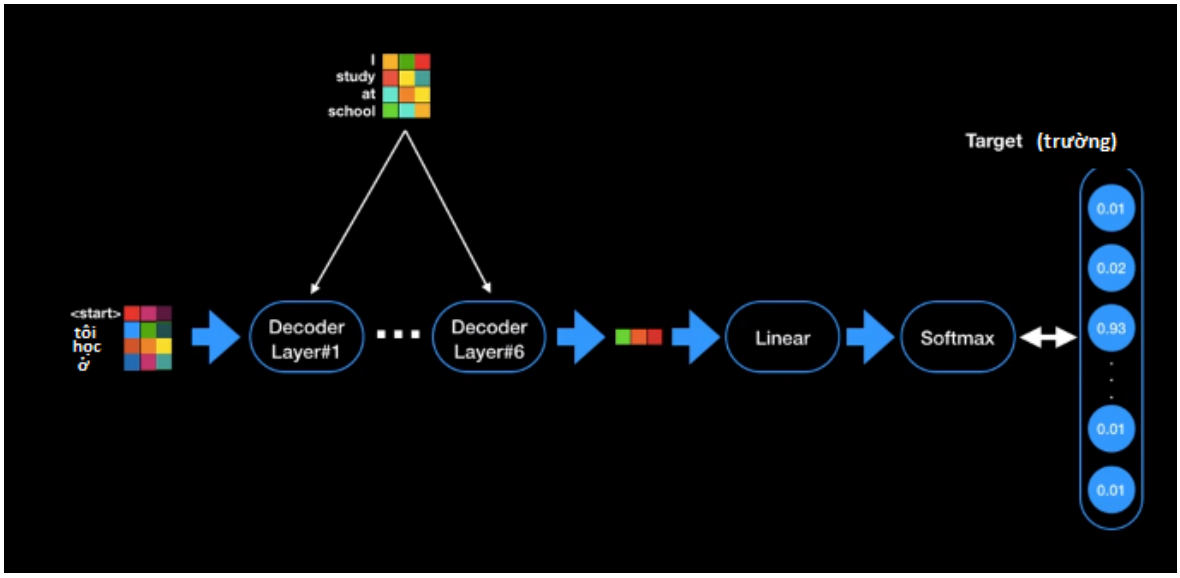
Nếu lặp lại block layer trên 6 lần và kí hiệu chúng là một encoder. Chúng ta có thể đơn giản hóa đồ thị của quá trình encoder như hình 1.25.



Hình 1.25: Sơ đồ quá trình encoder với 6 layer của transformer

Qúa trình decoder cũng hoàn toàn tương tự như encoder ngoại trừ một số lưu ý:

- Quá trình decoder sẽ tạo từ theo tuần tự từ trái qua phải tại mỗi bước thời gian.
- Ở mỗi một block layer của decoder chúng ta sẽ phải thêm ma trận cuối cùng của encoder như một input của multi-head attention.
- Thêm một layer Masked Multi-head Attention sub-layer ở đầu tiên ở mỗi block layer. Layer này không có gì khác so với Multi-head Attention ngoại trừ không tính đến attention của những từ trong tương lai.

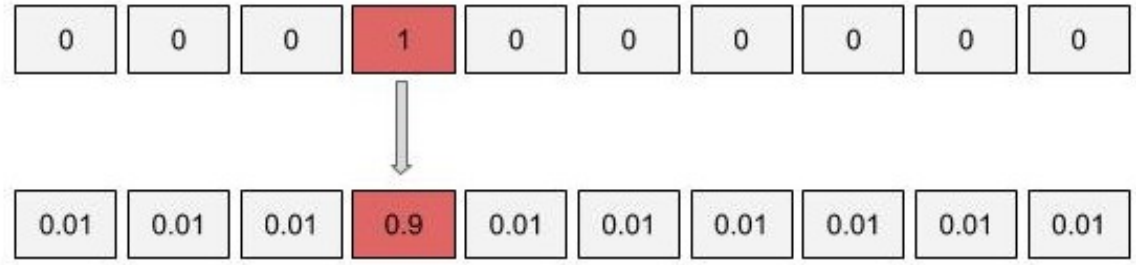


Hình 1.26: Sơ đồ quá trình decoder của transformer

Như hình 1.26, chúng ta thấy ở mỗi bước thời gian t, decoder sẽ nhận giá trị đầu vào là final-ouput của encoder, input của từ ở vị trí thứ $t-1$ ở decoder (đây là giá trị được dự báo ở bước thời gian thứ $t-1$ của model). Sau khi đi qua 6 block layers của decoder model sẽ trả ra một vector đại diện cho từ được dự báo. Hàm linear kết hợp với softmax được sử dụng để tính ra giá trị phân phối xác xuất của từ mục tiêu. Để nâng cao accuracy thì tác giả trong bài báo gốc có nói sử dụng kĩ thuật label smoothing tại ngưỡng label $\epsilon_{ls} = 0.1$ nhằm giảm các label tại vị trí mục tiêu xuống nhỏ hơn 1 và các vị trí khác lớn hơn 0. Việc này gây ảnh hưởng tới sự không chắc chắn của model nhưng có tác dụng trong gia tăng accuracy bởi trên thực tế 1 câu có thể có nhiều cách dịch khác nhau. Chẳng hạn như *I study at school* có thể dịch nhiều nghĩa như *tôi học ở trường* hoặc *tôi nghiên cứu ở trường*.

Label smoothing

Với mô hình nhiều triệu tham số của transformer, thì việc overfit là chuyện dễ dàng xảy ra. Để hạn chế hiện tượng overfit, có thể sử dụng kỹ thuật label smoothing. Về cơ bản thì ý tưởng của kỹ thuật này khá đơn giản, chúng ta sẽ phạt mô hình khi nó quá tự tin vào việc dự đoán của mình. Thay vì mã hóa nhãn là một one-hot vector, chúng ta sẽ thay đổi nhãn này một chút bằng cách phân bố một tí xác suất vào các trường hợp còn lại.

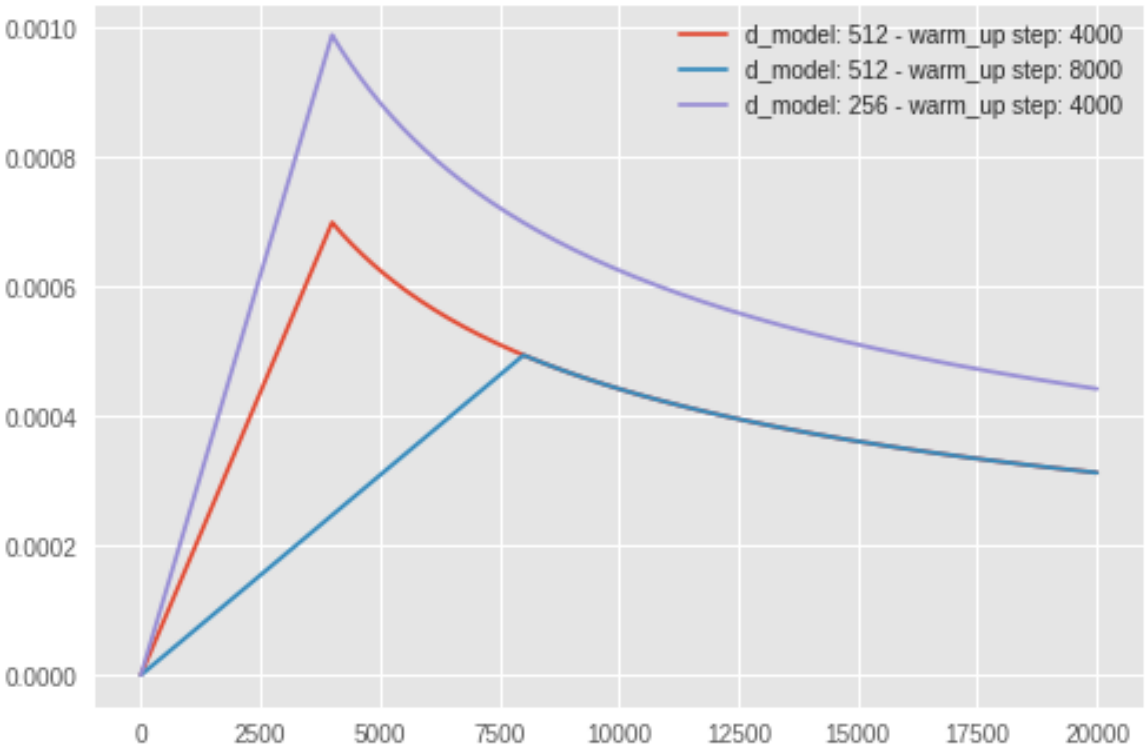


Hình 1.27: Kỹ thuật label smoothing

Optimizer

Để huấn luyện mô hình transformer, chúng ta vẫn sử dụng Adam optimizer với $\beta_1 = 0.9$, $\beta_2 = 0.98$ và $\epsilon = 10^{-9}$. Tuy nhiên, learning rate cần phải được điều chỉnh trong suốt quá trình học theo công thức sau:

$$lrate = d_{\text{model}}^{-0.5}.\min(step\_num^{-0.5}, step\_num * warmup\_steps^{-1.5})$$



Hình 1.28: Thực nghiệm với learning rate theo bài báo

Cơ bản thì learning rate sẽ tăng dần trong các lần cập nhật đầu tiên, các bước này được gọi là warm up step, lúc này mô hình sẽ 'chạy' tẹt ga. Sau đó learning rate lại giảm dần, để mô hình hội tụ.

Chương 2

Mô hình ngôn ngữ BERT

BERT là viết tắt của cụm từ **Bidirectional Encoder Respresetation from Transformer** là một mô hình được Google công bố vào tháng 11 năm 2018. Là mô hình biểu diễn từ theo 2 chiều ứng dụng kỹ thuật **Transformer**. BERT được thiết kế để huấn luyện trước các biểu diễn từ (pre-train word embedding) được sử dụng để transfer sang các bài toán khác trong lĩnh vực xử lý ngôn ngữ tự nhiên.

Cơ chế attention của Transformer sẽ truyền toàn bộ các từ trong câu văn đồng thời vào mô hình một lúc mà không cần quan tâm đến chiều của câu. Do đó Transformer được xem như là huấn luyện hai chiều (bidirectional) mặc dù trên thực tế chính xác hơn chúng ta có thể nói rằng đó là huấn luyện không chiều (non-directional). Đặc điểm này cho phép mô hình học được bối cảnh của từ dựa trên toàn bộ các từ xung quanh nó bao gồm cả từ bên trái và từ bên phải.

2.1 Kiến trúc BERT

Mô hình BERT là mô hình mã hóa sử dụng nhiều lớp Transformer hai chiều , trong đó các lớp Transformer được sử dụng hoàn toàn tương tự như mô hình được đề xuất và thực thi gốc của Vasawwani et al. (2017) . Kiến trúc của Transformer được nói rất kỹ ở phần trên và việc thực thi kiến trúc này trong BERT hoàn toàn giống với cách triển khai gốc đó.

Mô hình BERT định nghĩa số lượng lớp (khối Transformer) là **L**, kích thước của các lớp ẩn là **H** (hay còn gọi là kích thước của embedding vector) và số head ở lớp self attention là **A**. Tên gọi của 2 kiến trúc BERT bao gồm:

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

2.2 Biểu diễn dữ liệu đầu vào

Cách thức biểu diễn dữ liệu đầu vào của BERT cho phép biểu diễn đồng thời một câu hoặc một cặp câu (Ví dụ [câu hỏi, câu trả lời]) trong một chuỗi biểu diễn.

Khi có một chuỗi đầu vào cụ thể, biểu diễn đầu vào của chúng ta được xây dựng bằng cách tính tổng các token đó với vector phân đoạn và vị trí tương ứng của các từ trong chuỗi (Hình 2.1).

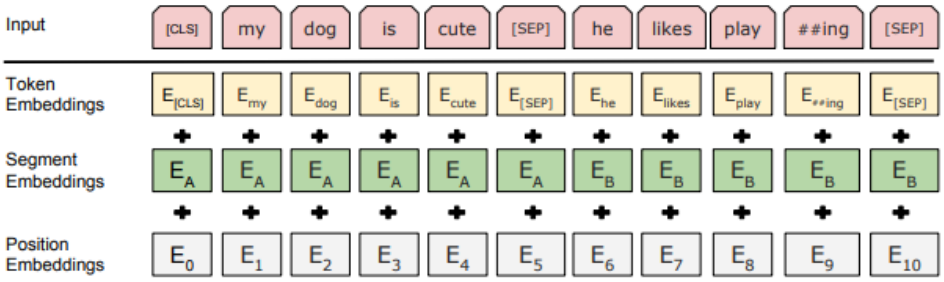
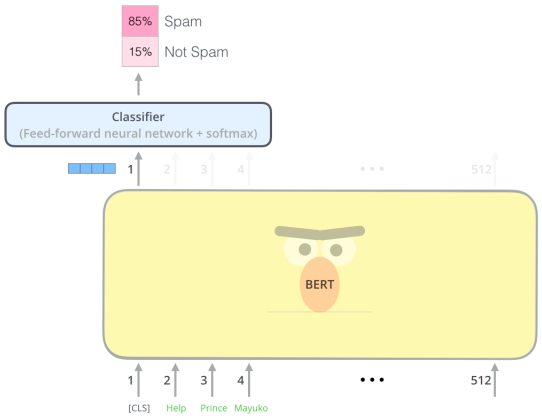


Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

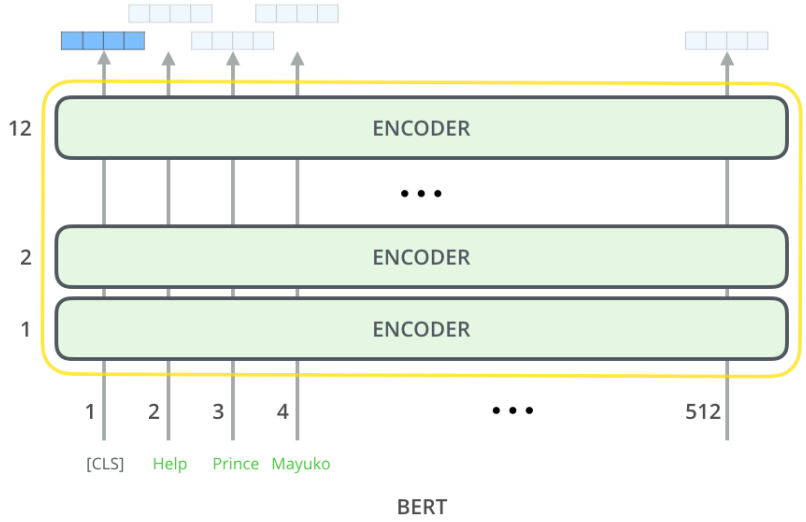
Hình 2.1: Biểu diễn đầu vào của BERT

Một số điểm cần chú ý:

- BERT sử dụng bộ WordPiece (Wu et al.2016) để tách các câu thành các từ nhỏ với bộ từ điển bao gồm 30000 từ và sử dụng ## làm dấu phân tách. Ví dụ từ playing được tách thành play##ing.
- Lớp nhúng vị trí (positional embedding) được sử dụng với độ dài tối đa 512.
- Token đầu tiên của mỗi chuỗi luôn mặc định là ký tự đặc biệt ([CLS]). Đầu ra của Transformer (hidden state cuối cùng) tương ứng với token này sẽ được sử dụng để đại diện cho cả câu trong các nhiệm vụ phân loại (Hình 2.2). Nếu không trong các nhiệm vụ phân loại, vector này được bỏ qua (Hình 2.3).



Hình 2.2: Biểu diễn đầu ra của BERT cho tác vụ phân loại



Hình 2.3: Biểu diễn đầu ra của BERT cho tác vụ không phải phân loại

- Trong trường hợp cặp các câu được gộp lại với nhau thành một chuỗi duy nhất, chúng ta phân biệt các câu theo 2 cách:
  - Dấu hiệu thứ nhất là sử dụng ký tự ngăn cách [SEP].
  - Dấu hiệu thứ hai là sử dụng giá trị nhúng câu A đã được học cho chuỗi đầu tiên và giá trị nhúng B cho chuỗi thứ hai . Đối với tác vụ chỉ sử dụng một câu đơn , BERT chỉ sử sử dụng duy nhất giá trị nhứng A.
- Lớp Position Embeddings có nhiệm vụ mã hóa thông tin vị trí của từ trong câu. Nếu để ý , có thể thấy toàn bộ mô hình BERT chỉ sử dụng Transformer encoder tức là chỉ có attention và các mạng neural truyền thẳng với kỹ thuật normalize, hoàn toàn không hề có thứ tự như các mang neural hồi quy. Để mô hình có thể

học được sự liên hệ về thứ tự xuất hiện của các từ thì người ta đưa thêm lớp Position Embeddings mang ý nghĩa về thứ tự của từ trong chuỗi.

- Lớp Segment Embedding mang thông tin về vị trí câu mà từ đó đang phụ thuộc. Ví dụ như hình những từ có Segment là E_A tức là nó thuộc câu A và ngược lại. Trên thực tế giá trị này sẽ là 0 nếu thuộc câu đầu tiên và 1 nếu thuộc câu thứ hai. Nếu tác vụ chỉ liên quan đến một câu thì chỉ cần gán các giá trị này bằng 0 hết.

- Lớp Token Embeddings có nhiệm vụ cho mô hình biết được vị trí của từ này trong tập từ điển, tương tự như mô hình word2vec nó được biểu diễn dưới dạng one hot vector với tập từ điển word piece 30000 token.

2.3 Pre-training BERT

Mô hình BERT được tiền huấn luyện bởi hai tác vụ đó là **Mô hình ngôn ngữ đánh dấu - Masked Language Model** và **Dự đoán câu kế tiếp - Next Sentence Prediction**

2.3.1 Masked Language Model

Masked Language Model (gọi tắt là **MLM**) là mô hình mà bối cảnh của từ được học từ cả 2 phía bên trái và bên phải cùng một lúc từ những bộ dữ liệu văn bản không giám sát. Dữ liệu đầu vào sẽ được masked (thay bằng một token MASK) một cách ngẫu nhiên với tỷ lệ thấp. Huấn luyện mô hình dự báo được từ masked dựa trên bối cảnh xung quanh là những từ không được masked nhằm tìm ra cách biểu diễn của từ.

Khoảng 15% các token của câu input được thay thế bởi [MASK] token 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) dấu xung quanh [MASK] 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%).

Cách tiếp cận này vẫn tồn tại hai nhược điểm sau khi nhận được mô hình tiền huấn luyện theo hai chiều . Nhược điểm thứ nhất là BERT tạo ra sự không phù hợp giữa quá trình tiền huấn luyện và quá trình tinh chỉnh mô hình , tức là ký tự [MASK] sẽ không bao giờ xuất hiện trong quá trình tinh chỉnh mô hình . Để giảm thiểu vấn đề này , BERT không luôn luôn đánh dấu bằng ký tự [MASK]. Thay vào đó:

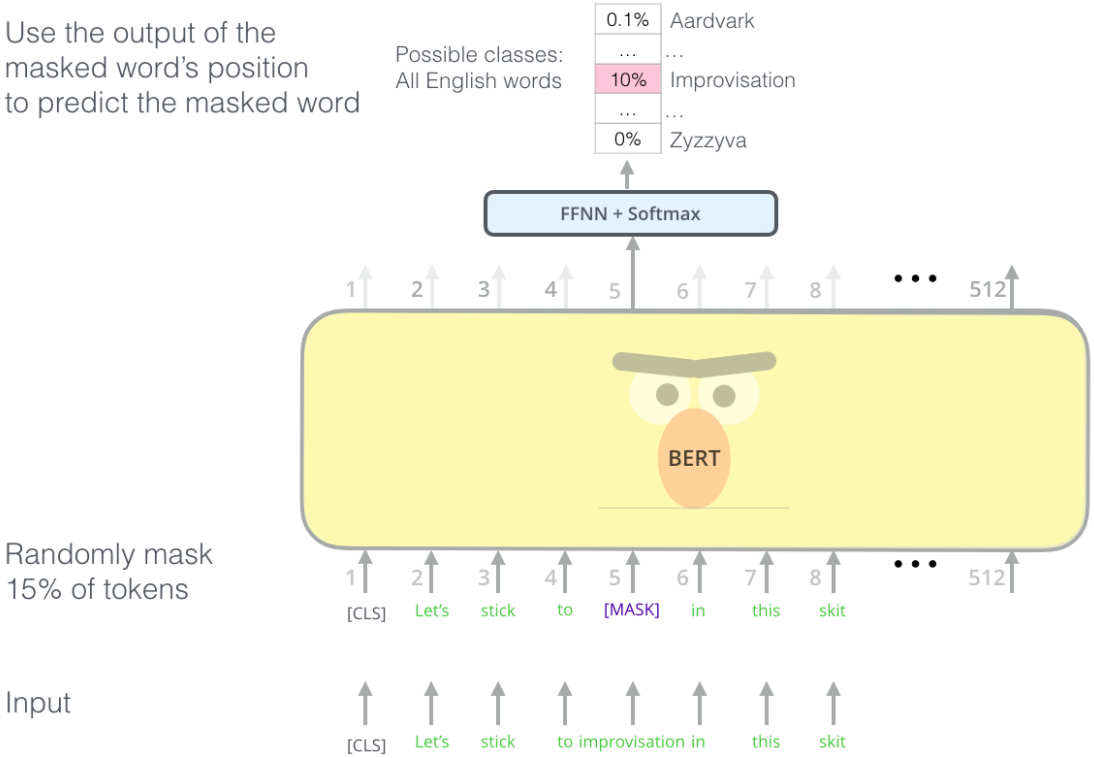
- Thay thế 80% số lượng từ được chọn bằng [MASK].

- Thay thế 10% số lượng từ được chọn bằng một từ bất kỳ khác.

- Giữ nguyên 10% số lượng từ được chọn.

Hàm loss function của BERT sẽ bỏ qua mất mát từ những từ không bị che dấu và chỉ đưa vào mất mát 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. Lớp Tranformer encoder không hề biết trong tập hợp các từ được chọn, từ nào sẽ bị thay thế bằng những từ khác và từ nào sẽ giữ nguyên, do đó mô hình sẽ học được phân phối của tất cả các từ. Thêm vào đó, bởi vì việc thay thế ngẫu nhiên chỉ xảy ra cho 1.5% số lượng các từ nên có vẻ sẽ không ảnh hưởng đến tính phù hợp của việc thu nhận kiến thức cho mô hình ngôn ngữ.

Nhược điểm thứ hai là 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.



Hình 2.4: BERT cho tác vụ Masked Language Model

2.3.2 Next Sentence Prediction

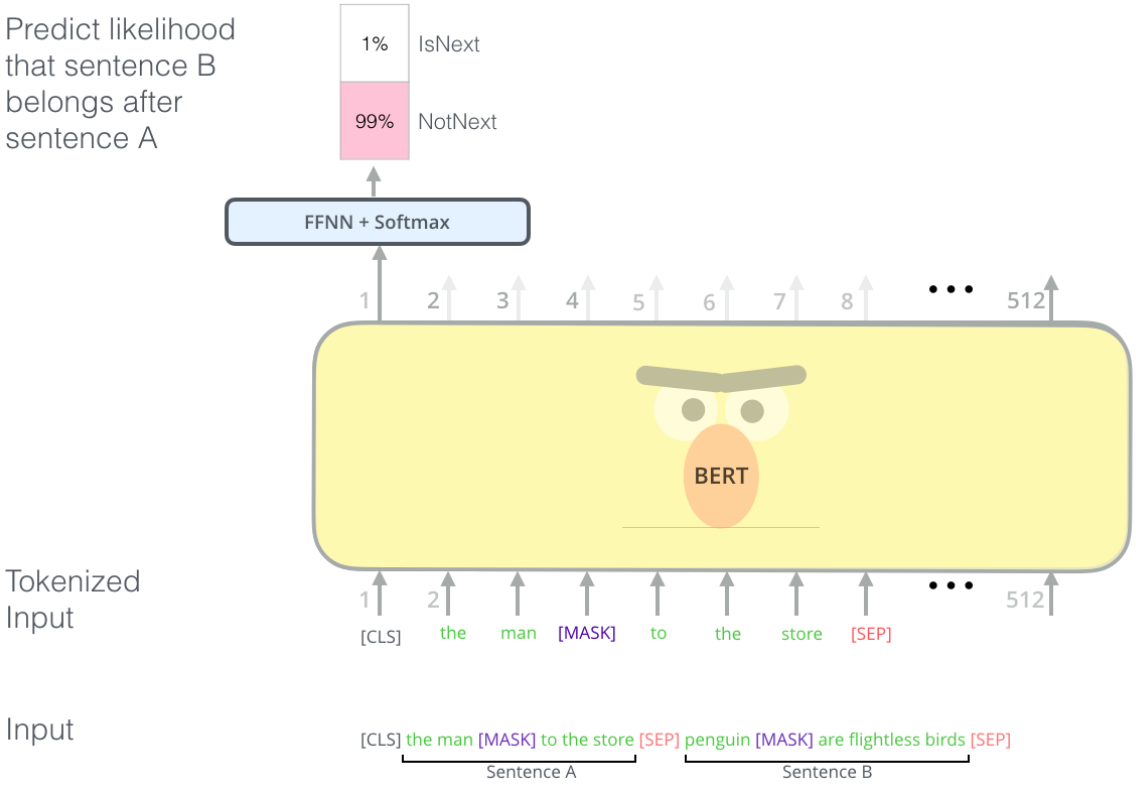
Dây là một bài toán phân loại học có giám sát với 2 nhãn (hay còn gọi là phân loại nhị phân). Input đầu vào của mô hình là một cặp câu (pair-sequence) sao cho 50% câu thứ 2 được lựa chọn là câu tiếp theo của câu thứ nhất và 50% được lựa chọn một cách ngẫu nhiên từ bộ văn bản mà không có mối liên hệ gì 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 mô hình Question and Answering, 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.

Ví dụ 4.

Chúng ta có các đầu vào và nhãn như sau:
Input: [CLS] người đàn_ông làm [MASK] tại cửa_hàng [SEP] anh_ta rất [MASK] và thân_thiện [SEP]
Label: isNext
Input: [CLS] người đàn_ông làm [MASK] tại cửa_hàng [SEP] cô_ta đang cầm súng [SEP]
Label: notNext

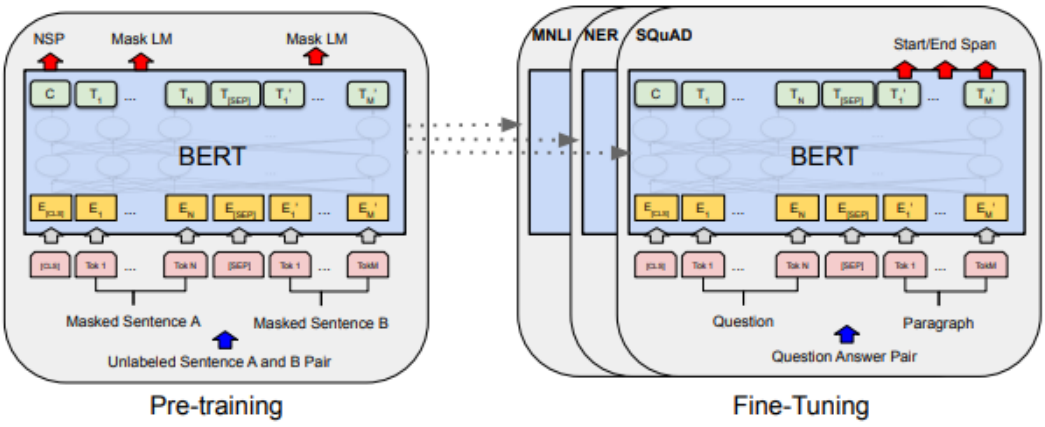


Hình 2.5: BERT cho tác vụ Next sentence prediction

Chúng ta chọn những câu notNext một cách ngẫu nhiên và mô hình cuối cùng đạt được độ chính xác 97%-98% trong nhiệm vụ này.

2.4 Fine-tuning BERT

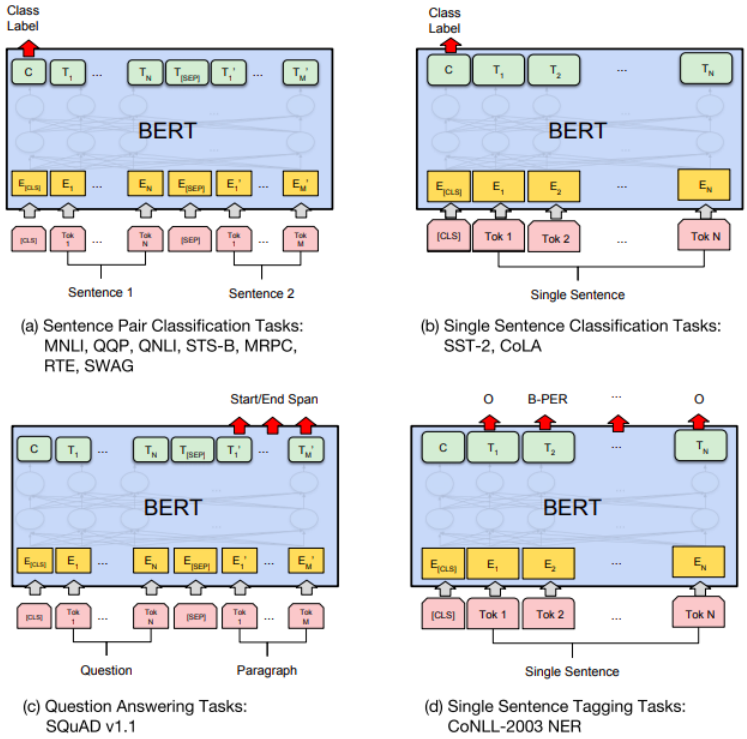
Một điểm đặc biệt ở BERT mà các model embedding trước đây chưa từng có đó là kết quả huấn luyện có thể fine-tuning được. Chúng ta sẽ thêm vào kiến trúc model một output layer để tùy biến theo tác vụ huấn luyện.



Hình 2.6: Tiến trình pre-train và fine-tuning của BERT

Trong suốt quá trình fine-tuning thì toàn bộ các tham số của layers học chuyển giao sẽ được fine-tune. Đối với các tác vụ sử dụng input là một cặp sequence (pair-sequence) ví dụ như question and answering thì ta sẽ thêm token khởi tạo là [CLS] ở đầu câu, token [SEP] ở giữa để ngăn cách 2 câu. Tiến trình áp dụng fine-tuning sẽ như sau:

- **Bước 1:** Embedding toàn bộ các token của cặp câu bằng các véc tơ nhúng từ pretrain model. Các token embedding bao gồm cả 2 token là [CLS] và [SEP] để đánh dấu vị trí bắt đầu của câu hỏi và vị trí ngăn cách giữa 2 câu. 2 token này sẽ được dự báo ở output để xác định các phần Start/End Spand của câu output.
- **Bước 2:** Các embedding véc tơ sau đó sẽ được truyền vào kiến trúc multi-head attention với nhiều block code (thường là 6, 12 hoặc 24 blocks tùy theo kiến trúc BERT). Ta thu được một véc tơ output ở encoder.
- **Bước 3:** Để dự báo phân phối xác suất cho từng vị trí từ ở decoder, ở mỗi time step chúng ta sẽ truyền vào decoder véc tơ output của encoder và véc tơ embedding input của decoder để tính encoder-decoder attention. Sau đó projection qua liner layer và softmax để thu được phân phối xác suất cho output tương ứng ở time step t.
- **Bước 4:** Trong kết quả trả ra ở output của transformer ta sẽ cố định kết quả của câu Question sao cho trùng với câu Question ở input. Các vị trí còn lại sẽ là thành phần mở rộng Start/End Span tương ứng với câu trả lời tìm được từ câu input.



Hình 2.7: Mô hình BERT với các tác vụ khác nhau

Lưu ý quá trình huấn luyện chúng ta sẽ fine-tune lại toàn bộ các tham số của model BERT đã cut off top linear layer và huấn luyện lại từ đầu các tham số của linear layer mà chúng ta thêm vào kiến trúc model BERT để customize lại phù hợp với bài toán.

BERT là một khái niệm đơn giản nhưng lại mang lại hiệu quả cực lớn trong thực tế. Nó đã thu được kết quả tối ưu mới nhất cho 11 nhiệm vụ xử lý ngôn ngữ tự nhiên, bao gồm:

- 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%.

2.5 Các biến thể của BERT

2.5.1 Xlnet

Các mô hình học ngữ cảnh của từ thông thường là những **mô hình ngôn ngữ (language model)**. Thông thường, với những language model cũ sử dụng mạng RNN và các biến thể của RNN được gọi là **autoregressive (AR) language model** tức là những language sử dụng **ngữ cảnh (context)** để dự đoán những từ tiếp theo. Chúng ta có thể sử dụng context forward kết hợp với context backward để dự đoán từ kế tiếp nhưng nó chỉ dùng được một trong hai trong cùng một mốc thời gian, tức là trong một mốc thời gian thì chỉ có forward hoặc backward được sử dụng. AR language model rất tốt trong các bài toán sinh ngôn ngữ vì nó có sử dụng context của từ nhưng lại có một nhược điểm là không thể sử dụng được cả hai ngữ cảnh cùng một lúc.

Không giống như AR language model, BERT là một **Autoencoder (AE) language model**, với mục đích dự đoán những token [MASK], AE language sử dụng ngữ cảnh của cả hai bên của token được mask nên trong cùng một thời điểm BERT có thể sử dụng được cả forward và backward context. Tuy nhiên, có một nhược điểm mà AE language model gặp phải, nếu trong một câu có hai token được mask thì khi dự đoán từ được mask, AE language model coi từ còn lại là một token ký hiệu là [MASK], rõ ràng nếu hai từ này ít có liên hệ và ít quan trọng thì sẽ không có vấn đề gì, nhưng nếu hai từ này có liên hệ mật thiết với nhau và có một vai trò quan trọng trong câu thì AE language model không học ra được mối quan hệ giữa hai từ đó. Xlnet [14] ra đời để giải quyết vấn đề này.

XLNet là một mô hình transformer hai chiều sử dụng phương pháp huấn luyện mô hình được nâng cấp so với BERT. Với dữ liệu huấn luyện lớn hơn, XLNET đạt kết quả vượt trội hơn BERT trên 20 nhiệm vụ ngôn ngữ.

Để cải thiện hiệu quả huấn luyện, XLNET sử dụng mô hình ngôn ngữ hoán vị, tức là các token được dự đoán theo một thứ tự ngẫu nhiên. Điều này khác với mô hình ngôn ngữ sử dụng mặt nạ, với 15% token được che đi để bắt mô hình phải dự đoán. Điều này cũng khác với mô hình ngôn ngữ thông thường, với các token được sắp xếp từ trái sang phải chứ không phải theo thứ tự ngẫu nhiên. Cách huấn luyện này giúp mô hình học được các quan hệ hai chiều cũng như quan hệ phụ thuộc giữa các từ. Bên cạnh đó, Transformer XL là kiến trúc cơ sở của XLNet cũng cho thấy hiệu quả tốt kể cả khi không dùng cách huấn luyện với hoán vị từ. Cách huấn luyện hoán vị của XLNet về mặt lý thuyết có thể nắm bắt tốt các ràng buộc trong câu nên có thể hoạt động tốt với các bài toán yêu cầu cao về ngữ nghĩa.

XLNet được huấn luyện với hơn 130GB dữ liệu text và 512 chip TPU trong 2 ngày rưỡi, các con số này đều lớn hơn so với BERT.

2.5.2 Roberta

RoBERTa [13] là viết tắt của cụm từ **Robustly optimized BERT approach** được giới thiệu bởi Facebook là một phiên bản được huấn luyện lại BERT với một phương pháp huấn luyện tốt hơn với dữ liệu được tăng gấp 10 lần.

Để tăng cường quá trình huấn luyện, RoBERTa không sử dụng cơ chế dự đoán câu kế tiếp (NSP) từ BERT mà sử dụng kỹ thuật mặt nạ động (dynamic masking), theo đó các token mặt nạ sẽ bị thay đổi trong quá trình huấn luyện. Sử dụng kích thước batch lớn hơn cho thấy hiệu quả tốt hơn khi huấn luyện.

Một điều quan trọng nữa, RoBERTa sử dụng 160GB văn bản để huấn luyện. Trong đó, 16GB là sách và Wikipedia tiếng Anh được sử dụng trong huấn luyện BERT. Phần còn lại bao gồm CommonCrawl News dataset (63 triệu bản tin, 76 GB), ngữ liệu văn bản Web (38 GB) và Common Crawl Stories (31 GB). Mô hình này được huấn luyện với GPU của Tesala 1024 V100 trong một ngày. Huấn luyện mô hình lâu hơn, với batch size lớn hơn và trên nhiều dữ liệu hơn.

Roberta cùng kiến trúc với BERT nhưng sử dụng mã hóa BPE để làm tokenizer. Robert không sử dụng Segment Embedding, chúng ta chỉ cần chỉ ra token thuộc phân đoạn nào bằng cách sử dụng token </s>.

Kết quả là, RoBERTa vượt trội hơn cả BERT và XLNet trên dữ liệu đánh giá GLUE:

Model	**data**	**bsz**	**steps**	**SQuAD** (v1.1/2.0)	**MNLI-m**	**SST-2**
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

Hình 2.8: So sánh RoBERTa với XLNet và BERT

2.5.3 Albert

Trung tâm nghiên cứu của Google và Viện công nghệ Toyota đã cùng nhau xuất bản bài báo giới thiệu về một số mô hình được coi là kế thừa của BERT, một mô hình hiệu quả với số lượng tham số ít hơn nhiều. Mô hình này có tên là **ALBERT** (*A Lite BERT*).

Models	SQuAD1.1 dev	SQuAD2.0 dev	SQuAD2.0 test	RACE test (Middle/High)
Single model (from leaderboard as of Sept. 23, 2019)				
BERT-large	90.9/84.1	81.8/79.0	89.1/86.3	72.0 (76.6/70.1)
XLNet	94.5/89.0	88.8/86.1	89.1/86.3	81.8 (85.5/80.2)
RoBERTa	94.6/88.9	89.4/86.5	89.8/86.8	83.2 (86.5/81.3)
UPM	-	-	89.9/87.2	-
XLNet + SG-Net Verifier++	-	-	90.1/87.2	-
ALBERT (1M)	94.8/89.2	89.9/87.2	-	86.0 (88.2/85.1)
ALBERT (1.5M)	**94.8/89.3**	**90.2/87.4**	**90.9/88.1**	**86.5 (89.0/85.5)**
Ensembles (from leaderboard as of Sept. 23, 2019)				
BERT-large	92.2/86.2	-	-	-
XLNet + SG-Net Verifier	-	-	90.7/88.2	-
UPM	-	-	90.7/88.2	
XLNet + DAAF + Verifier	-	-	90.9/88.6	-
DCMN+	-	-	-	84.1 (88.5/82.3)
ALBERT	**95.5/90.1**	**91.4/88.9**	**92.2/89.7**	**89.4 (91.2/88.6)**

Hình 2.9: Kết quả Albert trên SQuAD và RACE

Albert đạt kết quả vượt trội trên nhiều bài toán với tham số ít hơn. Kết quả này đến từ những nâng cấp trong kiến trúc và cách huấn luyện mô hình. Dưới đây là bảng các kiến trúc và số tham số của Albert so với BERT:

Model		Parameters	Layers	Hidden	Embedding	Parameter-sharing
BERT	base	108M	12	768	768	False
	large	334M	24	1024	1024	False
ALBERT	base	12M	12	768	128	True
	large	18M	24	1024	128	True
	xlarge	60M	24	2048	128	True
	xxlarge	235M	12	4096	128	True

Hình 2.10: Kiến trúc của Albert và Bert

Nhìn vào bảng ta có thể thấy các phiên bản của ALBERT đều nhỏ gọn hơn BERT. Phiên bản xlarge của ALBERT chỉ xấp xỉ một nửa số tham số của phiên bản base và nhỏ hơn 21.5 lần so với phiên bản xlarge của BERT, đây là một con số ấn tượng.

Trong những năm vừa qua, cùng với sự ra đời của các mô hình Transformer có kích thước ngày một lớn hơn, ta chứng kiến được sự cải thiện từng ngày về độ chính xác của các mô hình trong các bài toán NLP. Trong bài báo gốc của BERT, các tác giả chứng minh rằng với càng nhiều nơ ron, càng nhiều lớp ẩn, càng nhiều đầu attention thì kết quả càng tốt.

Chạy theo xu hướng đó, mô hình NLP lớn nhất cho đến nay là của NVIDIA với tên gọi Megatron, mô hình này chứa 8 tỷ tham số, gấp 24 lần BERT và 6 lần GPT-2. Mô hình này được huấn luyện trong 9 ngày với 512 GPU.

Tác giả của ALBERT chỉ ra trong thực nghiệm của họ trên RACE, rằng phiên bản xlarge của BERT thực tế đã giảm hiệu quả đến 20% so với phiên bản large của mô hình này dù có số lượng tham số gấp 4 lần.

Điều này tương tự với hiệu quả của việc tăng độ sâu trong thị giác máy. Tăng cường độ sâu ban đầu có thể giúp tăng kết quả dự đoán nhưng nếu tiếp tục tăng, kết quả sẽ bắt đầu giảm. Ví dụ, một mạng ResNet với 1000 lớp không hiệu quả hơn một mạng ResNet với 152 lớp dù số lớp gấp đến 6.5 lần. Nói một cách khác, tại điểm bão hòa của dữ liệu, kết quả dự đoán sẽ không tăng dù cho năng lực của mạng được tăng cường.

Với quan sát đó, tác giả của ALBERT muốn tạo ra kết quả tốt hơn dựa trên kiến trúc của mô hình và phương pháp huấn luyện, hơn là chỉ tạo ra một phiên bản lớn hơn của BERT.

Kiến trúc lõi của ALBERT giống với BERT, đều sử dụng kiến trúc của encoder của transformer với hàm kích hoạt GELU. Trong bài báo, các tác giả cũng sử dụng vocabulary có kích thước 30K giống với BERT. So với BERT, ALBERT có 3 điểm khác biệt quan trọng (2 cải tiến trong kiến trúc và 1 cải tiến trong phương pháp huấn luyện):

Hai cải tiến trong kiến trúc

1. **Embedding có trọng số**: Các tác giả của ALBERT chỉ ra rằng, đối với BERT, XLNet và RoBERTa, kích thước của WordPiece Embedding (E) bị ràng buộc với kích thước lớp ẩn (H). Mặc dù vậy, theo họ, embedding của đầu vào được thiết kế để học các biểu diễn không phụ thuộc vào ngữ cảnh trong khi đó các lớp ẩn có nhiệm vụ học các biểu diễn phụ thuộc vào ngữ cảnh.
   Sức mạnh của BERT phần lớn nằm ở khả năng học các biểu diễn phụ thuộc ngữ cảnh bởi lớp ẩn. Nếu H và E bị ràng buộc với nhau, với một bài toán yêu cầu vocabulary (V) lớn, kích thước lớp ẩn cũng vì thế mà tăng lên. Kết quả là ta sẽ có một mô hình có hàng tỷ tham số mà phần lớn trong số đó hiếm khi được cập nhật trong quá trình huấn luyện. Do đó, ràng buộc hai thành phần có các mục đích khác nhau làm giảm hiệu quả của các tham số.
   Cải thiện điều này, ALBERT thực hiện embedding với hai ma trận con. Thay vì ánh xạ các véc tơ embedding trực tiếp đến các lớp ẩn (tức là E = H). Thì chúng được ánh xạ đến một ma trận ít chiều hơn, sau đó mới ánh xạ ma trận này tới các lớp ẩn. Như vậy, số tham số được giảm từ O(V*H) xuống còn O(V*E+E*H).

2. **Chia sẻ tham số giữa các lớp**: Cải tiến thứ hai của ALBERT là phương thức chia sẻ các tham số giữa tất cả các lớp. Cụ thể, các tham số được dùng chung là các tham truyền thẳng và tham số attention. Các tác giả cho rằng việc dùng chung các tham số như vậy giúp thông tin lưu chuyển giữa các lớp thông suốt

hơn và giúp mạng trở nên ổn định hơn.

Thay đổi cách huấn luyện - SOP (Sentence Order Prediction)

Hai cơ chế trong cách huấn luyện của BERT giúp mô hình này tạo ra sự khác biệt là MLM và NSP. ALBERT kế thừa MLM của BERT nhưng thay thế NSP với một cơ chế khác mang tên SOP (Sentence Order Prediction – Dự đoán thứ tự câu).

Có một điểm cần chú ý rằng, các tác giả của RoBERTa chỉ ra rằng việc không sử dụng NSP trong huấn luyện cũng không làm giảm hiệu quả mô hình. Các tác giả của ALBERT đã lập luận để chứng minh tại sao NSP không hiệu quả và phát triển SOP.

	Intrinsic Tasks			Downstream Tasks					
SP tasks	MLM	NSP	SOP	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
None	54.9	52.4	53.3	88.6/81.5	78.1/75.3	81.5	89.9	61.7	79.0
NSP	54.5	90.5	52.0	88.4/81.5	77.2/74.6	81.6	**91.1**	62.3	79.2
SOP	54.0	78.9	86.5	**89.3/82.3**	**80.0/77.1**	**82.0**	90.3	**64.0**	**80.1**

Hình 2.11: So sánh hiệu quả của SOP

Các tác giả của ALBERT lập luận rằng NSP đã vô tình kết hợp năng lực dự đoán chủ đề và năng lực đánh giá tính mạch lạc của mô hình. Cụ thể, NSP cố gắng phát hiện hai câu liền nhau trong cùng một tài liệu với các phản ví dụ là các cặp câu ở hai tài liệu khác nhau.

Trái lại, các tác giả của ALBERT cho rằng tính mạch lạc giữa các câu là quan trọng hơn việc phát hiện chủ đề và SOP được phát triển để giúp mô hình có năng lực dự đoán thứ tự các câu. Các cặp câu được lấy ra từ cùng một tài liệu và phản ví dụ là trường hợp các câu này được đổi chỗ cho nhau. Nhờ đó, mô hình sẽ không dựa vào thông tin chủ đề mà chỉ tập trung vào tính liên kết giữa hai câu để học ra thứ tự giữa chúng.

Sự thành công của ALBERT đã cho thấy tầm quan trọng của việc xác định các yếu tố mô hình giúp phân tích ngữ cảnh hiệu quả hơn. Thông qua việc tập trung vào các yếu tố này, mô hình sẽ được cải thiện về cả hiệu suất lẫn hiệu quả khi thực hiện các tác vụ NLP

Chương 3

Thực nghiệm

3.1 Giới thiệu bài toán

Với tất cả các bình luận lưu hành trên internet, thật khó để có thể biết đằng sau một bình luận, cảm xúc đó sẽ ảnh hưởng như thế nào đến công ty hoặc của một cá nhân nào đó. Vì nó có tính lan truyền. Vì vậy, việc nắm bắt được cảm sắc thái bình luận là việc quan trọng cho mỗi cá nhân và công ty để đưa ra quyết định và xử lý đúng đắn trong từng giai đoạn. Tuy nhiên, vấn đề đặt ra là liệu thực sự cụm từ nào của câu mới đem lại cảm xúc cho toàn câu. Bài toán đặt ra là trích xuất từ hoặc cụm từ (gọi là tweet) phản ánh cảm xúc trong câu.

Dữ liệu được lấy từ cuộc thi **Tweet sentiment extraction** tại: `https://www.kaggle.com/c/tweet-sentiment-extraction`. Dữ liệu huấn luyện gồm 27464 bình luận và tweet tương ứng của nó. Các trường trong dữ liệu gồm:

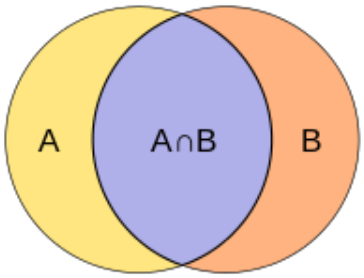
1. **textID**: ID duy nhất cho mỗi bình luận/văn bản.
2. **text**: Văn bản đầy đủ.
3. **sentiment**: Tâm lý/Cảm xúc chung của bình luận.
4. **selected_text**: [Nhãn] Văn bản chứa từ/cụm từ trong câu phản ánh cảm xúc.

	textID	text	selected_text	sentiment
0	cb774db0d1	I`d have responded, if I were going	I`d have responded, if I were going	neutral
1	549e992a42	Sooo SAD I will miss you here in San Diego!!!	Sooo SAD	negative
2	088c60f138	my boss is bullying me...	bullying me	negative
3	9642c003ef	what interview! leave me alone	leave me alone	negative
4	358bd9e861	Sons of ****, why couldn`t they put them on t...	Sons of ****,	negative

Hình 3.1: Một phần của bảng dữ liệu huấn luyện

Kết quả được đánh giá trên độ đo **Jaccard** cấp độ từ, còn được gọi là **Intersection over Union** hay **Jaccard similarity coefficient** đo sự tương tự giữa 2 tập hợp A và B như sau:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$



Hình 3.2: Độ đo jaccard

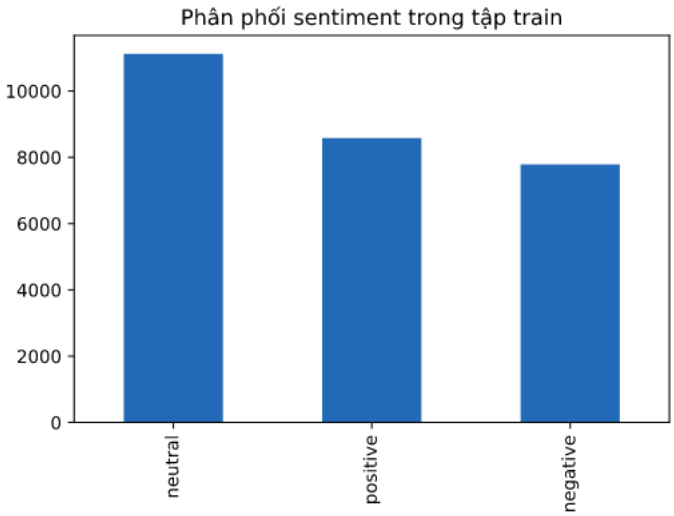
Triển khai trên Python:

```
def jaccard(str1, str2):
    a = set(str1.lower().split())
    b = set(str2.lower().split())
    c = a.intersection(b)
    return float(len(c)) / (len(a) + len(b) - len(c))
```

Hình 3.3: Triển khai hàm đo Jaccard trên Python

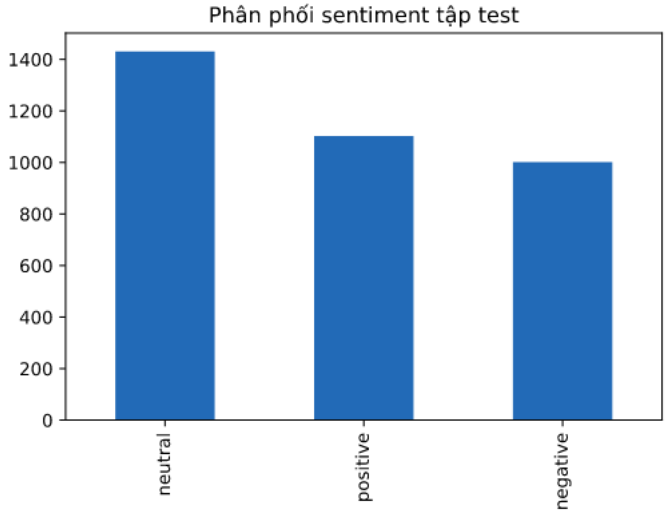
3.2 Phân tích dữ liệu

Phân phối cảm xúc ở tập huấn luyện:



Hình 3.4: Phân phối cảm xúc ở tập huấn luyện

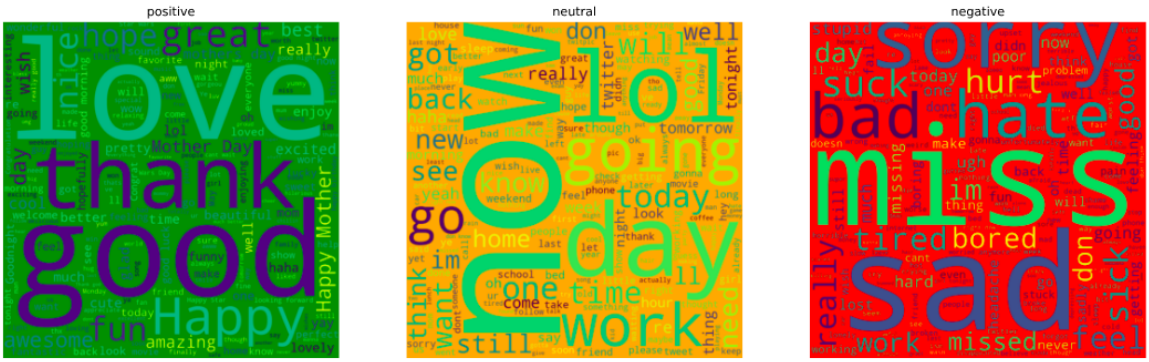
Phân phối cảm xúc ở tập kiểm thử:



Hình 3.5: Phân phối cảm xúc ở tập kiểm thử

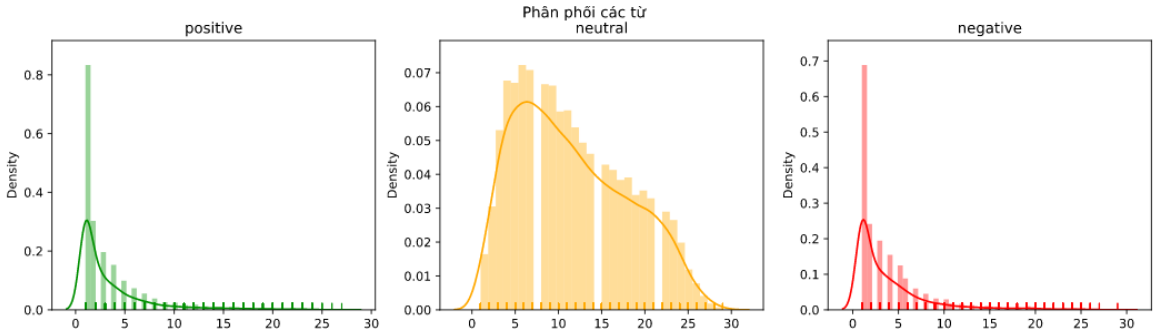
Có thể thấy là dữ liệu khá đẹp và tương đồng giữa tập train và tập test. Do cuộc thi đã kết thúc và dữ liệu test không được public, và cũng không thể submit kết quả để kiểm thử được. Vì vậy trong bài toàn này, chúng ta tạm sử dụng tập train để chia thành validation và train cho việc kiểm tra độ tốt của mô hình trên tập validation. Do dữ liệu ít nên ta sử dụng thêm phương pháp K-fold (cụ thể báo cáo này dùng 5-Fold) để chia và huấn luyện mô hình để đảm bảo tránh overfitting.

Top các từ phổ biến tương ứng với từng trạng thái cảm xúc được biểu diễn hình sau:



Hình 3.6: Top các từ phổ biến tương ứng với từng trạng thái cảm xúc

Phân phối các từ selected text:



Hình 3.7: Phân phối các từ selected text

3.3 Xây dựng mô hình và đánh giá kết quả

3.3.1 Tổng quan về mã nguồn

Mã nguồn cho báo cáo này được lưu trữ tại: `https://github.com/nducthang/BERT-Tweet-Sentiment-Kaggle` , bao gồm code cho 3 mô hình là Bert, Roberta, ALBert sử dụng framework Pytorch. Trong mã nguồn sử dụng 5 Fold để chia và huấn luyện mô hình, mỗi fold huấn luyện với 3 epochs và lưu trọng số mô hình lại để dùng cho việc kiểm thử.

Các thư mục

- **data**: Chứa dữ liệu cho dạng .csv cho việc huấn luyện và kiểm thử.
- **utils**: Chứa các file .py là các hàm hỗ trợ. Trong code đã comment ý nghĩa của các hàm và các file trong thư mục này.
- **weight**: Gồm 3 thư mục con là bert, roberta, albert. Mỗi thư mục chứ weight của 5 Fold đã huấn luyện tương ứng với mô hình đó.

Các file mô hình

Với mỗi mô hình **[XYZ]** ([XYZ] = Albert, Bert, Roberta) sẽ có 4 file ngoài thư mục root:

- **[XYZ]Run.py**: file chạy để thực hiện việc huấn luyện mô hình.
- **[XYZ]TrainModel.py**: Chứa hàm cho quá trình huấn luyện mô hình.
- **[XYZ]TweetDataset.py**: Chứa class cho việc tạo và load Dataset.
- **[XYZ]TweetModel.py**: Chứa class của kiến trúc mô hình.

Cài đặt môi trường

Ngoài các thư viện phổ biến trong ngôn ngữ Python thì cần cài đặt **tokenizers==0.8.1rc2** và **transformers==3.3.1**, **pytorch** (GPU) để chạy được mã nguồn. Chú ý là tokenizers và transformers phải là các phiên bản trên để đảm bảo tương thích.

Clone mã nguồn và huấn luyện

Sau khi đã cài đặt các thư viện cần thiết, tiến hành chạy code để huấn luyện mô hình như sau:
Clone mã nguồn:

```
git clone https://github.com/nducthang/BERT-Tweet-Sentiment-Kaggle
```

Huấn luyện:

```
python3 [XYZ]Run.py
```

Ví dụ: `python3 BertRun.py`

3.3.2 Kết quả thực nghiệm

Biểu diễn dữ liệu đầu vào

Với mô hình BERT, cấu trúc BERT tokenizer dựa trên WordPiece. Giả sử ta có:

```
text = "NLP is very hard"
selectec_text = "very hard"
sentiment = "negative"
```

Thì biểu diễn đầu vào cho huấn luyện của BERT được biểu diễn như sau:

tokens	[CLS]	negative	[SEP]	nl	##p	is	very	hard	!	[SEP]	[PAD]	[PAD]
ids	101	4997	102	17953	2361	2003	2200	2524	999	102	0	0
attention_mask	1	1	1	1	1	1	1	1	1	1	0	0
token_type_ids	0	0	0	1	1	1	1	1	1	1	0	0
start_idx	0	0	0	0	0	0	1	0	0	0	0	0
end_idx	0	0	0	0	0	0	0	1	0	0	0	0

Hình 3.8: Biểu diễn dữ liệu đầu vào cho BERT

Với Roberta, kiến trúc Roberta Tokenizer sử dụng byte-level Byte-Pair-Encoding.

tokens	<s>	negative	</s>	</s>	Ġn	lp	Ġis	Ġvery	Ġhard	!	</s>	<pad>	<pad>
ids	0	2430	2	2	295	39031	16	182	543	328	2	1	1
attention_mask	1	1	1	1	1	1	1	1	1	1	1	0	0
token_type_ids	0	0	0	0	1	1	1	1	1	1	1	0	0
start_idx	0	0	0	0	0	0	0	1	0	0	0	0	0
end_idx	0	0	0	0	0	0	0	0	1	0	0	0	0

Hình 3.9: Biểu diễn dữ liệu đầu vào cho RoBerta

Với ALBert, kiến trúc ALBert Tokenizer sử dụng SentencePiece.

tokens	[CLS]	negative	[SEP]	_	n	lp	_is	_very	_hard	!	[SEP]	<pad>	pad>
ids	2	3682	3	13	103	5478	25	253	552	187	3	0	0
attention_mask	1	1	1	1	1	1	1	1	1	1	1	0	0
token_type_ids	0	0	0	1	1	1	1	1	1	1	1	0	0
start_idx	0	0	0	0	0	0	0	1	0	0	0	0	0
end_idx	0	0	0	0	0	0	0	0	1	0	0	0	0

Hình 3.10: Biểu diễn dữ liệu đầu vào cho ALBert

Kiến trúc mô hình

Mô hình được đưa về như một bài toán phân lớp 2 nhánh để tìm vị trí bắt đầu (start logits) và vị trí kết thúc (end logits) của cụm từ cần trích xuất. Dưới đây là kiến trúc lan truyền tiến cho Bert, tương tự với các mô hình Albert và Roberta.

```
def forward(self, input_ids, attention_mask, token_type_ids):
        # Đầu vào Bert cần chỉ số các token (input_ids)
        # Và attention_mask (Mặt nạ biểu diễn câu 0 = pad, 1 = otherwise)
        # Và token_type_ids
        _, _, hs = self.bert(input_ids, attention_mask, token_type_ids)

        # len(hs) = 13 tensor, mỗi tensor shape là (1, 128, 768)
        x = torch.stack([hs[-1], hs[-2], hs[-3], hs[-4]])
        # x shape (4,1,128,768)
        x = torch.mean(x, 0)
        # x shape (1,128,768)
        x = self.dropout(x)
        x = self.fc(x)
        # x shape (1,128,2)
        start_logits, end_logits = x.split(1, dim=-1)

        # Nếu số chiều cuối là 1 thì bỏ đi (1,128,1) -> (1,128)
        # Ví dụ (AxBxCX1) --> size (AxBxC)
        start_logits = start_logits.squeeze(-1)
        end_logits = end_logits.squeeze(-1)

        return start_logits, end_logits
```

Hình 3.11: Kiến trúc lan truyền tiến của BERT

Các mô hình được cài đặt với các tham số giống hệt nhau để so sánh, độ dài tối đa cho câu đầu vào là 128, sử dụng Adam để tối ưu. Dữ liệu chia thành 5-Fold và mỗi fold được huấn luyện với 3 epoch tương ứng với mỗi mô hình.

Kết quả

Quá trình huấn luyện và đo được thực hiện trên Google Colab GPU. Các thông số jaccard được đo trên dữ liệu kiểm thử.

#	**Bert (base)**	**Roberta (base)**	**AlBert (base v2)**
Jaccard Fold 1	0.6949	0.7068	0.7028
Jaccard Fold 2	0.7026	0.7117	0.7003
Jaccard Fold 3	0.6937	0.7045	0.6881
Jaccard Fold 4	0.6924	0.6922	0.6939
Jaccard Fold 5	0.6950	0.7021	0.7026
Jaccard Mean	0.69572	0.70346	0.69754
Time train/epoch (phút)	33:11	31:41	15:58

Bảng 3.1: Kết quả thực nghiệm

KẾT LUẬN

Quá báo cáo, em đã trình bày về mô hình BERT và các mở rộng, biến thể của BERT. Cho đến thời điểm viết báo cáo này, BERT đã ra đời được 2 năm những vẫn đang là một chủ đề nóng trong lĩnh vực NLP hiện tại. Hàng năm, rất nhiều cải tiến và biến thể của BERT đang và tiếp tục được công bố.

Có thể thấy, BERT khá tốt khi áp dụng vào nhiều bài toán NLP. Kết quả thực nghiệm cũng đánh giá đúng bản chất và lý thuyết của các mô hình. Vì tài nguyên hạn chế nên thực nghiệm chỉ chạy được 3 epoch trên mỗi Fold và chưa thực nghiệm được với nhiều sự thay đổi về các tham số khác. Từ bảng kết quả chúng ta có thể thấy rằng, với cùng một kiến trúc mô hình thì Roberta đem lại kết quả tốt nhất cho bài toán, Albert thì lại có số lượng tham số nhỏ nhất và huấn luyện nhanh nhất khi thời gian huấn luyện không bằng 1 nửa thời gian huấn luyện của Bert và Roberta. Vì vậy, nếu ưu tiên về độ hiệu quả chúng ta có thể lựa chọn Roberta. Nếu tài nguyên và thời gian hạn hẹp và đánh đổi một chút hiệu quả thì có thể lựa chọn AlBert. Tuy nhiên, không vì thế mà Bert không được lựa chọn trong các bài toán, Bert vẫn đem lại kết quả tốt không thua kém các mô hình cải tiến của nó sau này, nó vẫn là nền tảng để cho các mô hình sau này phát triển, là cốt lõi ý tưởng.

Tuy nhiên, cũng có thể thấy rằng, các mô hình trên đều vẫn chưa đạt hiệu quả đủ để áp dụng vào thực tế cũng như các ứng dụng thời gian thực mặc dù BERT được coi như là bước ngoặt lớn của ngành xử lý ngôn ngữ. Để cải thiện, chúng ta cần có những ý tưởng đột phá hơn nữa. Xử lý ngôn ngữ tự nhiên vẫn đang tiếp tục phát triển những bước đi đầu, nên có thể hy vọng rằng trong tương lai không xa, các mô hình mới sẽ tốt hơn. Đây vẫn là những vấn đề nghiên cứu mở. Hy vọng có thể nhận được sự đóng góp của thầy cô và các bạn để báo cáo được hoàn thiện tốt hơn.

Tài liệu tham khảo

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Colin Raffel and Daniel PW Ellis. Feed-Forward networks with attention can solve some long-term memory problems. *arXiv preprint arXiv:1512.08756*, 2016

[3] Graves, A., Wayne, G. & Danihelka, I. Neural Turing machines. *http://arxiv.org/abs/1410.5401*, 2014.

[4] Luong, M.-T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. *In Conference on Empirical Methods in Natural Language Processing*, 2015.

[5] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *In ICML*, 2015.

[6] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. *In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[8] Diedelrik P.Kingma, Jimmy Lei Ba, Adam: A method for stochastic optimization, *ICLR* , 2015.

[9] Sashank J.Reddi, Satyen Kate, Sanjiv Kumar, On the convergence of adam and beyond,*ICLR*, 2018.

[10] Sutskever, I., Vinyals, O., and Le, Q. Sequence to sequence learning with neural networks. *In Advances in Neural Information Processing Systems*, (NIPS 2014).

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, *arXiv preprint arXiv:1810.04805*, 2018.

[12] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[13] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[14] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. XLNet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

[15] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.